MJ-4000 MICRJET®

Driver API

Specification

Addmaster Corporation

Address: 225 East Huntington Drive
Monrovia, CA 91016

Web: www.addmaster.com
Phone: (626) 358-2395
FAX: (626) 358-2784

Document: mj4000 Software API rev100.doc (source)
mj4000 Software API rev100.pdf (pdf)

Revisions: 1.00 May 23, 2005

Table of Contents

1 Overview and General Description 3
2 Functions 4
2.1 Function Overview 4
2.1 Function Detail 5
3 Using the DLL 22
3.1 Coding Overview 22
3.2 Header File - MJ-4000.h 23
3.3 Loading & Linking 26
4 Document Revision Information 27

1 Overview and General Description

To be written.

2 Functions

2.1 Function Overview

An overview of the available functions is provided in the table below.

Function	***Usage***
Init_Scanner	Initialize Scanner and Interface
Reset_Scanner	Reset Scanner & Clear any pending jobs.
Get_Scanner_Info	Retrieve model, firmware, revision numbers.
Get_Scanner_Capabilities	Retrieve scanner capabilities
Get_Scanner_Statistics	Retrieve usage statistics from scanner.
Get_Scanner_Status	Retrieve current status - Form insertion, pending job, jams, etc.
Set_Scanner_Parameters	Sets scanner operational modes.
Set_ImageData_Format	Sets data image formatting.
Set_ImageData_Tags	Can tag image with various fields.
Set_Validation	Sets current validation message and type.
Scanner_Cmd	Commands scanner action.
Parse_MICR	Parse MICR line into its parts.
Write_Data_Files	Writes out images as required.

2.1 Function Detail

Init_Scanner

Function: Initialize Scanner and Interface

Syntax: int Init_Scanner(void);

Remarks: Init_Scanner opens a communication path to the scanner, initializes communications, and resets the scanner. Init_Scanner must be called before the scanner can be accessed by user software.

Returns: Returns one of the following values.
MJ_INIT initialized
MJ_NOTINIT not initialized, not attached, or unknown error

Examples:

```
#include “mj4000.h”
int main(void) {

	if (init_Scanner() == MJ_INIT ) {
		printf(“ Scanner is initialized and ready.\n” );
	} else {
		printf(“ Scanner is not ready for use.\n” );
	}
	return 0;
}
```

Reset_Scanner

Function: Reset Scanner & Clear any pending jobs.

Syntax: int Reset_Scanner(void);

Remarks: Reset_Scanner will reset the scanner and reset communications.
Reset_Scanner can be called to abort pending c

Returns: Returns one of the following values.

MJ_RESET	reset successful
MJ_NOTRESET	not successful, not attached, or unknown error

Examples:

```
#include “mj4000_dl.h”
int main(void) {

	/*  scanner code here */

	if (Reset_Scanner() == MJ_RESET ) {
		printf(“ Scanner successfully reset and is ready.\n” );
	} else {
		printf(“ Error - Scanner is not ready for use.\n” );
	}
	return 0;
}
```

Get_Scanner_Info

Function: Retrieve model, firmware, revision numbers.

Syntax: int Get_Scanner_Info(int *infotype*, char **results*);

Remarks: Retrieve information from the attached device concerning its construction. *infotype* specifies the requested string and **results* points to a character string to contain the results. Strings can be up to 64 characters long.

Supported *infotype* values:

infotype	*Define*	*String returned*
0	INFO_MOD	Model
1	INFO_MODNO	Model Name
2	INFO_VER	Firmware Version Number
3	INFO_FTVER	Font Version Number
7	INFO_MFG	Manufacturer
9	INFO_TEMP0	Temporary String #0
12	INFO_USER1	User String #1
13	INFO_USER2	User String #2
14	INFO_FACT	Factory String #1

Returns: Returns one of the following values.

MJ_COMMOK communications successful

MJ_NOTINIT not successful, not attached, or unknown error

Examples:

```
#include "mj4000_dl.h"
int main(void) {
char   msgbuf[64];

       Init_Scanner();     /* presume successful */

       if (Get_Scanner_Info(2, msgbuf) == MJ_COMMOK ) {
              printf( "Firmware Version = %s.\n", msgbuf );
       } else {
              printf(" Error - Scanner is not ready for use.\n" );
       }
       return 0;
}
```

Get_Scanner_Capabilities

Function: Retrieve scanner capabilities

Syntax: int Get_Scanner_Capabilities(struct CapScanner *);

Remarks: Retrieve a list of the capabilities of the attached scanner.

Returns: Returns one of the following values.

MJ_COMMOK	communications successful
MJ_NOTINIT	not successful, not attached, or unknown error

Sets the components of the referenced structure.

```
Struct CapScanner {
      int    Scansides;    /* number of sides scanned         */
      int    color;        /* color support, 0=no             */
      int    bitdepth;     /* bits of gray scale, 1=binary    */
      int    xdpi;         /* x resolution, dots per inch     */
      int    ydpi;         /* y resolution, dots per inch     */
      int    x_maxpixels;  /* max pixels scanned in x         */
      int    y_maxpixels;  /* max pixels scanned in y         */
      int    MICR;         /* MICR support, 1=yes, 0=no       */
      int    Validation;   /* Validation - number of lines    */
      int    Journal;      /* Journal, 1=yes, 0=no            */
      int    Magcard;      /* Magcard, 1=yes, 0=no            */
}
```

Examples:

```
#include “mj4000_dl.h”
int main(void) {
struct CapScanner mjcaps;

      Init_Scanner();     /* presume successful */

      if (Get_Scanner_Capabilities( &mjcaps ) == MJ_COMMOK ) {
            printf( “Scanner xdpi = %d.\n”, mjcaps.xdpi );
      } else {
            printf(“ Error - Scanner is not ready for use.\n” );
      }
      return 0;
}
```

Get_Scanner_Statistics

Function: Retrieve usage statistics from scanner.

Syntax: int Get_Scanner_Statistics(int statno, long * value);

Remarks: Returns various internal counters specified by *statno.*

statno	*Define*	*Description*
0	STAT_POR	Number of power-on resets.
9	STAT_DOTS	Number of ink-dots printed.
11	STAT_FORMS	Number of forms validated.
16	STAT_MICRDOCS	Number of MICR document read attempts.
24	STAT_SCANDOCS	Number of documents scanned.
Reserved		Others to be determined.

Returns: Returns one of the following values.

MJ_COMMOK communications successful
MJ_NOTINIT not successful, not attached, or unknown error

Returns long value for specified statistic.

Examples:

```
#include "mj4000_dl.h"
int main(void) {
long docs_scanned;

      Init_Scanner();     /* presume successful */

      if (Get_Scanner_Statistics( &docs_scanned ) == MJ_COMMOK ) {
            printf( "Documents scanned = %ld.\n", docs_scanned );
      } else {
            printf(" Error - Scanner is not ready for use.\n" );
      }
      return 0;
}
```

Get_Scanner_Status

Function: Retrieve current status.

Syntax: int Get_Scanner_Status(int statno, int * mechstatus);

Remarks: Retrieves status from the scanner specifying its current state of operation. *statno* specifies the type of status requested.

statno	*Description*
0	Mechanism status.
Reserved	Others to be determined.

Returns: Returns one of the following values.

MJ_COMMOK communications successful
MJ_NOTINIT not successful, not attached, or unknown error

Returns integers specific to each *statno* as specified below.

For Statno = 0 -- Mechanism Status

Mechstatus Bit position	*Description*
15-8	Reserved 0
7	PWRDWN - 1=scanner in power down mode.
6	BEMP- receive buffer empty
5	TEMP - transmit buffer empty
4	PINIT - 1=scanner initialized bit, user settable
3	PERR - 1=printer validation error
2	VMP - 1=Valid Command/Message Pending
1	READY - 1=scanner ready to execute commands
0	FORM - 1=form inserted, 0=no form inserted.

Examples:

```
#include “mj4000_dl.h”
int main(void) {
int mechstatus;

      Init_Scanner();     /* presume successful */

      Get_Scanner_Status(0, &mechstatus );
      if ( mechstatus & 0x0001 ) {
            printf( “Document inserted.\n” d );
      } else {
            printf( “No Document inserted in scanner.\n” );
      }
      return 0;
}
```

Set_Scanner_Parameters

Function: Sets scanner operational modes.

Syntax: int Set_Scanner_Parameters(struct ScanParameters *);

Remarks: Modifies the default settings of the attached scanner.

This command is not yet implemented.

Returns: Returns one of the following values.

MJ_COMMOK	communications successful
MJ_NOTINIT	not successful, not attached, or unknown error

Examples:

This command is not yet implemented.

Set_ImageData_Format

Function: Sets data image and MICR-line formatting.

Syntax: void Set_Data_Format(struct scan_fmt *);

Remarks: This function sets the data formats that should be used when the imaged data is returned to memory after a scan. This function should be called before the Scanner_Cmd is called to process a document. It may be called only once, as settings are persistent.

At present, only FMT_TIFF4 is supported. Other functionality will be added at a later time.

```
Struct scan_fmt {
        int     Filefmt;            // see below
        int     Scansides;          // 1= side A, 2 = sides A + B
        int     color;              // 1 = color,  0=B/W imaging
        int     bitdepth;           // bits stored in memory (not bits scanned)
        int     xdpi;               // reserved
        int     ydpi;               // reserved
        int     x_maxpixels;        // reserved
        int     y_maxpixels;        // reserved
}
```

Filefmt = 0 = FMT_RAW = Addmaster proprietary raw data format
= 16 = FMT_BMP = windows bmp format
= 32 = FMT_TIFF = tiff uncompressed
= 33 = FMT_TIFF4 = tiff, g4 compression

Returns: void return.

Examples:

To store both sides of a document in TIFF G4 compressed format the following code is used.

```
#include "mj4000_dl.h"
int main(void) {
scan_fmt setting;

        Init_Scanner();     /* presume successful */
        /* more code here */

        setting.Filefmt = FMT_TIFF4;
        setting.Scansides = 2;
        setting.color = 0;      setting.bitdepth = 1;
        setting.xdpi = 200;     setting.ydpi = 200;
        Set_ImageData_Format( setting );

        return 0;
}
```

Set_ImageData_Tag

Function: Attaches tags to image data for future use in storing image.

Syntax: void Set_ImageData_Tag(int tag_no, int tag_type, char * tag_val);

Remarks: **Not supported at present. Functionality will be added at a later time.**

Returns: void return.

Examples:

Set_Validation

Function: Sets current validation message and type.

Syntax: void Set_Validation(char * messagestring);

Remarks: The string pointed to by *messagestring* will be validated on subsequent documents as specified by the Scan_Cmd function. Set_Validation by itself will no cause any actions, but will simply specify the string that will be used later. The string must be of length 80 characters or less including the terminating carriage return.

Each call to Set_Validation, clears any previous or pending message.

Strings should terminate with a Carriage Return (0x0d) and 2 nulls.

Fonts may be selected by inserting the following special characters into the strings:

Symbol	Value	Font / Style
IJ_STD	0x1f	Font = Standard
IJ_STDBOLD	0x1e	Font = Standard Bold
IJ_LRG	0x1d	Font = Large
IJ_LRGBOLD	0x1c	Font = Large Bold
IJ_BC39	0x14	Font = Barcode 39
IJ_SINGLE	0x0e	Single Wide
IJ_DOUBLE	0x0f	Double Wide

Returns: Returns one of the following values.

MJ_COMMOK	communications successful
MJ_NOTINIT	not successful, not attached, or unknown error

Examples:

```
#include "mj4000_dl.h"
int main(void) {
char valmsg[80] = "Test Print - \x1c LARGE\x0d";

       Init_Scanner();     /* presume successful */

       Set_Validation( valmsg );

       /* scanner code here */

       return 0;
}
```

Scanner_Cmd

Function: Commands scanner action.

Syntax: int Scanner_Cmd(int cmdno, int cmdoptions, Scan_Results * scanresults);

Remarks: This function commands the scanner to perform its actions such as scanning, MICR reading, validation, etc. The command executed is specified by the bit-mapped variable *cmdno*, selected from the list below.

Value	Define	Execution
0x01	CMD_VALIDATE	Validate only
0x02	CMD_MICR	MICR read document
0x04	CMD_SCAN	Scan both sides of document
0x06	CMD_SCAN_MICR	Scan and MICR read document
0x07	CMD_SCAN_MICR_VAL	Scan, MICR, and validate document.

The variable *cmdoptions* is reserved for future use. Set to 0. Pass a pointer to a structure of type Scan_Results. This structure will be filled with values collected from the scan or MICR read.

Returns: Returns one of the following values.

MJ_COMMOK communications successful
MJ_NOTINIT not successful, not attached, or unknown error

Fills the structure pointed to by *scanresults*:

```
Struct Scan_Results{
        int cmdno;                  // cmdno used, this scan
        int cmdoptions;             // options used, this scan
        int scan_errorcode;         // scanning error code
        char * buf_sideA;           //
        char * buf_sideB;           //
        int Filefmt;                // see below
        int Scansides;              // 1= side A, 2 = sides A + B
        int color;                  // 1 = color,  0=B/W imaging
        int bitdepth;               // bits stored in memory (not bits scanned)
        int xdpi;                   // xdpi of image
        int ydpi;                   // ydpi of image
        int x_pixels;               // pixels in x direction
        int y_pixels;               // pixels in y direction
        int MICR_returncode;        // MICR return code
        char micrline_raw[80];      // MICR line received, raw format
        char validation_msg[80];    // validation message used
}
```

Examples:

```
#include “mj4000_dl.h”
int main(void) {
Scan_Restults scandata;

       Init_Scanner();     /* presume successful */

       Set_Validation( “Test Print - \x1c LARGE\n” );
       Set_Data_Formats(  );
       Scanner_Cmd(CMD_SCAN_MICR_VAL, 0, &scandata );

       printf( “MICR ERRCODE: %d\n”, scandata.MICR_errorcode] );
       printf( “MICR LINE: %s\n”, &scandata.micrline_raw[0] );

       return 0;
}
```

Parse_MICR

Function: Parse MICR line into its component parts.

Syntax: int Parse_MICR(char * MICRLine_Raw, int *returncode*, int fmtoptions, struct * MICR_Parsed,);

Remarks: This command parse the MICR line pointed to by *MICRLine_Raw* and place its components into the structure shown below. Pass also the parameter *returncode* to keep all elements concerning a MICR read in one structure.

```
Struct MICR_Parsed{
        int returncode;                 // MICR return code
        int doctype;                    // document type identified by parsing routine
        int fmtoptions;                 // options used in parsing
        char micrline_raw[80];          // raw data of MICR read.
        char routingnumber[32];         // routing / transit number
        char accountnumber[32];         // account number
        char consecutivenumber[32];     // check consecutive number
        char onusfield[32];             // not filled at this time.
        char aux_onusfield[32];         // auxiliary on-us field
        char amountfield[32];           // amount field
        char epcfield[32];              // epc field
}
```

The parameters **MICRLine_Raw* and *MICR_returncode* are available after scanning or reading and are found in the *Scan_Results* structure. See **Scanner_Cmd()** function for more details.

The variable *fmtoptions* is used to specify various formatting options. Set this variable to 0. *fmtoptions* will be implemented later to allow for differing representations for MICR symbols and other similar issues.

The field *onusfield* is not specified in the first version of the driver.

Returns: TRUE = valid MICR line was passed and parsed.
FALSE = no valid MICR line passed.

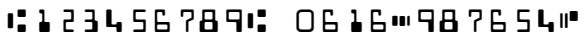
Returns values in structure *MICR_Parsed* specifying values of various fields.

MICR_returncode has one of the following values:

Return Code	*Meaning*
0x30	No errors. Good Read
0x31	Error: Hardware unavailable or environment bad.
0x32	Error: No data in Transmit Buffer
0x33	Error: Read or Decoding error.
0x34	Error: Insufficient magnetic ink detected.
0x35	Error: Time-out on Read operation.
0x36	Error: Document too long or feed jam.
0x37	Error: Read aborted by Host.
0x38 -3F	Error: Reserved for future use.

Examples:

Consider this example E13B document:

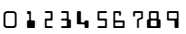


The following fields will be identified:

returncode = 0x30 (Presume no error).
doctype = 0x31 (Describes which internal rule used).
micrline_raw = "C123456789C 0616A987654D"
routingnumber = "123456789"
accountnumber = "0616A987654"
consecutivenumber = ""
onusfield = " 0616A987654"
aux_onusfield = ""
amountfield = ""
epcfield = ""

Notes on E13B MICR Symbology:

For E13B special symbols are included. This are mapped to ASCII characters using the following table.

Symbol Name	*MICR Symbol*	*Character Returned <Default>*	***Hex Value***
space	" "	" "	20H
0123456789		0123456789	30H to 39H
Dash		A	41H
Amount		B	42H
Transit		C	43H
On-Us		D	44H
Unknown		?	3FH

The question mark character "?" is returned if a character can not be deciphered with an adequate confidence level.

Write_Data_Files

Function: Writes out images to files as specified.

Syntax: int Write_Data_Files(char * pathname, char * fileprefix, char * filebody, char * filename_sideA, char * filename_sideB);

Remarks: Writes the images stored in memory to the data files specified. To omit storing one of the two files, use a null string for the filename. Filename extensions are either .raw, .bmp, or .tif as required.

Images are written with the tags specified by the Set_ImageData_Tags() function.

Returns: Returns one of the following values.
0x00 = Files both written successfully.
0x01 = File for sideA not written or write failure.
0x02 = File for sideB not written or write failure.
0x03 = Files for both sideA and sideB not written or write failure.

Examples:

Write the following sides to the following files.

Side	Path & File
A	C:\mj4000\images\W0000001A.tif
B	C:\mj4000\images\W0000001B.tif

Use these variable choices (or equivalents):

```
#include "mj4000_dl.h"
int main(void) {
char   path[32],prefix[32],body[32],sideA[32],sideB[32];

       Init_Scanner();     /* presume successful */
       /* scanner code here */

       strcpy( path, "c:\mj4000\images\" );
       strcpy( prefix, "W" );
       strcpy( body,  "0000001" );
       strcpy( sideA,  "A" );
       strcpy( sideB,  "B" );
       Write_Data_Files( path, prefix, body, sideA, sideB );

       return 0;
}
```

3 Using the DLL

3.1 Coding Overview

To be written.

3.2 Header File - MJ4000_dl.h

```
// The following ifdef block is the standard way of creating macros which make exporting
// from a DLL simpler. All files within this DLL are compiled with the MJ4000_DLL_EXPORTS
// symbol defined on the command line. this symbol should not be defined on any project
// that uses this DLL. This way any other project whose source files include this file see
// MJ4000_DLL_API functions as being imported from a DLL, wheras this DLL sees symbols
// defined with this macro as being exported.

#ifdef MJ4000_DLL_EXPORTS
#define MJ4000_DLL_API __declspec(dllexport)
#else
#define MJ4000_DLL_API __declspec(dllimport)
#endif

// This class is exported from the mj4000_dll.dll
class MJ4000_DLL_API CMj4000_dll {
public:
	CMj4000_dll(void);
	// TODO: add your methods here.
};

extern MJ4000_DLL_API int nMj4000_dll;

MJ4000_DLL_API int fnMj4000_dll(void);


struct MICR_Parsed
{
	int	returncode;
	int	doctype;
	char	micrline_raw[80];
	char	routingnumber[32];
	char	accountnumber[32];
	char	consecutivenumber[32];
	char	onusfield[32];			// to be added
	char	bankno[32];
	char	aux_onusfield[32];
	char	amountfield[32];
	char	epcfield;
};

struct Scan_Results
{
	int	cmdno;			// cmdno used, this scan
	int	cmdoptions;		// options used, this scan
	int	scan_returncode;	// scanning return code
	char*	buf_sideA;
	char*	buf_sideB;
	int	filefmt;		// see below	????????
	int	scansides;		// 1 = side A, 2 = sides A + B
	int	color;			// 1 = color, 0 = B/W imaging
	int	bitdepth;		// bits stored in memory (not bits scanned)
	int	xdpi;			// xdpi of image
	int	ydpi;			// ydpi of image
	int	x_pixels;		// pixels in x direction
	int	y_pixels;		// pixels in y direction
	int	micr_returncode;	// MICR return code
	char	micrline_raw[80];	// MICR line received, raw format
```

```
	char	validation_msg[80]; // validation msg used.
};

struct CapScanner
{
	int	scansides;		// number of sides scanned
	int	color;			// color support, 0 = no
	int	bitdepth;		// bits of grey scale, 1 = binary
	int	xdpi;			// x resolution, dots per inch
	int	ydpi;			// y resolution, dots per inch
	int	x_maxpixels;		// max pixels scanned in x
	int	y_maxpixels;		// max pixels scanned in y
	int	MICR;			// MICR support, 1 = yes, 0 = no
	int	Validation;		// Validation - number of lines
	int	Journal;		// Journal, 1 = yes, 0 = no
	int	Magcard;		// Magcard, 1 = yes, 0 = no
};

struct scan_fmt
{
	int	Filefmt;		// see below	????????
	int	scansides;		// 1 = side A, 2 = sides A + B
	int	color;			// 1 = color, 0 = B/W imaging
	int	bitdepth;		// bits stored in memory (not bits scanned)
	int	xdpi;			// reserved;
	int	ydpi;			// reserved;
	int	x_maxpixels;		// reserved;
	int	y_maxpixels;		// reserved;
};

/**********************    FUNCTIONS    **************************/

MJ4000_DLL_API int Init_Scanner(HANDLE hwnd);
MJ4000_DLL_API int Reset_Scanner(void);
MJ4000_DLL_API int Get_Scanner_Info( int infotype, char * results );
MJ4000_DLL_API int Get_Scanner_Capabilities( struct CapScanner * p_sc );
MJ4000_DLL_API int Get_Scanner_Statistics( int statno, long * value );
MJ4000_DLL_API int Get_Scanner_Status( int statno, int * mechstatus );
MJ4000_DLL_API int Set_Scanner_Parameters( struct ScanParameters * );
MJ4000_DLL_API int Set_Image_Data_Format( struct * scan_fmt );
//MJ4000_DLL_API int Set_Image_Data_Tag( tag_no, tag_type, tag_val );
MJ4000_DLL_API int Set_Validation( char * messagestring );
MJ4000_DLL_API int Scanner_Cmd( int cmdno, int cmdoptions, Scan_Results *
scanresults );
MJ4000_DLL_API int Parse_MICR( char * MICRLine_Raw, int fmtoptions,
	struct MICR_Parsed * micr_s);
MJ4000_DLL_API int Write_Data_Files( char * pathname, char * fileprefix, char * filebody,
	 char * filename_sideA, char * filename_sideB  );


/************************    CONSTANTS and DEFINES    *******************/
#define	MJ_INIT			1
#define	MJ_NOT_INIT		0

#define	MJ_RESET		1
#define	MJ_NOT_RESET		0

#define	MJ_COMMOK		1

#define	MJ_BOTH_FILES_WRITTEN		0
#define	MJ_SIDE_A_FILE_NOT_WRITTEN	0x01
#define	MJ_SIDE_B_FILE_NOT_WRITTEN	0x02
#define	MJ_NO_FILES_WRITTEN		0x03
```

```
/************************     FONTS    *************************/

#define IJ_SRD              0x1f
#define IJ_STDBOLD          0x1e
#define IJ_LRG              0x1d
#define IJ_LRGBOLD          0x1c
#define IJ_BC39             0x14
#define IJ_SINGLE           0x0e
#define IJ_DOUBLE           0x0f

/************************     INFO TYPES     ***********************/

#define      INFO_MOD            0            //  ??????????????
#define      INFO_MODNO          1            //  ??????????????
#define      INFO_VER            2            //  ??????????????
#define      INFO_FTVER          3            //  ??????????????
#define      INFO_MFG            7            //  ??????????????
#define      INFO_TEMP0          9            //  ??????????????
#define      INFO_USER1          12           //  ??????????????
#define      INFO_USER2          13           //  ??????????????
#define      INFO_FACT           14           //  ??????????????

/************************     STATISTICS     ***********************/

#define      STAT_POR            0            //  ??????????????
#define      STAT_DOTS           9            //  ??????????????
#define      STAT_FORMS          11           //  ??????????????
#define      STAT_MICRDOCS 16           //  ??????????????
#define      STAT_SCANDOCS 24           //  ??????????????

/************************     SCANNER COMMANDS      *********************/

#define      CMD_RESET           0
#define      CMD_VALIDATE        1
#define      CMD_MICR            2
#define      CMD_SCAN            4
#define      CMD_SCAN_MICR       6
#define      CMD_SCAN_MICR_VAL   7
```

3.3 Loading & Linking

To be written.

4 Document Revision Information

Revision	***Date***	***Changes***
1.00	05-23--2005	Initial release.