EXCELLA
&
EXCELLA STX

WINDOWS API SPECIFICATIONS
PROGRAMMING REFERENCE MANUAL

MANUAL PART NUMBER 99875340-10

MAY 2014

MAGTEK®
REGISTERED TO ISO 9001:2008
1710 Apollo Court
Seal Beach, CA 90740
Phone: (562) 546-6400
FAX: (562) 546-6301
Technical Support: (651) 415-6800
www.magtek.com

REVISIONS

Rev Number	Date	Notes
1	30 Mar 06	Initial Release
2	15 Feb 07	Added new Sections 4, 5, 6, and 7. Move old Section 4 to new Section 8. Miscellaneous editorial changes.
3	13 Jun 07	Added new Appendix C for Mitek ImageScore.
4	16 Nov 09	Added JPEGQG and JPEGQC
5	22 Dec 09	Added StartTimeout, ExpressCapable, ExpressEnabled, USBSpeed
6	7 Jun 2010	Added MICROptions section
7	5 Apr 2011	Added MTMICRSetConfigFile function and error codes 67, 68, 69, and 70.
8	23 Oct 2011	Added new option for <ProcessOptions><DocFeed>
9	1 Mar 2012	Updated MagneSafe MSR info for clarification
10	6 May 2014	Updated requirements for Virtual Endorsing; minor formatting corrections

SOFTWARE LICENSE AGREEMENT

IMPORTANT: YOU SHOULD CAREFULLY READ ALL THE TERMS, CONDITIONS AND RESTRICTIONS OF THIS LICENSE AGREEMENT BEFORE INSTALLING THE SOFTWARE PACKAGE. YOUR INSTALLATION OF THE SOFTWARE PACKAGE PRESUMES YOUR ACCEPTANCE OF THE TERMS, CONDITIONS, AND RESTRICTIONS CONTAINED IN THIS AGREEMENT. IF YOU DO NOT AGREE WITH THESE TERMS, CONDITIONS, AND RESTRICTIONS, PROMPTLY RETURN THE SOFTWARE PACKAGE AND ASSOCIATED DOCUMENTATION TO THE ABOVE ADDRESS, ATTENTION: CUSTOMER SUPPORT.

TERMS, CONDITIONS, AND RESTRICTIONS

MagTek, Incorporated (the "Licensor") owns and has the right to distribute the described software and documentation, collectively referred to as the "Software".

LICENSE: Licensor grants you (the "Licensee") the right to use the Software in conjunction with MagTek products. LICENSEE MAY NOT COPY, MODIFY, OR TRANSFER THE SOFTWARE IN WHOLE OR IN PART EXCEPT AS EXPRESSLY PROVIDED IN THIS AGREEMENT. Licensee may not decompile, disassemble, or in any other manner attempt to reverse engineer the Software. Licensee shall not tamper with, bypass, or alter any security features of the software or attempt to do so.

TRANSFER: Licensee may not transfer the Software or license to the Software to another party without the prior written authorization of the Licensor. If Licensee transfers the Software without authorization, all rights granted under this Agreement are automatically terminated.

COPYRIGHT: The Software is copyrighted. Licensee may not copy the Software except for archival purposes or to load for execution purposes. All other copies of the Software are in violation of this Agreement.

TERM: This Agreement is in effect as long as Licensee continues the use of the Software. The Licensor also reserves the right to terminate this Agreement if Licensee fails to comply with any of the terms, conditions, or restrictions contained herein. Should Licensor terminate this Agreement due to Licensee's failure to comply, Licensee agrees to return the Software to Licensor. Receipt of returned Software by the Licensor shall mark the termination.

LIMITED WARRANTY: Licensor warrants to the Licensee that the disk(s) or other media on which the Software is recorded are free from defects in material or workmanship under normal use.

THE SOFTWARE IS PROVIDED AS IS. LICENSOR MAKES NO OTHER WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Because of the diversity of conditions and PC hardware under which the Software may be used, Licensor does not warrant that the Software will meet Licensee specifications or that the operation of the Software will be uninterrupted or free of errors.

IN NO EVENT WILL LICENSOR BE LIABLE FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE, OR INABILITY TO USE, THE SOFTWARE. Licensee's sole remedy in the event of a defect in material or workmanship is expressly limited to replacement of the Software disk(s) if applicable.

GOVERNING LAW: If any provision of this Agreement is found to be unlawful, void, or unenforceable, that provision shall be removed from consideration under this Agreement and will not affect the enforceability of any of the remaining provisions. This Agreement shall be governed by the laws of the State of California and shall inure to the benefit of MagTek, Incorporated, its successors or assigns.

ACKNOWLEDGMENT: LICENSEE ACKNOWLEDGES THAT HE HAS READ THIS AGREEMENT, UNDERSTANDS ALL OF ITS TERMS, CONDITIONS, AND RESTRICTIONS, AND AGREES TO BE BOUND BY THEM. LICENSEE ALSO AGREES THAT THIS AGREEMENT SUPERSEDES ANY AND ALL VERBAL AND WRITTEN COMMUNICATIONS BETWEEN LICENSOR AND LICENSEE OR THEIR ASSIGNS RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

QUESTIONS REGARDING THIS AGREEMENT SHOULD BE ADDRESSED IN WRITING TO MAGTEK, INCORPORATED, ATTENTION: CUSTOMER SUPPORT, AT THE ABOVE ADDRESS, OR E-MAILED TO support@magtek.com.

TABLE OF CONTENTS

TABLES AND FIGURES

SECTION 1. OVERVIEW

The sections of this manual are as follows:

Section 1. Overview
Section 2. Excella Software Architecture – includes flow diagrams, screen captures, and several "How To" descriptions.
Section 3. Excella API – describes Excella device API functions and return codes.
Section 4. Commands Sent To Device – describes commands sent by the application to the Excella device.
Section 5. Keys Sent to Device – lists and explains keys sent to the Excella device by the application.
Section 6. Keys Received From Device – lists and explains keys received from the Excella device.
Section 7. Other Keys Available from Device – lists additional keys available from Excella device.
Section 8. Examples of Key-Value Pairs
Appendix A. Format List – built-in list of MICR data formats from which the user may select one to become the active Format every time a check is read.
Appendix B. Error codes and messages from Excella API MTXMLMCR.dll.

REQUIREMENTS

The following item is required for software installation:

P/N 22359069, API/Demo for Excella STX (CD)

For the USB interface, this CD installs USB Drivers, MTXMLMCR.dll and Demo program.

SECTION 2. EXCELLA SOFTWARE ARCHITECTURE

(For RNDIS USB Drivers Only)

The architecture of the system is shown in Figure 2-1. Descriptions of the terms and operations used follow the illustration.

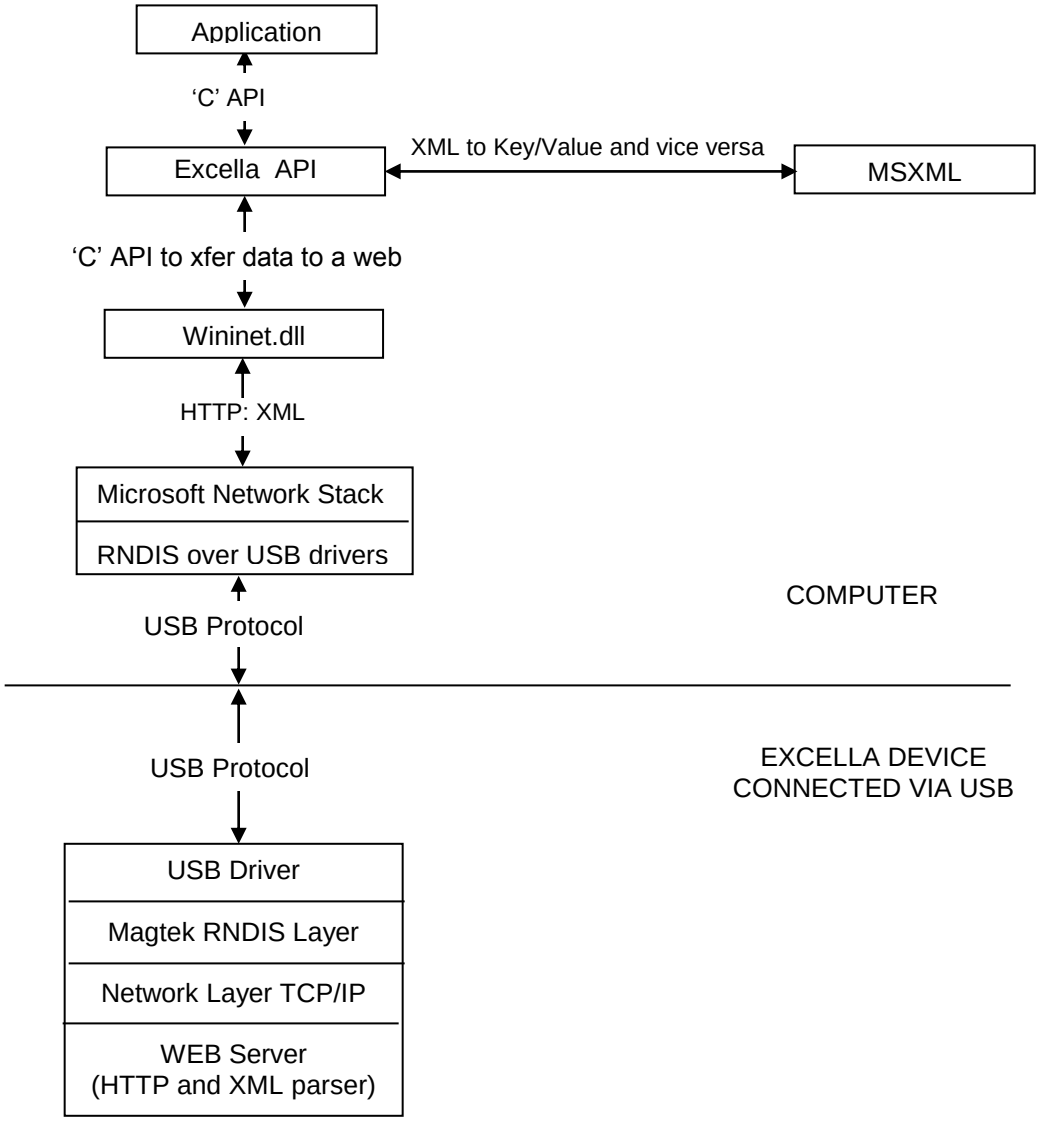


Figure 2-1. Excella Software Architecture

TERM DESCRIPTION

MTXMLMCR.dll.
Application provides API to upper level application to talk to the device. The application does not have any knowledge of how the device is connected to the computer. Thus it does not expose the transport protocol to the application. We are using HTTP protocol with XML to communicate with the device. The **MTXMLMCR.dll** uses **wininet.dll** to talk to internet protocols. The **MTXMLMCR.dll.** provides functions to convert the scan request into XML format using **msxml4.dll** and then send it to the device using **wininet.dll**. It also provides functions to convert the response from the device (XML format) to key/value pairs.

msxml4.dll : Microsoft XML Parser**. MTXMLMCR.dll.**
uses **msxml4.dll** to convert the key/value pair to XML language. **MTXMLMCR.dll** also uses the **msxml4.dll** to convert the XML data back to key/ value pair.

wininet.dll: MFC Win32 Internet Extension. **wininet.dll**, provide access to common Internet protocols, including Gopher, FTP, and HTTP. **MTXMLMCR.dll** uses **wininet.dll** to establish an Internet connection with Excella device. It then communicates with Excella using GET and POST requests provide by **wininet.dll**.

RNDIS: Microsoft Ethernet to USB driver that makes USB device look like an Ethernet device. This driver supports Windows 98, ME, 2000 and XP.

DEVICES ATTACH THROUGH USB NETWORK CARD

Devices attach through the USB Network card are shown in Figure 2-2.*

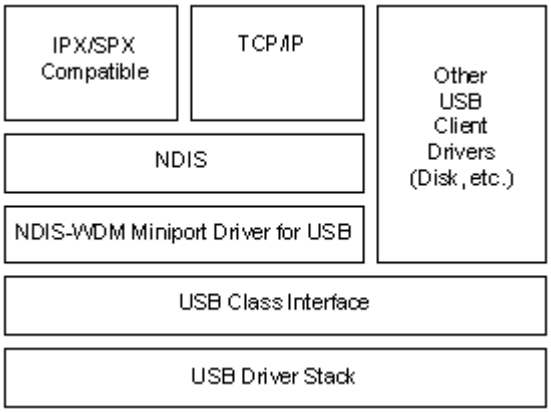


Figure 2-2. Devices attach through USB Network Card

* Figures 2-2 and 2-3 are produced by Microsoft.

RNDIS SUPPORT FOR A USB NETWORK DEVICE

RNDIS Support for a USB Network device is shown in Figure 2-3.*

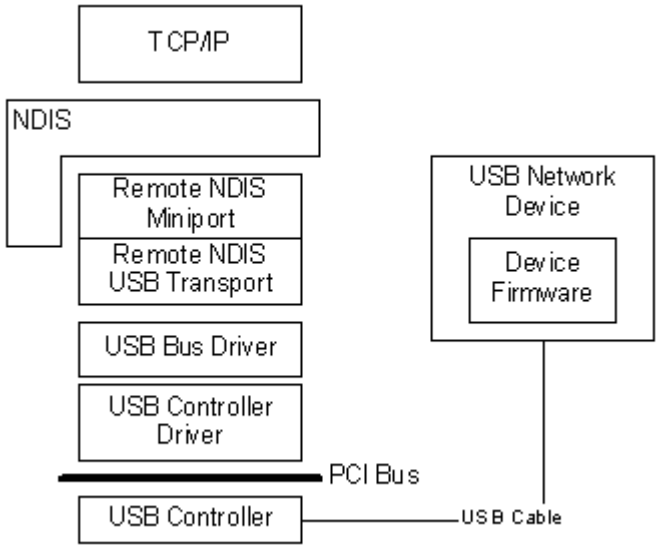


Figure 2-3. RNDIS Support for a USB Network Device

SUPPORT FOR EXCELLA DEVICE ON PC SIDE:

To support Excella device on the PC side, a template INF file provided by Microsoft was modified to install the RNDIS drivers. There are two driver files provided by Microsoft:

rndismp.sys (export driver and is linked to usb8023.sys)
usb8023.sys (For an RNDIS USB Device)

When usb8023.sys is loaded the system automatically loads **rndismp.sys**.

Windows XP has built in support for RNDIS. Windows 2000 does not have RNDIS drivers. MagTek supplies RNDIS driver for Windows 2000.

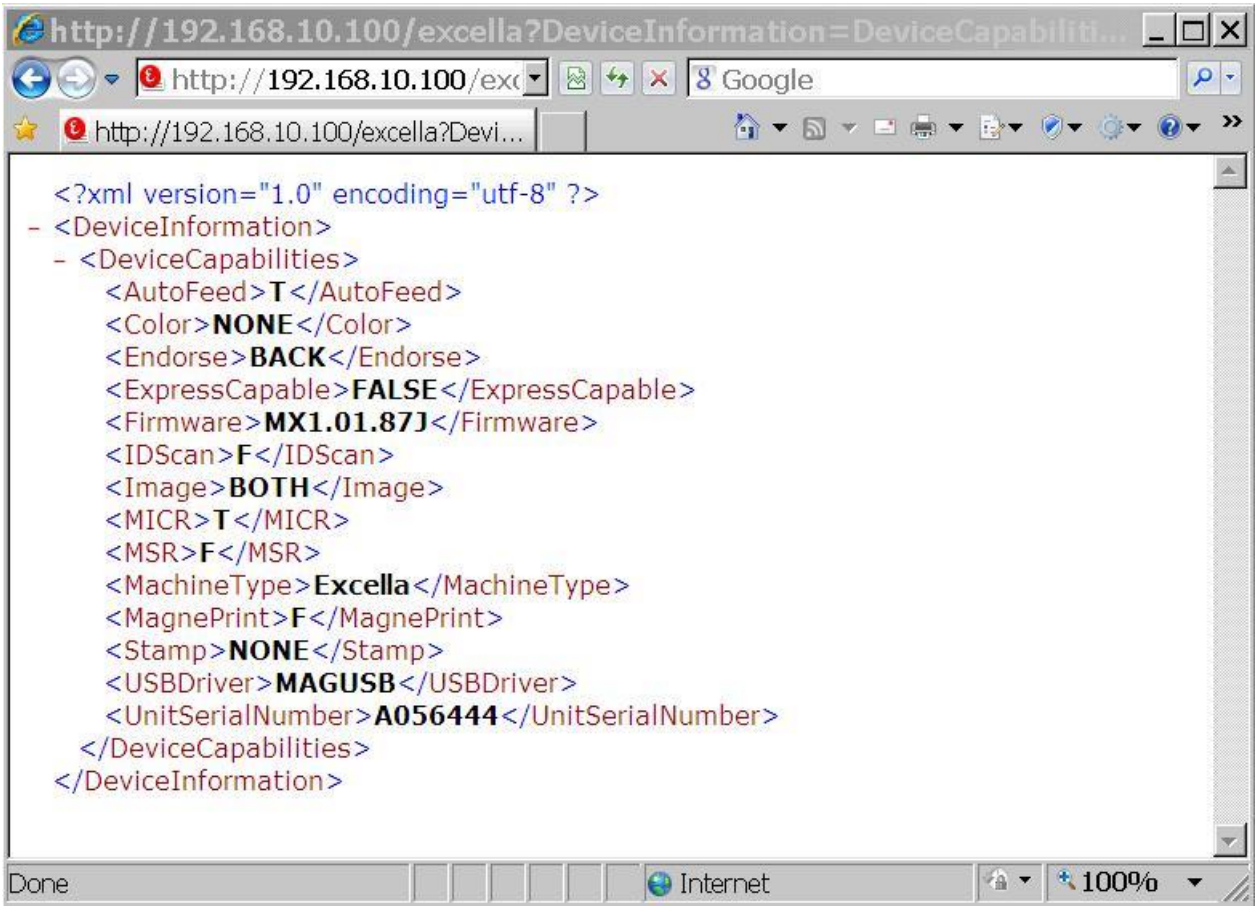
HOW TO COMMUNICATE WITH EXCELLA USING WEB BROWSER

Excella device can be accessed from an Internet Web browser using the IP address of the device. An example in using the IP address of the Excella device to obtain device capabilities information is followed.

Assuming the Excella device has IP address 192.168.10.100, type the following line in the address box of the Web browser. Internet Explorer is used in this example:

192.168.10.100\Excella?DeviceInformation=DeviceCapabilities

Press the Enter key, and the Excella device responds with the results shown on the next page:

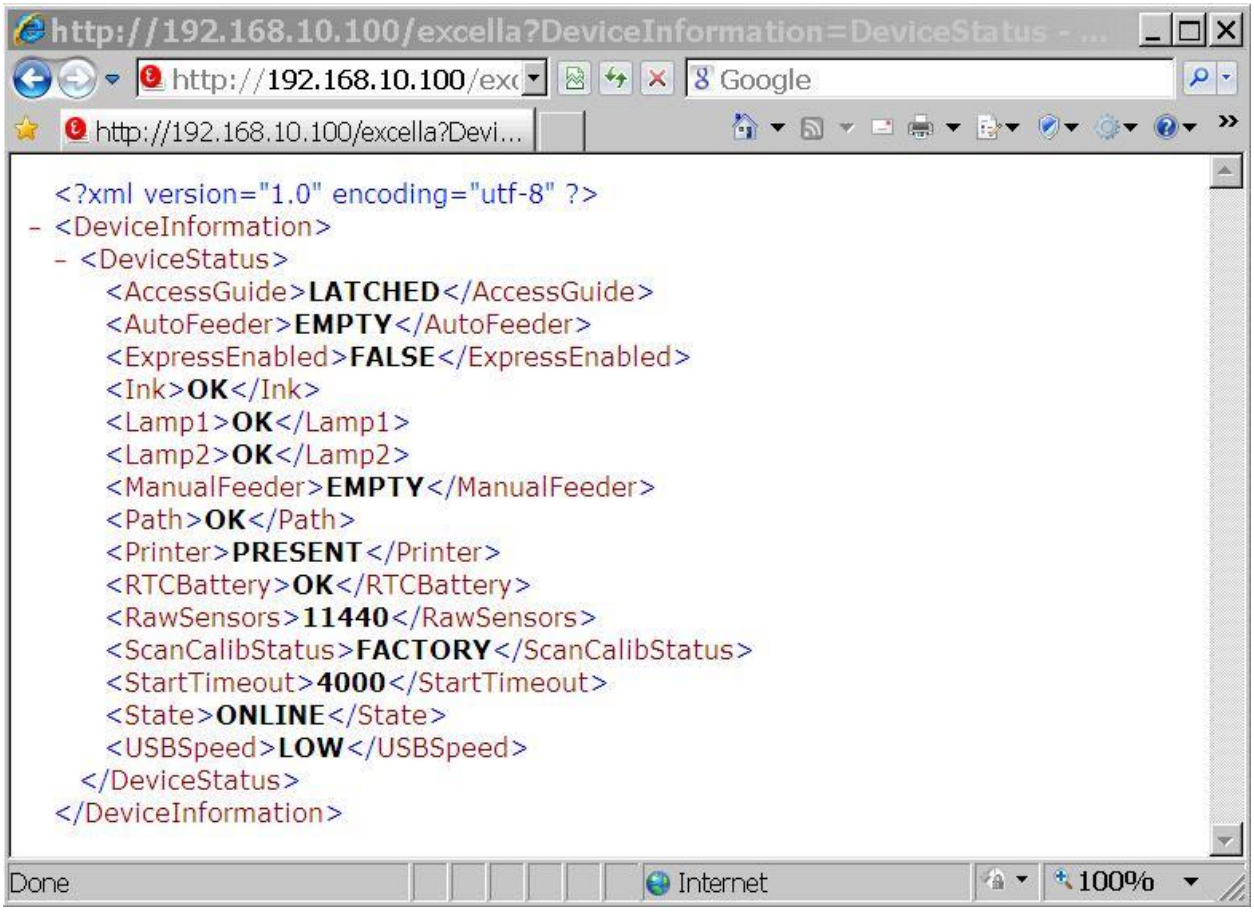

http://192.168.10.100/excella?DeviceInformation=DeviceCapabiliti...
http://192.168.10.100/excella?Devi...
Google
<?xml version="1.0" encoding="utf-8" ?>
- <DeviceInformation>
- <DeviceCapabilities>
<AutoFeed>T</AutoFeed>
<Color>NONE</Color>
<Endorse>BACK</Endorse>
<ExpressCapable>FALSE</ExpressCapable>
<Firmware>MX1.01.87J</Firmware>
<IDScan>F</IDScan>
<Image>BOTH</Image>
<MICR>T</MICR>
<MSR>F</MSR>
<MachineType>Excella</MachineType>
<MagnePrint>F</MagnePrint>
<Stamp>NONE</Stamp>
<USBDriver>MAGUSB</USBDriver>
<UnitSerialNumber>A056444</UnitSerialNumber>
</DeviceCapabilities>
</DeviceInformation>
Done
Internet
100%

Get Device Status

The following is an example in using the IP address of the Excella device to obtain device status information. Assuming the Excella device has IP address 192.168.10.100, type the following line in the address box of the Web browser. Internet Explorer is used in this example:

192.168.10.100\Excella?DeviceInformation= DeviceStatus

Press the Enter key, and the Excella device responds with the results shown below:

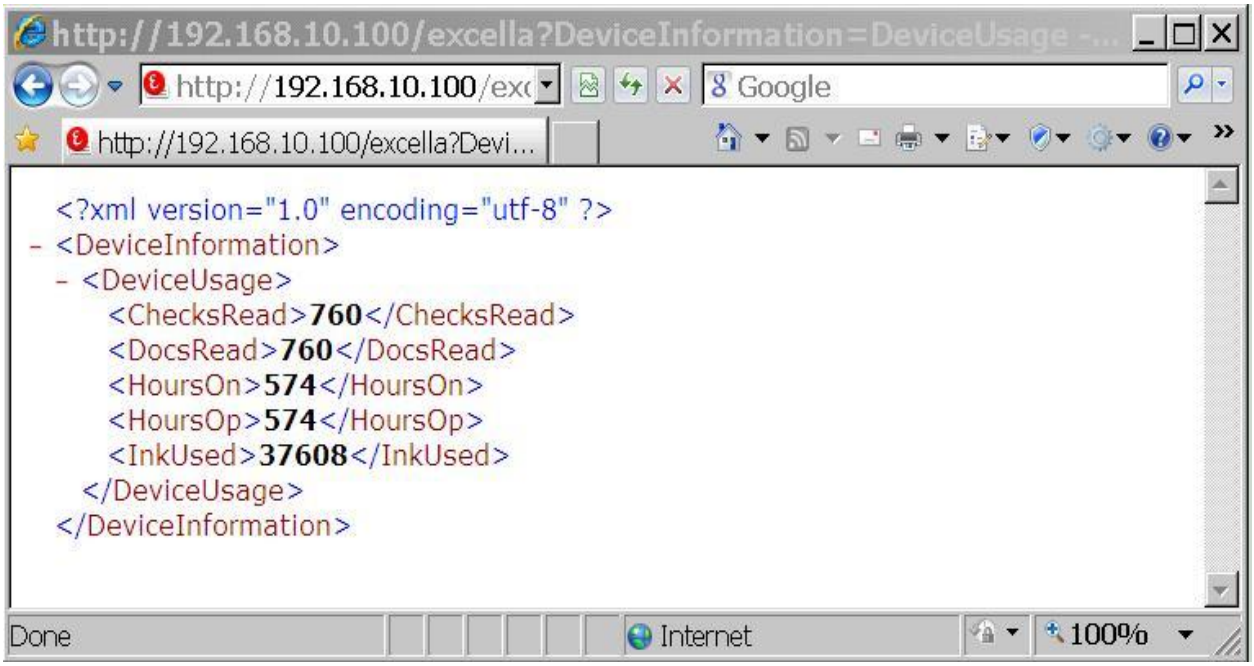


Get Device Usage

The following is an example in using the IP address of the Excella device to obtain device usage information. Assuming the Excella device has IP address 192.168.10.100, type the following line in the address box of the Web browser. Internet Explorer is used in this example:

192.168.10.100\Excella?DeviceInformation= DeviceUsage

Press the Enter key, and the Excella device responds with the results shown below:



HOW TO COMMUNICATE WITH EXCELLA USING THE EXCELLA API

The following table lists required files in order to use Excella API:

Files	Location	Description
micrdev.ini	Windows folder	This file contains a list of default Excella connections and IP addresses.
MTXMLMCR.dll	Windows System32 folder	Excella API
wininet.dll	Windows System32 folder.	Win32 API for Internet Protocols. This file is provided by Microsoft
msxml4.dll	Windows System32 folder.	MSXML XML parser. ActiveX Object for XML API. This file is installed with the installation of Excella API

API FUNCTIONS

Table 2-1 lists functions provided by **MTXMLMCR.dll**. Please refer to Section 3 for a complete description of these functions.

Table 2-1. Functions

NAME	DESCRIPTION	PAGE
MTMICRGetDevice	Get device name of all devices present.	**15**
MTMICROpenDevice	Opens device with the given name.	**16**
MTMICRCloseDevice	Closes device with the given name.	**17**
MTMICRDeviceConnect	Connects device with IP or DNS name.	**18**
MTMICRDeviceDisconnect	Disconnect device with the given IP or DNS name.	**19**
MTMICRSetValue	Adds a key/value pair to a given section.	**20**
MTMICRSetIndexValue	Adds a key/value pair with index to a given section.	**21**
MTMICRGetValue	Returns value of a key/value pair from a given section.	**22**
MTMICRGetIndexValue	Returns value of a key/value pair with index from a given section.	**24**
MTMICRQueryInfo	Sends a request to a given device name to query for info on a given query parameter.	**25**
MTMICRSendCommand	Sends a single command to a given device name.	**27**
MTMICRProcessCheck	Sends a scan request to a given device to scan with a given process options.	**28**
MTMICRGetImage	Sends request to the given device name to get an image of a previously scanned check.	**30**
MTMICRGetImages	Sends request to the given device name to get one or more images of a previously scanned check.	**32**
MTMICRGetSectionCount	Returns the number of sections present in a given document information.	**34**
MTMICRGetSectionName	Returns the section name of the given section number in a given document information.	**36**
MTMICRGetKeyCount	Returns the number of keys present in a given section in a given document information.	**37**
MTMICRGetKeyName	Returns the key name of the given key number in a given document information.	**40**
MTMICRSetTimeout	Sets device communication timeout in milliseconds.	**48**
MTMICRGetTimeout	Gets device communication timeout in milliseconds.	**49**
MTMICRLogEnable	Enable the logging.	**50**
MTMICRSetLogFileHandle	Logs errors to the given file handle.	**51**
MTMICRSetLogLevel	Sets the level of details for logging	**52**
MTMICRCOMInitialize	Enable MSXML instantiation at DLL level	**53**
MTMICRCOMUnInitialize	Disable MSXML instantiation at DLL level	**54**
MTMICRSetConfigFile	Set the path and file name of the config file. Default is "MICRDEV.INI"	**55**

SOFTWARE FLOW FOR CHECK PROCESSING

Figure 2-4 illustrates the sequence of check processing.

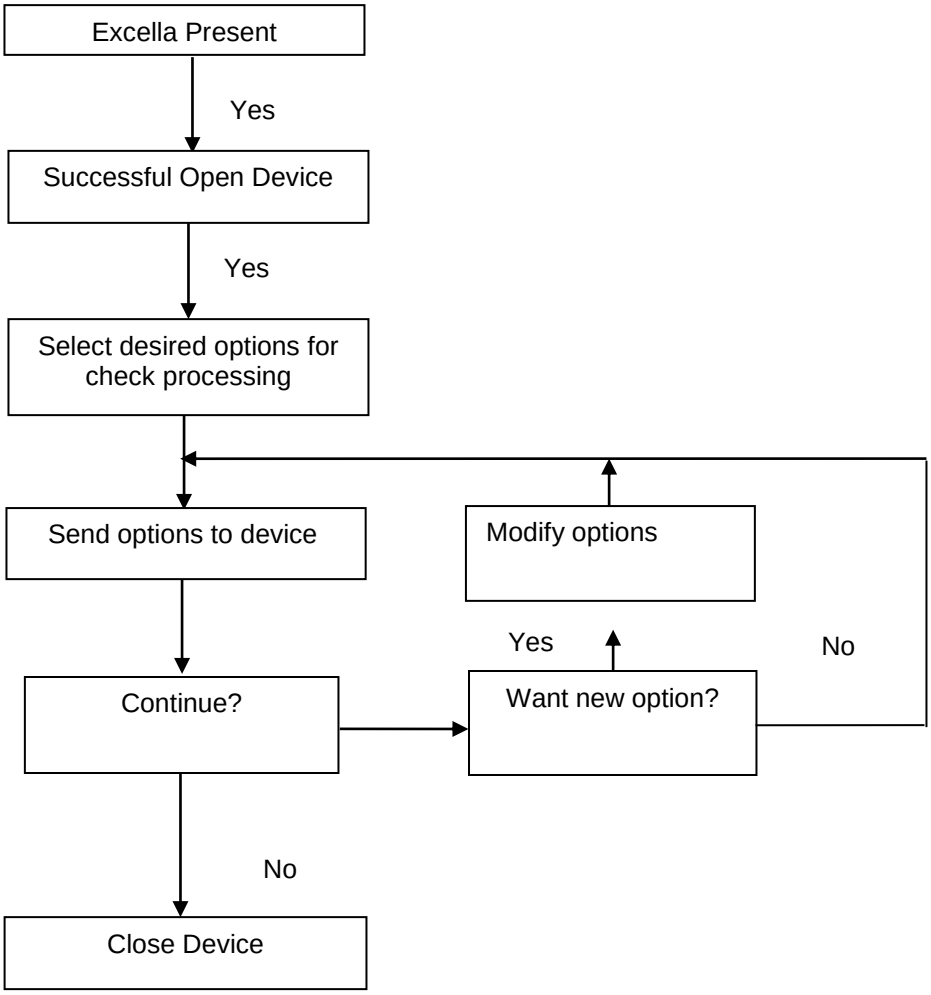


Figure 2-4. Software Flow for Check Processing

HOW TO PROCESS DOCUMENT USING EXCELLA API

To process document, follow these steps:

1. Find the attached Excella device by calling function **MTMICRGetDevice**.

2. Use function **MTMICROpenDevice** to open the device.

3. Excella is now ready to accept the XML options using function **MTMICRProcessCheck**. For every document that is processed, a set of options must be sent to the device. These options define how the document will be processed, i.e., image color, image type, image resolution, print back, print front etc. The options are stored as key/value pairs in a buffer. The memory for this buffer must be allocated big enough to store all selected options. These options must be sent to the Excella device using function **MTMICRProcessCheck**. Use function **MTMICRSetValue** and function **MTMICRSetIndexValue** to store the key/value pairs in the option buffer. The options can be set in the buffer only once.

4. Use function **MTMICRProcessCheck** to send the process options to the Excella device.

5. Excella processes the document with the given process options and returns the result of the process operation. The result is stored in the third parameter of the function **MTMICRProcessCheck**. The result contains only the information of the operation and information of the device. It does not contain scanned data. Information such as the size of the scanned images and the image identifier are included. The image information can be found under ImageInfo section. Use function **MTMICRGetValue** and function **MTMICRGetIndexValue** to retrieve key/value pair for image information. If there is no error returned in the document information, use function **MTMICRGetImage** or function **MTMICRGetImages** to get image data.

6. Go to step 4 to process the next check.

HOW TO GET CHECK IMAGES

To get a check image, follow these steps:

1. The function **MTMICRProcessCheck** contains information of the process operation provided by Excella device after it completes document processing. Use this information to find the information of the scanned images in the ImageInfo section. The ImageInfo section contains the size of the image and the image identification for each image.

2. Use function **MTMICRGetValue** or function **MTMICRGetIndexValue** to retrieve the size and the URL of each image. Allocate memory for each image according to its given size.

3. Use function **MTMICRGetImage** to get one image at a time or function **MTMICRGetImages** to get all images at once. The image data is transferred to the buffer that has memory allocated.

PROCESS OPTIONS

Process options are stored as key/value pairs in a buffer. The structure of the options follows XML format. Use function MTMICRSetValue or function MTMICRSetIndexValue to add an option to this structure. Options are string-based key/value pairs. Each option is stored only once in the buffer. Function MTMICRGetValue and function MTMICRGetIndexValue can be used to retrieve a key/value pair from this buffer.

ERROR REPORTING

When a document is processed, if there is an error in the Excella device or in the processing operation, the error is returned in the function MTMICRProcessCheck . Use function MTMICRGetValue or function MTMICRGetIndexValue to retrieve the error information. Table 4-13 to table 4-20 list return codes and return messages from Excella.

If an error occurs due to the failure of API function, the error is returned as the return code of the API provided by MTXMLMCR.dll. Appendix B contains a list of errors returning from MTXMLMCR.dll

DEBUGGING API

On the Excella Demo GUI program, there is an option to enable the error logging function. If this option is enabled, all errors returned from the device are logged into log file named "STXDemo.log". This file resides in the Excella Demo program installation directory.

SECTION 3. EXCELLA API

MTMICRGetDevice

MTMICRGetDevice function returns the device name of the device present in the system.

ULONG MTMICRGetDevice (
DWORD dwDeviceContext,
char *pcDevName
);

Parameters

dwDeviceContext
This is the device number of the device. This must be set to 1 to get the first device in the system. Increment it by 1 to get the next device name.

pcDevName
Pointer to the buffer where the device name will be stored.

Return Values

MICR_ST_OK
MICR_ST_DEVICE_NOT_FOUND
MICR_ST_BAD_PARAMETER

Remarks

If the function succeeds, the return value is MICR_ST_OK. The device name of the device is filled in the buffer pointed by the parameter pcDevName.

If there is no device present or there is no device that is associated with the given dwDeviceContext, then the function fails and MICR_ST_DEVICE_NOT_FOUND is returned.

If there is no memory allocated for the pcDevName parameter, then MICR_ST_BAD_PARAMETER is returned.

Example

```
#define DEVICE_NAME_LEN 128
int i=1;
DWORD dwResult;
char pcDevName[DEVICE_NAME_LEN]="";
while ((dwResult = MTMICRGetDevice(i,(char*) pcDevName)) != MICR_ST_DEVICE_NOT_FOUND)
{
      // Device found, increment the device number
      i++;
}
```

MTMICROpenDevice

MTMICROpenDevice function opens the device with the given device name.

ULONG MTMICROpenDevice (
char *pcDevName
);

Parameters

pcDevName
Pointer to null terminated string that specifies the name of the device to open. Use function **MTMICRGetDevice** to obtain the device name.

Return Values

If the function succeeds, the return value is MICR_ST_OK.

MICR_ST_OK
MICR_ST_BAD_DEVICE_NAME
MICR_ST_DEVICE_NOT_FOUND
MICR_ST_DEVICE_NOT_RESPONDING
MICR_ST_MSXML_NOT_FOUND
MICR_ST_MSXML_FAILED

Remarks

If the pcDevName is NULL, the return value is MICR_ST_BAD_DEVICE_NAME.
If no device is found, the return value is MICR_ST_DEVICE_NOT_FOUND.
If MSXML is not installed, return value is MICR_ST_MSXML_NOT_FOUND
If MSXML cannot be instantiated, return value is MICR_ST_MSXML_FAILED
If device is found but cannot connect, return value is MICR_ST_DEVICE_NOT_RESPONDING

Example

```
#define DEVICE_NAME_LEN 128
int i=1;
DWORD dwResult;
char pcDevName[DEVICE_NAME_LEN]="";
while ((dwResult = MTMICRGetDevice(i,(char*) pcDevName)) != MICR_ST_DEVICE_NOT_FOUND)
{
      if (MTMICROpenDevice (pcDevName) == MICR_ST_OK)
      {
            ///close the device
            MTMICRCloseDevice (pcDevName);
      }
      i++;
}
```

MTMICRCloseDevice

MTMICRCloseDevice function closes the device with the given device name.

ULONG MTMICRCloseDevice (
char *pcDevName
);

Parameters

pcDevName
Pointer to null terminated string containing the name of the device to close. This is the device that is previously opened using function **MTMICROpenDevice**.

Return Values

MICR_ST_OK
MICR_ST_BAD_DEVICE_NAME
MICR_ST_DEVICE_NOT_FOUND

Remarks

If the pcDevName is NULL, the return value is MICR_ST_BAD_DEVICE_NAME.

Example

```
#define DEVICE_NAME_LEN 128
int i=1;
DWORD dwResult;
char pcDevName[DEVICE_NAME_LEN]="";

// Get the device name at device number "i"
while ((dwResult = MTMICRGetDevice(i,(char*) pcDevName)) != MICR_ST_DEVICE_NOT_FOUND)
{
        // Success in getting the device name

       // Open this device
      if (MTMICROpenDevice (pcDevName) == MICR_ST_OK)
     {
           // Now close this device
           MTMICRCloseDevice (pcDevName);
      }
      else
     {
      // Error while opening this device.
     }

     i++;
}
```

MTMICRDeviceConnect

MTMICRDeviceConnect function connects host to a device that has the given IP address or a DNS name.

ULONG MTMICRDeviceConnect (LPSTR lpszDevice,
INTERNET_PORT nServerPort,
DWORD dwAccessType,
LPSTR lpszProxyName,
LPSTR lpszProxyBypass);

Parameters

lpszDevice: Pointer to a null terminated string that contains the IP address or DNS name of the device.
nServerPort: Unsigned integer that specifies the TCP/IP port on the server to which a connection is made.
dwAccessType: Type of Internet access. This parameter can be one of the following values:

INTERNET_OPEN_TYPE_DIRECT

INTERNET_OPEN_TYPE_PRECONFIG

INTERNET_OPEN_TYPE_PROXY

INTERNET_OPEN_TYPE_PRECONFIG_WITH_NO_AUTOPROXY

lpszProxyName: Pointer to a null-terminated string that specifies the name of the proxy server to use when proxy access is specified by setting *dwAccessType* to INTERNET_OPEN_TYPE_PROXY.
lpszProxyBypass: Pointer to a null-terminated string that specifies an optional list of host names or IP addresses, or both, that should not be routed through the proxy when *dwAccessType* is set to INTERNET_OPEN_TYPE_PROXY

Return Values

If the function succeeds, the return value is MICR_ST_OK.

MICR_ST_OK
MICR_ST_BAD_DEVICE_IP_OR_DOMAIN_NAME
MICR_ST_DEVICE_NOT_RESPONDING
MICR_ST_MSXML_NOT_FOUND
MICR_ST_MSXML_FAILED

Remarks

If parameter **lpszDevice** is NULL, the return value is MICR_ST_BAD_DEVICE_IP_OR_DOMAIN_NAME.
If no device is found, the return value is MICR_ST_DEVICE_NOT_FOUND.
If MSXML is not installed, return value is MICR_ST_MSXML_NOT_FOUND
If MSXML cannot be instantiated, return value is MICR_ST_MSXML_FAILED
If device is found but cannot connect, return value is MICR_ST_DEVICE_NOT_RESPONDING

Example

```
char logString[256];
DWORD dwStatus;

      dwStatus = MTMICRDeviceConnect("192.168.10.100",
                                    INTERNET_DEFAULT_HTTP_PORT,
                        INTERNET_OPEN_TYPE_DIRECT,
            "ProxyServer",
                                    "ProxyByPassIP");
      if(dwStatus!=MICR_ST_OK)
      {
            sprintf (logString, "Opening Device FAILED - ERROR %d", dwStatus);
            MessageBox (logString, "Demo", MB_OK);

      }
```

MTMICRDeviceDisconnect

MTMICRDeviceDisconnect function disconnects the device that has the given IP address or DNS name.

ULONG MTMICRDeviceDisconnect (LPSTR lpszDevice);

Parameters

lpszDevice Pointer to a null terminated string that contains the IP address or DNS name of the device. This is the device that is previously connected using function **MTMICRDeviceConnect**.

Return Values

MICR_ST_OK
MICR_ST_BAD_DEVICE_IP_OR_DOMAIN_NAME
MICR_ST_DEVICE_NOT_FOUND
MICR_ST_BAD_DEVICE_IP_OR_DOMAIN_NAME

Remarks

If the parameter **lpszDevice** is NULL, the return value is MICR_ST_BAD_DEVICE_IP_OR_DOMAIN_NAME.

Example

```
char logString[256];
DWORD dwStatus;

      dwStatus = MTMICRDeviceDisconnect("192.168.10.100");
      if(dwStatus!=MICR_ST_OK)
      {
            sprintf( logString, "CANNOT CLOSE DEVICE - ERROR %d", dwStatus);
            MessageBox (logString, "Demo", MB_OK);

      }
```

MTMICRSetValue

MTMICRSetValue function adds a key/value pair to the given device settings specified in pcOptions buffer and in a given section specified in pcSection buffer.

ULONG MTMICRSetValue (

```
        char        *pcOptions,
        char        *pcSection,
        char        *pcKey,
        char        *pcValue,
        DWORD       *pdwLength
);
```

Parameters

pcOptions
Pointer to null terminated string containing all key/value pairs.
pcSection
Pointer to null terminated string containing the section name.
pcKey
Pointer to null terminated string containing the key name.
pcValue
Pointer to null terminated string containing the key value. If this is NULL, then the key pcKey is deleted from the section pcSection.
pdwLength:
Pointer to a double word that contains the size of the pcOptions buffer.

Return Values

MICR_ST_OK
MICR_ST_NOT_ENOUGH_MEMORY
MICR_ST_BAD_DATA
MICR_ST_BAD_SECTION_NAME
MICR_ST_BAD_BUFFER_LENGTH

Remarks

The functions return MICR_ST_NOT_ENOUGH_MEMORY, if the pcOptions buffer is not enough to add the new key/value pair. The required size of the buffer is returned in pdwLength.

The minimum size of the buffer should be equal to MTMICR_OPTIONS_BUFFER_SIZE.

The **MTMICRSetValue** function saves the new key/value pair in the pcOptions buffer only. This function does not send this key/value pair to the device. Use function **MTMICRProcessCheck** to send this key/value pair to the device.

If pcKey is NULL, the whole section pcSection will be removed from pcOptions string.
If pcValue is NULL and a key is specified, the specified key will be removed from the specified section
If pdwLength is less than the size of the results pcOptions after new key/value pair is added then
MICR_ST_NOT_ENOUGH_MEMORY is returned

Example

```
char Settings [4096];
DWORD SettingsBufferSize;

// Initialize Settings

// Initialize the Settings variable first
SettingsBufferSize =4096;

// Set a value of "4" to the key "Number" in section "ImageOptions"and store this new
key/value pair in the buffer" Settings"
dwStatus=MTMICRSetValue(Settings, "ImageOptions","Number","4", & SettingsBufferSize);
```

MTMICRSetIndexValue

MTMICRSetIndexValue is similar to the **MTMICRSetVaue**, except that, **MTMICRSetIndexValue** adds the Index number to the pcKey before adding the key/value pair to the given Options buffer.

ULONG MTMICRSetIndexValue (

char	*pcOptions,
char	*pcSection,
char	*pcKey,
unsigned int	nIndex,
char	*pcValue,
DWORD	*pdwLength

);

Parameters

pcOptions
Pointer to null terminated string containing all key/value pairs.
pcSection
Pointer to null terminated string containing the section name.
pcKey
Pointer to null terminated string containing the key name.
nIndex
Index of the key.
pcValue
Pointer to null terminated string containing the key value.
pdwLength:
Pointer to a double word that contains the size of the pcOptions buffer.

Return Values

MICR_ST_OK
MICR_ST_NOT_ENOUGH_MEMORY
MICR_ST_BAD_PARAMETER
MICR_ST_BAD_DATA
MICR_ST_BAD_SECTION_NAME
MICR_ST_BAD_KEY_NAME
MICR_ST_BAD_VALUE_BUFFER
MICR_ST_BAD_BUFFER_LENGTH

Remarks

The function returns MICR_ST_NOT_ENOUGH_MEMORY, if the memory allocated for pcOptions buffer is not big enough to store the additional key/value pair. The required size for the buffer is returned in pdwLength;
The minimum size of the buffer should be equal to MTMICR_OPTIONS_BUFFER_SIZE.
The **MTMICRSetIndexValue** function saves the new key/value pair in the pcOptions buffer only. This function does not send the new key/value pair to the device. Use function **MTMICRProcessCheck** to send this key/value pair to the device.
If parameter **nIndex is less than zero or greater than 4 then** MICR_ST_BAD_PARAMETER is returned.
If pdwLength is less than the length of the results pcOptions string after new key/value pair is added then
MICR_ST_NOT_ENOUGH_MEMORY is returned.

Example

```
char Settings [4096];
DWORD SettingsBufferSize;

// Initialize Settings

// Intialize the Settings variable first
SettingsBufferSize =4096;

// The following command will set ImageColor1 = BW under the ImageOptions section.
dwStatus=MTMICRSetIndexValue(Settings, "ImageOptions","ImageColor",1, "BW",
&SettingsBufferSize);
```

MTMICRGetValue

MTMICRGetValue function retrieves a key/value pair that was previously stored in the pcDocInfo parameter using **MTMICRSetValue** function .

ULONG MTMICRGetValue (
char *pcDocInfo,
char *pcSection,
char *pcKey,
char *pcValue,
DWORD *pdwLength
);

Parameters

pcDocInfo
Buffer pointer containing all the key/value pairs.

pcSection
Pointer to null terminated string containing the section name.

pcKey
Pointer to null terminated string containing the key name.

pcValue
Pointer to the buffer that receives the retrieved string.

pdwLength
Specifies the size of the pcValue buffer.

Return Values

MICR_ST_OK
MICR_ST_NOT_ENOUGH_MEMORY
MICR_ST_ERR_LOAD_XML
MICR_ST_ERR_GET_DOM_POINTER
MICR_ST_BAD_DATA
MICR_ST_BAD_SECTION_NAME
MICR_ST_BAD_KEY_NAME
MICR_ST_BAD_VALUE_BUFFER
MICR_ST_BAD_BUFFER_LENGTH
MICR_ST_KEY_NOT_FOUND

Remarks

MTMICRGetValue finds the key in the pcDocInfo buffer then returns its value in the pcValue.
If **MTMICRGetValue** succeeds it returns MICR_ST_OK.
If pdwLength is less than the size of the returned value then MICR_ST_NOT_ENOUGH_MEMORY is returned and the required size for the pcValue buffer is returned in the pdwLength.
If the key/value pair can not be found, then a NULL is returned for pcValue and error parameter MICR_ST_KEY_NOT_FOUND is returned.

Example

```
char Settings [4096];
char DocInfo [4096];
char device[4096] ="";
DWORD SettingsBufferSize;
DWORD DocInfoSize;
char cValue [1024];
DWORD valueSize;
DWORD dwStatus;

// Initialize Settings

DocInfoSize = 4096;

// Use function MTMICRGetDevice to get device name for variable "device"
// Call MTMICRProcessCheck function to process a document.
dwStatus = MTMICRProcessCheck (device, Settings, DocInfo, &DocInfoSize);

if (dwStatus == MTMICR_ST_OK)
{

      //Let us check the return status from the device
      valueSize = 1024;
      dwStatus=MTMICRGetValue(DocInfo, "CommandStatus", "ReturnCode", cValue,
&valueSize);
      if (dwStatus != MICR_ST_OK)
            // error retrieving key value
      else
      {
            // do further process
      }
}
```

MTMICRGetIndexValue

MTMICRGetIndexValue function retrieves a key/value pair that was previously stored in the pcDocInfo parameter. **MTMICRGetIndexValue** function is similar to the function **MTMICRGetValue**. **MTMICRGetIndexValue** function adds index to the key name before searching for the value of the key name pcKey in the pcDocInfo.

```
ULONG MTMICRGetIndexValue (
char            *pcDocInfo,
char            * pcSection,
char            *pcKey,
unsigned int    nIndex,
char            *pcValue,
DWORD           *pdwLength
);
```

Parameters

pcDocInfo
: Buffer pointer containing all the key/value pairs.

pcSection
: Pointer to null terminated string containing the section name.

pcKey
: Pointer to null terminated string containing the key name.

nIndex
: Key Index Number

pcValue
: Pointer to the buffer that receives the retrieved value.

pdwLength
: Specifies the size of the buffer pointed to by the pcValue parameter.

Return Values

MICR_ST_OK
MICR_ST_NOT_ENOUGH_MEMORY
MICR_ST_ERR_LOAD_XML
MICR_ST_ERR_GET_DOM_POINTER
MICR_ST_BAD_DATA
MICR_ST_BAD_SECTION_NAME
MICR_ST_BAD_KEY_NAME
MICR_ST_BAD_VALUE_BUFFER
MICR_ST_BAD_BUFFER_LENGTH
MICR_ST_KEY_NOT_FOUND

Example

```
char Settings [4096];
char DocInfo [4096];
char device[4096] ="";
DWORD SettingsBufferSize;
DWORD DocInfoSize;
char cValue [1024];
DWORD valueSize;
DWORD dwStatus;

// Intialize Settings

DocInfoSize = 4096;

// Use function MTMICRGetDevice to get device name for variable "device"
// Call MTMICRProcessCheck function to process a document.
dwStatus = MTMICRProcessCheck (device, Settings, DocInfo, &DocInfoSize);

if (dwStatus == MTMICR_ST_OK)
{

      //Let us check the return status from the device
      valueSize = 1024;
      dwStatus=MTMICRGetValue(DocInfo, "CommandStatus", "ReturnCode", cValue,
&valueSize);
      if (dwStatus != MICR_ST_OK)
            // error retrieving key value
      else
      {
            // Get the key ImageSize1 under ImageInfo section
            dwStatus=MTMICRGetIndexValue(DocInfo, "ImageInfo", "ImageSize",1, cValue,
                                                                  &valueSize);
            if (dwStatus == MICR_ST_OK)
            {
            }
      }
}
```

MTMICRQueryInfo

MTMICRQueryInfo function inquires data of a given section name from the given device name.

ULONG MTMICRQueryInfo (

	char	*pcDevName,
	char	*pcSection,
	char	*pcSectionData,
	DWORD	*pdwLength

);

Parameters

pcDevName
Pointer to null terminated string containing device name.

pcSection
Pointer to null terminated string containing the section name. e.g. DeviceCapabilities, DeviceUsage, DeviceStatus, etc.

pcSectionData
Pointer to the buffer that is used to store the data of the inquiry section.

pdwLength
Specifies the size of the pcSectionData buffer.

Return Values

MICR_ST_OK
MICR_ST_DEVICE_NOT_OPEN
MICR_ST_DEVICE_NOT_RESPONDING
MICR_ST_DEVICE_CONNECTION_ERROR
MICR_ST_REQUEST_TIMEDOUT
MICR_ST_CONNECT_REQUEST_TIMEDOUT
MICR_ST_ERR_INTERNET_CONNECT
MICR_ST_ERR_HTTP_OPEN_REQUEST
MICR_ST_ERR_HTTP_SEND_REQUEST
MICR_ST_NOT_ENOUGH_MEMORY
MICR_ST_BAD_DEVICE_NAME
MICR_ST_BAD_QUERY_PARM
MICR_ST_BAD_BUFFER
MICR_ST_BAD_BUFFER_LENGTH
MICR_ST_USB_GET_DATA_FAILED
MICR_ST_INET_GET_DATA_FAILED

Remarks

If the function succeeds MICR_ST_OK is returned.
If the device fails to respond, MICR_ST_DEVICE_NOT_RESPONDING is returned.
If the memory allocated for pcSectionData buffer that is not big enough to store the data of the inquiry section,
MICR_ST_NOT_ENOUGH_MEMORY is returned.

Error results from bad connection with device can be one of the following:
MICR_ST_CONNECT_REQUEST_TIMEDOUT
MICR_ST_REQUEST_TIMEDOUT
MICR_ST_DEVICE_NOT_RESPONDING
MICR_ST_ERR_INTERNET_CONNECT
MICR_ST_ERR_HTTP_OPEN_REQUEST
MICR_ST_ERR_HTTP_SEND_REQUEST
MICR_ST_USB_GET_DATA_FAILED
MICR_ST_INET_GET_DATA_FAILED

Example

```
#define BUFFER_LEN 4096
char pResult [BUFFER_LEN];
DWORD resultLength;
resultLength = BUFFER_LEN;
MTMICRQueryInfo(pcDeviceName, "DeviceCapabilities",pResult,&resultLength);
```

MTMICRSendCommand

MTMICRSendCommand function is used to send a single command to the device.

ULONG MTMICRSendCommand (

```
    char        *pcDevName,
    char        *pcCommand,
    char        *pcResult,
    DWORD       *pdwLength
);
```

Parameters

pcDevName
Pointer to null terminated string containing device name.

pcCommand
Pointer to null terminated string containing the command string. There are two commands available: command "ClearPathExit" and command "ClearPathEntry."

pcResult
Pointer to the buffer that is used to store results returning from device.

pdwLength
Specifies the size of the pcResult buffer.

Return Values

MICR_ST_OK
MICR_ST_DEVICE_NOT_OPEN
MICR_ST_DEVICE_NOT_RESPONDING
MICR_ST_DEVICE_CONNECTION_ERROR
MICR_ST_REQUEST_TIMEDOUT
MICR_ST_CONNECT_REQUEST_TIMEDOUT
MICR_ST_ERR_INTERNET_CONNECT
MICR_ST_ERR_HTTP_OPEN_REQUEST
MICR_ST_ERR_HTTP_SEND_REQUEST
MICR_ST_NOT_ENOUGH_MEMORY
MICR_ST_ERR_CREATE_EVENT
MICR_ST_BAD_DEVICE_NAME
MICR_ST_BAD_PARAMETER
MICR_ST_INSUFFICIENT_DISKSPACE
MICR_ST_USB_GET_DATA_FAILED
MICR_ST_INET_GET_DATA_FAILED
MICR_ST_HTTP_HEADER_NOT_FOUND

Remarks

When sending command "ClearPathExit" or command "ClearPathEntry" to the device, the device is trying to clear the document path. If there is document in the path, the document is removed through the exit point if the command is "ClearPathExit" or the document is removed through the entry point if the command is "ClearPathEntry."

Example

```
char szResult [4096];
char szDeviceName[4096];
char Buffer[256];
DWORD dwBufferLength ;
DWORD dwStatus;

szDeviceName = "Excella"; // assuming this device has been opened

dwBufferLength = 4096;

if (MTMICRSendCommand(szDeviceName, "ClearPathEntry", szResult , & dwBufferLength)==
MICR_ST_OK)
{
       dwBufferLength = 256;
       dwStatus = MTMICRGetValue(szResult, SECTION_COMMAND_STATUS,"ReturnCode",
Buffer,
                                                                    & dwBufferLength);
      if (dwStatus == MICR_ST_OK)
      {
            dwStatus = atol (Buffer);

            if (dwStatus == 0)
                  prinft ("Clear path done!");
            else
                  .printf ("Clear path failed!");

      }
}
```

MTMICRProcessCheck

MTMICRProcessCheck function sends a scan check request with the given process options to the given device name. When the device completes the scan request, the result of the scan is returned in the buffer pcDocInfo.

ULONG MTMICRProcessCheck (

char	*pcDevName,
char	*pcProcessOptions,
char	*pcDocInfo,
DWORD	*pdwDocInfoSize

);

Parameters

pcDevName
Pointer to null terminated string containing device name.

pcProcessOptions
Pointer to a buffer containing the options to be used in processing the check. The options are stored in the buffer by using function **MTMICRSetValue** or function **MTMICRSetIndexValue**.

pcDocInfo
Pointer to the buffer containing the information returned from the device. The returned information contains command status, MICR data, and image information.

pdwDocInfoSize
Size of the buffer pcDocInfo.

Return Values

MICR_ST_OK
MICR_ST_NOT_ENOUGH_MEMORY
MICR_ST_PROCESS_CHECK_FAILED
MICR_ST_BAD_DATA
MICR_ST_BAD_BUFFER
MICR_ST_BAD_BUFFER_LENGTH
MICR_ST_BAD_DEVICE_NAME
MICR_ST_DEVICE_NOT_OPEN
MICR_ST_DEVICE_NOT_RESPONDING
MICR_ST_REQUEST_TIMEDOUT
MICR_ST_CONNECT_REQUEST_TIMEDOUT
MICR_ST_ERR_INTERNET_CONNECT
MICR_ST_ERR_HTTP_OPEN_REQUEST
MICR_ST_ERR_HTTP_SEND_REQUEST
MICR_ST_INET_GET_DATA_FAILED
MICR_ST_USB_GET_DATA_FAILED
MICR_ST_HTTP_HEADER_NOT_FOUND

Remarks

If the function succeeds MICR_ST_OK is returned.
If there is no data returns from device, error MICR_ST_PROCESS_CHECK_FAILED is returned
If pcProcessOptions is NULL, error MICR_ST_BAD_DATA is returned
If pcDocInfo is NULL, error MICR_ST_BAD_BUFFER is returned
If the size of returned data is larger than the value specified in pdwDocInfoSize, error
MICR_ST_NOT_ENOUGH_MEMORY is returned
If the function fails to get HTTP header information, MICR_ST_HTTP_HEADER_NOT_FOUND is returned.
Error results from bad connection with device can be one of the following:
MICR_ST_CONNECT_REQUEST_TIMEDOUT
MICR_ST_REQUEST_TIMEDOUT
MICR_ST_DEVICE_NOT_RESPONDING
MICR_ST_ERR_INTERNET_CONNECT
MICR_ST_ERR_HTTP_OPEN_REQUEST
MICR_ST_ERR_HTTP_SEND_REQUEST
MICR_ST_USB_GET_DATA_FAILED
MICR_ST_INET_GET_DATA_FAILED

Example

```
#define BUFFER_LEN 4096

char docInfo [BUFFER_LEN];
char options [BUFFER_LEN];
char Device[4096] ="";
DWORD docInfoSize;;

// Set up options using function MTMICRSetValue or function MTMICRSetIndexValue
// Use function MTMICRGetDevice to get device name for variable "Device"

docInfoSize = BUFFER_LEN;
dwStatus=MTMICRProcessCheck(Device, options, docInfo, &docInfoSize);

// Use MTMICRGetValue and MTMICRGetIndexValue to parse the docInfo.
```

MTMICRGetImage

MTMICRGetImage function sends to the given device name a request for image data results from a previously scan.

ULONG MTMICRGetImage (
char *pcDevName,
char *pcImageID,
char *pcBuffer,
DWORD *pdwLength
);

Parameters

pcDevName
Pointer to null terminated string containing device name.

pcImageID
the identification of the requested image

pcBuffer
Pointer to buffer for storing the image data.

pdwLength
Specifies the size of the pcBuffer.

Return Values

MICR_ST_OK
MICR_ST_DEVICE_NOT_RESPONDING
MICR_ST_IMAGE_NOT_FOUND
MICR_ST_NOT_ENOUGH_MEMORY
MICR_ST_UNKOWN_ERROR
MICR_ST_BAD_BUFFER
MICR_ST_BAD_IMAGE_NAME
MICR_ST_BAD_DEVICE_NAME
MICR_ST_BAD_BUFFER_LENGTH
MICR_ST_DEVICE_NOT_OPEN
MICR_ST_CONNECT_REQUEST_TIMEDOUT
MICR_ST_REQUEST_TIMEDOUT
MICR_ST_DEVICE_NOT_RESPONDING
MICR_ST_ERR_INTERNET_CONNECT
MICR_ST_ERR_HTTP_OPEN_REQUEST
MICR_ST_ERR_HTTP_SEND_REQUEST
MICR_ST_USB_GET_DATA_FAILED
MICR_ST_INET_GET_DATA_FAILED

Remarks

If the function succeeds, MICR_ST_OK is returned.
If the requested image is not found, MICR_ST_IMAGE_NOT_FOUND is returned.
If the device fails to respond to the command, the return value is MICR_ST_DEVICE_NOT_RESPONDING.
If the size of the buffer uses to store the image is not large enough, MICR_ST_NOT_ENOUGH_MEMORY is returned.

Error results from bad connection with device can be one of the following:

MICR_ST_CONNECT_REQUEST_TIMEDOUT
MICR_ST_REQUEST_TIMEDOUT
MICR_ST_DEVICE_NOT_RESPONDING
MICR_ST_ERR_INTERNET_CONNECT
MICR_ST_ERR_HTTP_OPEN_REQUEST
MICR_ST_ERR_HTTP_SEND_REQUEST
MICR_ST_USB_GET_DATA_FAILED
MICR_ST_INET_GET_DATA_FAILED

Example

```
#define BUFFER_LEN 512
// DocInfo is returned by the MTMICRProcessCheck()

void GetNthImages (char *Device, unsigned int nIndex, char *DocInfo)
{
      char key [512];
      DWORD bufferSize;
      char cValue[BUFFER_LEN]="";
      DWORD nValueSize;

      // Get the image size from DocInfo
      nValueSize = BUFFER_LEN;
      MTMICRGetIndexValue (DocInfo, "ImageInfo","ImageSize", nIndex,  cValue, &
nValueSize);

      // Convert size to integer
      bufferSize = atol (cValue);

      // Get the Image Id from DocInfo
      nValueSize = BUFFER_LEN;
      MTMICRGetIndexValue (DocInfo, "ImageInfo","ImageSize, nIndex, cValue, &
nValueSize);
      if (bufferSize)
      {
            // Allocate memory for image
            ImageBuffer = (char *) VirtualAlloc (NULL, bufferSize, MEM_COMMIT,
                                                 PAGE_READWRITE);

            // Use function MTMICRGetDevice to get device name for variable "Device"
            // Get the image from the device
            MTMICRGetImage (Device, cValue, ImageBuffer, &bufferSize);
            .
            .
            // Free the memory
            VirtualFree (ImageBuffer, bufferSize, MEM_DECOMMIT);
      }
}
```

MTMICRGetImages

MTMICRGetImages is similar to **MTMICRGetImage** except **MTMICRGetImages** can get multiple images in one function call.

ULONG MTMICRGetImages (
char *pcDevName,
GET_IMAGE *pGetImage,
DWORD dwTotalImages
);

Parameters

pcDevName

Pointer to null terminated string containing device name.

pGetImage

Pointer to a list of GET_IMAGE structures containing the image information.

```
typedef struct _GET_IMAGE
{
    char *pcImageId;
    char *pcBuffer;
    DWORD dwBufferLength;
    DWORD dwBytesReturned;
    DWORD dwStatus;
} GET_IMAGE;
```

pcImageId = ImageURLx returned by the **MTMICRProcessCheck** in the document information. Where x is the image index.

pcBuffer = Allocate memory required to store the image. The image size is also returned in the document information from the **MTMICRProcessCheck**.

dwBufferLength = Size of pcBuffer in bytes.

dwBytesReturned = **MTMICRGetImages** will store the number of bytes returned by the device.

dwStatus = **MTMICRGetImages** fills the status of each image.

dwTotalImages=Number of Images requested.

Return Values

MICR_ST_OK
MICR_ST_DEVICE_NOT_RESPONDING
MICR_ST_IMAGE_NOT_FOUND
MICR_ST_NOT_ENOUGH_MEMORY
MICR_ST_ERR_INTERNET_CONNECT
MICR_ST_ERR_HTTP_OPEN_REQUEST
MICR_ST_ERR_HTTP_SEND_REQUEST
MICR_ST_DEVICE_CONNECTION_ERROR
MICR_ST_BAD_PARAMETER
MICR_ST_BAD_DEVICE_NAME
MICR_ST_BAD_BUFFER
MICR_ST_DEVICE_NOT_OPEN
MICR_ST_BAD_BUFFER_SIZE
MICR_ST_BAD_IMAGE_NAME
MICR_ST_CONNECT_REQUEST_TIMEDOUT
MICR_ST_REQUEST_TIMEDOUT
MICR_ST_INSUFFICIENT_DISKSPACE
MICR_ST_QUERY_CONTENT_ERROR
MICR_ST_USB_GET_DATA_FAILED
MICR_ST_INET_GET_DATA_FAILED
MICR_ST_HTTP_HEADER_NOT_FOUND

Remarks

If the function succeeds, MICR_ST_OK is returned.
If the requested image is not found, MICR_ST_IMAGE_NOT_FOUND is returned.
If the function fails to get data packet information, MICR_ST_QUERY_CONTENT_ERROR is returned.
If the function fails to get HTTP header information, MICR_ST_HTTP_HEADER_NOT_FOUND is returned.
If the device fails to respond to the command, the return value is MICR_ST_DEVICE_NOT_RESPONDING.
If the size of the buffer uses to store the image data is not large enough, MICR_ST_NOT_ENOUGH_MEMORY is returned.
Error results from bad connection with device can be one of the following:

MICR_ST_CONNECT_REQUEST_TIMEDOUT
MICR_ST_REQUEST_TIMEDOUT
MICR_ST_DEVICE_NOT_RESPONDING
MICR_ST_ERR_INTERNET_CONNECT
MICR_ST_ERR_HTTP_OPEN_REQUEST
MICR_ST_ERR_HTTP_SEND_REQUEST
MICR_ST_USB_GET_DATA_FAILED
MICR_ST_INET_GET_DATA_FAILED

Example

```
#define BUFFER_LEN 512

// DocInfo is returned by the MTMICRProcessCheck()
#define BUFFER_LEN 4096

GET_IMAGE getImages [4];
DWORD dwSize = BUFFER_LEN;
char szData [BUFFER_LEN];
DWORD dwStatus;

for (int i = 1; i < 5; i++) {
      // Get the image size for image number i

     dwStatus = MTMICRGetIndexValue (DocInfo, "Image Info", "Image Size", i, szData,
& dwSize);

     if (dwStatus == MICR_ST_OK) {
      // Convert the size to integer
      dwSize = atol (szData);
      if (dwSize < = 0)
           continue;

      dwSize = BUFFER_LEN;

            // Get the image id
              dwStatus = MTMICRGetIndexValue (DocInfo, "Image Info", "Image URL", i,
szData, & dwSize);
          if (dwStatus  == MICR_ST_OK)) {
            char *p = (char *) malloc (strlen (szData)+1);
            getImages [i-1].pcImageId =  p;
            strcpy (getImage [i-1]. pcImageId, szData);
            getImages  [i-1].dwBufferLength = bufferSize;
            getImages  [i-1].pcBuffer =
            (char *) VirtualAlloc (NULL, getImages [i-1].dwBufferLength,
      MEM_COMMIT,
                                                    PAGE_READWRITE);
      }
       }
}
// Use function MTMICRGetDevice to get device name for variable "Device"
error = MTMICRGetImages (Device, getImages, 4);
```

MTMICRGETSECTIONCOUNT

MTMICRGetSectionCount function returns the number of section present in a null terminated buffer which contains a set of key/value pairs. The key/value pairs get stored in the buffer using function **MTMICRSetValue** or **MTMICRSetIndexValue**.

ULONG MTMICRGetSectionCount (

char	*pcData,
DWORD	*pdwSectionCount

);

Parameters

pcData
Pointer to null terminated string containing a set of key/value pairs.

pdwSectionCount
When the function returns, this variable contains the number of sections present in the pcData.

Return Values

MICR_ST_OK
MICR_ST_BAD_PARAMETER
MICR_ST_ERR_GET_DOM_POINTER
MICR_ST_ERR_LOAD_XML
MICR_ST_SECTION_NOT_FOUND
MICR_ST_BAD_DATA
MICR_ST_BAD_BUFFER_LENGTH

Remarks

If the function succeeds MICR_ST_OK is returned.
If pcData is NULL, MICR_ST_BAD_DATA is returned.
If data in pcData string is not in XML format, MICR_ST_ERR_LOAD_XML is returned.
If there is problem using MSXML, MICR_ST_ERR_GET_DOM_POINTER is returned.

Example

```
char Settings [4096];
char DocInfo [4096];
char device[4096] ="";
DWORD SettingsBufferSize;
DWORD DocInfoSize;
char cValue [1024];
DWORD valueSize;
DWORD dwStatus;

// Intialize Settings

DocInfoSize = 4096;

// Use function MTMICRGetDevice to get device name for variable "device"
// Call MTMICRProcessCheck function to process a document.
dwStatus = MTMICRProcessCheck (device, Settings, DocInfo, &DocInfoSize);

if (dwStatus == MICR_ST_OK)
{
      // Let us scan through all the key pair values returned in the DocInfo
      DWORD sectionCount;

      // Get total number of sections
      if (MTMICRGetSectionCount (DocInfo, &sectionCount) ==MICR_ST_OK)
      {
      }

}
```

MTMICRGetSectionName

MTMICRGetSectionName function returns the name of the section which has the section number given in variable dwSectionNumber. **MTMICRGetSectionName** parses through the buffer pcData containing a set of key/value pairs which was previously stored in the buffer by using function MTMICRSetValue or function **MTMICRSetIndexValue**.

ULONG MTMICRGetSectionName (

char	*pcData,
DWORD	dwSectionNumber,
char	*pcSectionName,
DWORD	*pdwSectionNameSize

);

Parameters

pcData
Pointer to null terminated string containing a set of key/value pairs..

dwSectionNumber
The section number of the section requesting for name.

pcSectionName
Pointer to a buffer uses to store the section name.

pdwSectionNameSize
Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the *pcSectionName* parameter. When the function returns, this variable contains the size of the data copied to *pcSectionName*. If the buffer specified by *pcSectionName* parameter is not large enough to hold the data, the function returns the value MICR_ST_NOT_ENOUGH_MEMORY, and stores the required buffer size, in bytes, into the variable pointed to by pdwSectionNameSize.

Return Values

MICR_ST_OK
MICR_ST_ERR_GET_DOM_POINTER
MICR_ST_ERR_LOAD_XML
MICR_ST_SECTION_NOT_FOUND
MICR_ST_BAD_DATA
MICR_ST_BAD_BUFFER_LENGTH
MICR_ST_BAD_SECTION_NAME
MICR_ST_NOT_ENOUGH_MEMORY

Remarks

If the function succeeds MICR_ST_OK is returned.
If pdwSectionNameSize is NULL, error MICR_ST_BAD_BUFFER_LENGTH is returned.
If the value of pdwSectionNameSize is not big enough to hold the name of requested section,
MICR_ST_NOT_ENOUGH_MEMORY is returned.
If pcData is NULL, MICR_ST_BAD_DATA is returned.
If data in pcData string is not in XML format, MICR_ST_ERR_LOAD_XML is returned.
If there is problem using MSXML, MICR_ST_ERR_GET_DOM_POINTER is returned.

Example

```
char Settings [4096];
char DocInfo [4096];
char device[4096] ="";
DWORD SettingsBufferSize;
DWORD DocInfoSize;
char cValue [1024];
DWORD valueSize;
DWORD dwStatus;

// Intialize Settings

DocInfoSize = 4096;

// Use function MTMICRGetDevice to get device name for variable "device"
// Call MTMICRProcessCheck function to process a document.
dwStatus = MTMICRProcessCheck (device, Settings, DocInfo, &DocInfoSize);

if (dwStatus == MICR_ST_OK)
{
      // Let us scan through all the key pair values returned in the DocInfo
     DWORD sectionCount;
     DWORD sectionIndex;
      // Get total number of sections
      if (MTMICRGetSectionCount (DocInfo, &sectionCount) ==MICR_ST_OK) {

      for (sectionIndex = 0; sectionIndex < sectionCount; sectionIndex) {
           char SectionName [128];
           DWORD SectionNameSize = 128;
           dwStatus = MTMICRGetSectionName(DocInfo, sectionIndex, SectionName,
&SectionNameSize);
           if (dwStatus == MICR_ST_OK)
           {
                    }
              }

      }

}
```

MTMICRGetKeyCount

MTMICRGetKeyCount function returns the number of keys that belong to a section with the name specified in variable pcSection. The function **MTMICRGetKeyCount** parses through a null terminated string which contains a set of key/value pairs. The key/value pairs get stored in the buffer using function **MTMICRSetValue** or **MTMICRSetIndexValue**.

ULONG MTMICRGetKeyCount (

char	*pcData,
char	*pcSection,
DWORD	*pdwKeyCount

);

Parameters

pcData
: Pointer to null terminated string containing a set of key/value pairs.

pcSection
: Pointer to null terminated string containing the section name.

pdwKeyCount
: When the function returns, this variable contains the number of keys under the section name pcSection present in the pcData.

Return Values

MICR_ST_OK
MICR_ST_ERR_GET_DOM_POINTER
MICR_ST_ERR_LOAD_XML
MICR_ST_SECTION_NOT_FOUND
MICR_ST_BAD_DATA
MICR_ST_BAD_SECTION_NAME
MICR_ST_BAD_KEY_NAME

Remarks

If the function succeeds MICR_ST_OK is returned.
If pcData is NULL, MICR_ST_BAD_DATA is returned.
If data in pcData string is not in XML format, MICR_ST_ERR_LOAD_XML is returned.
If there is problem using MSXML, MICR_ST_ERR_GET_DOM_POINTER is returned

Example

```
char Settings [4096];
char DocInfo [4096];
char device[4096] ="";
DWORD SettingsBufferSize;
DWORD DocInfoSize;
char cValue [1024];
DWORD valueSize;
DWORD dwStatus;

// Intialize Settings

DocInfoSize = 4096;

// Use function MTMICRGetDevice to get device name for variable "device"
// Call MTMICRProcessCheck function to process a document.
dwStatus = MTMICRProcessCheck (device, Settings, DocInfo, &DocInfoSize);

if (dwStatus == MICR_ST_OK){
      // Let us scan through all the key pair values returned in the DocInfo
      DWORD sectionCount;
      DWORD sectionIndex;

      // Get total number of sections
      if (MTMICRGetSectionCount (DocInfo, &sectionCount) ==MICR_ST_OK)      {
      for (sectionIndex = 0; sectionIndex < sectionCount; sectionIndex)   {
             char SectionName [128];
            DWORD SectionNameSize = 128;
            dwStatus = MTMICRGetSectionName(DocInfo, sectionIndex, SectionName,
      &SectionNameSize);
            if (dwStatus == MICR_ST_OK)            {
            // Display the section Name
            DWORD keyCount;
            if (MTMICRGetKeyCount (DocInfo, SectionName, &keyCount) == MICR_ST_OK){
                  // Display the key count
            }
            }

            }
      }
}
```

MTMICRGetKeyName

MTMICRGetKeyName function returns the name of the key which has the key index number given in variable dwKeyNumber and belongs to the section group with the section name specified in variable pcSection. **MTMICRGetKeyName** parses through the buffer pcSettings containing a set of key/value pairs which was previously stored in the buffer by using function MTMICRSetValue or function **MTMICRSetIndexValue**.

ULONG MTMICRGetKeyName (

char	*pcSettings,
char	*pcSection,
DWORD	dwKeyNumber,
char	*pcKey,
DWORD	*pdwKeySize

);

Parameters

pcSettings
Pointer to null terminated string containing a set of key/value pairs.

pcSection
Pointer to null terminated string containing the section name.

dwKeyNumber
Specifes the key index number of the requested key.

pcKey
When the function returns, this variable contains key name of the key at index dwKeyNumber.

pdwKeySize
Pointer to a variable that specifies the size, in bytes, of the buffer pcKey. If the size of the buffer uses to store the name of the requested key, the required size is stored in this variable when the function completes execution.

Return Values

MICR_ST_OK
MICR_ST_ERR_GET_DOM_POINTER
MICR_ST_ERR_LOAD_XML
MICR_ST_SECTION_NOT_FOUND
MICR_ST_KEY_NUMBER_NOT_FOUND
MICR_ST_NOT_ENOUGH_MEMORY
MICR_ST_BAD_BUFFER_LENGTH
MICR_ST_BAD_DATA
MICR_ST_BAD_SECTION_NAME
MICR_ST_BAD_BUFFER

Remarks

If the function succeeds MICR_ST_OK is returned.
If pcSettings is NULL, MICR_ST_BAD_DATA is returned.
If data in pcSettings string is not in XML format, MICR_ST_ERR_LOAD_XML is returned.
If there is problem using MSXML, MICR_ST_ERR_GET_DOM_POINTER is returned
If pdwKeySize is NULL, error MICR_ST_BAD_BUFFER_LENGTH is returned
If the value of pdwSectionNameSize is not big enough to hold the name of requested section,
MICR_ST_NOT_ENOUGH_MEMORY is returned

Example

```
char Settings [4096];
char DocInfo [4096];
char device[4096] ="";
DWORD SettingsBufferSize, DocInfoSize, valueSize, dwStatus , dwCount, nIndex ;
char cValue [1024];
DocInfoSize = 4096;
 char szSectionName [128];
DWORD dwSize = 128;
// Use function MTMICRGetDevice to get device name for variable "device"
// Call MTMICRProcessCheck function to process a document.
If ( MTMICRProcessCheck (device, Settings, DocInfo, &DocInfoSize) == MICR_ST_OK) {
      // Let us scan through all the key pair values returned in the DocInfo
      // Get total number of sections
      if (MTMICRGetSectionCount (DocInfo, &dwSectionCnt) ==MICR_ST_OK)      {
      for (nIndex = 0; nIndex < dwSectionCnt; nIndex++){
            If (MTMICRGetSectionName(DocInfo, nIndex, szSectionName, &dwSize  ==
MICR_ST_OK)  {
            // Get number of total keys which belong to section
            DWORD keyCount;
            if (MTMICRGetKeyCount (DocInfo, szSectionName, &keyCount) == MICR_ST_OK){
                  // Get name of each key
                  for (int keyIndex = 0; keyIndex < keyCount; keyIndex++)      {
                  char szKeyName [128];
                  dwStatus = MTMICRGetKeyName(DocInfo, szSectionName, keyIndex,
      szKeyName, & dwSize);
                  if (dwStatus == MICR_ST_OK)   {
                        // We now have the section and key, let get the value
                        char szValue [1024];
                        dwStatus = MTMICRGetValue(DocInfo, szSectionName, szKeyName,
      szValue, & dwSize);
                       if (dwStatus == MICR_ST_OK)
                       {
                       }
                       }
                }
               }
      }

       }

}
```

MTMICRSetTimeout

MTMICRSetTimeout function sets a duration for timeout in case there is a communication problem with the Excella device. **MTMICRProcessCheck**, **MTMICRGetImage**, **MTMICRGetImages**, and **MTMICRQueryInfo** functions use this timeout value to wait for the device to respond. This timeout should be greater than the total time required to wait for a check feed, check movement along the path, and image processing. A recommended time is 30 seconds.

Note

This time must always be longer than DocFeedTimeout when used (see ProcessOptions in Section 5)

ULONG MTMICRSetTimeout (
DWORD dwMilliSeconds
);

Parameters

dwMilliSeconds
Duration for timeout. The timeout is in milliseconds unit.

Return Values

No return value for this function.

Example

```
#define DEVICE_NAME_LEN 128
int i=1;
DWORD dwResult;
char cDeviceName[DEVICE_NAME_LEN]="";

while ((dwResult = MTMICRGetDevice(i,(char*) cDeviceName)) !=
MICR_ST_DEVICE_NOT_FOUND)
{
      if (MTMICROpenDevice (cDevicesNames) == MICR_ST_OK)
      {
            // Set timeout
            MTMICRSetTimeout (15000); // 15 seconds
            ///close the device
      }
      i++;
}
```

MTMICRGetTimeout

MTMICRGetTimeout function returns the set timeout in milliseconds. **MTMICRProcessCheck**, **MTMICRGetImage**, **MTMICRGetImages**, and **MTMICRQueryInfo** functions use this timeout value to wait for device to respond. By default the timeout is 5 seconds.

ULONG MTMICRGetTimeout (
DWORD *pdwMilliSeconds
);

Parameters

pdwMilliSeconds
Pointer to a DWORD. When the function completes, this variable contains the timeout in milliseconds

Return Values

No return value for this function.

Example

```
#define DEVICE_NAME_LEN 128
int i=1;
DWORD dwResult;
char cDeviceName[DEVICE_NAME_LEN]="";
while ((dwResult = MTMICRGetDevice(i,(char*) cDeviceName)) !=
MICR_ST_DEVICE_NOT_FOUND)
{
      if (MTMICROpenDevice (cDevicesNames) == MICR_ST_OK)
      {
            //Get timeout
            DWORD timeout;
            MTMICRGetTimeout (&timeout);
            ///close the device
      }
      i++;
}
```

MTMICRLogEnable

MTMICRLogEnable enable or disable the logging. If there is a desire to get a log file of all messages and errors, a valid log file handle must be set with function **MTMICRSetLogFileHandle** and also a TRUE value is set for parameter of function **MTMICRLogEnable.** By default, logging is disable.

void MTMICRLogEnable (
BOOL bEnable
);

Parameters

bEnable
a Boolean value indicates logging enabling or disabling request.

Return Values

No return value for this function.

Example

```
if (hLogFileHandle != NULL)
{
      MTMICRLogEnable(TRUE);
      MTMICRSetLogFileHandle (hLogFileHandle);
}
```

MTMICRSetLogFileHandle

MTMICRSetLogFileHandle specifies a handle for a log file**.** All messages and errors created by all API functions will be logged in the log file that has this specified handle. By default, there is no log file.

VOID MTMICRSetLogFileHandle (
HANDLE hFileHandle
);

Parameters

hFileHandle
This is a handle to a file. All the API functions will log errors and information to this log file.

Return Values

No value is returned from this function.

Example

```
#define DEVICE_NAME_LEN 128
int i=1;
DWORD dwResult;
HANDEL fileHandle;
char cDeviceName[DEVICE_NAME_LEN]="";
while ((dwResult = MTMICRGetDevice(i,(char*) cDeviceName)) !=
MICR_ST_DEVICE_NOT_FOUND)
{
      if (MTMICROpenDevice (cDevicesNames) == MICR_ST_OK)
      {
            //Get timeout
            DWORD timeout;
            HANDLE fileHandle;

            // Create log file and save the handle in fileHandle
            // Pass the handle of the log file
            MTMICRSetLogFileHandle (fileHandle);
            ///close the device

            // Close the log file

            // Close the device
      }
      i++;
}
      if (MTMICROpenDevice (cDevicesNames) == MICR_ST_OK)
      {
            // Process documents
            //close the device
            MTMICRCloseDevice (cDeviceName);
      }
      i++;
}
```

MTMICRSETLOGLEVEL

MTMICRSetLogLevel sets the desired level of logging details**.** There are four levels of details: DLL_INTERNAL, DLL_EXTERNAL, XML_DATA, IMAGE_DATA. By default, none is set.

VOID MTMICRSetLogLevel (
DWORD dwDebugLevel
);

Parameters

dwDebugLevel

This value specifies the type of logging details. The value is composed of one or more of the following:

DBGLOG_DLL_INTERNAL	= 0x0001
DBGLOG_DLL_EXTERNAL	= 0x0002
DBGLOG_DLL_XMLDATA	= 0x0004
DBGLOG_DLL_IMAGEDATA	= 0x0010

Return Values

No value is returned from this function.

Example

```
MTMICRSetLogLevel(DBGLOG_DLL_INTERNAL | DBGLOG_DLL_XMLDATA);
```

Remarks

If DLL_INTERNAL level is set, errors from API functions listed in file MTXMLMCR.h are logged.
If DLL_EXTERNAL level is set, errors from DLL functions which are not listed in MTXMLMCR.h are logged.
If DLL_XMLDATA level is set, the contents of XML document sent to device or received from device are logged,
If DLL_IMAGEDATA level is set, the binary data of an image is logged**.**

MTMICRCOMInitialize

MTMICRCOMInitialize indirectly executed the function CoInitialize() to enable MSXML COM Object to be instantiable. By default, CoInitialize() is executed when MTXMLMCR.DLL is loaded.

VOID MTMICRCOMInitialize (
    void
);

Parameters

NONE

Return Values

No value is returned from this function.

Example

```
MTMICRCOMInitialize();
```

MTMICRCOMUnInitialize

MTMICRCOMInitialize indirectly executed the function CoUninitialize(). By default, CoUninitialize() is executed when MTXMLMCR.DLL is unloaded.

VOID MTMICRCOMUnInitialize (
    void
);

Parameters

NONE

Return Values

No value is returned from this function.

Example

```
MTMICRCOMUnInitialize();
```

Remark

When this function is executed, subsequent API calls which require MSXML stop functioning.

MTMICRSetConfigFile

MTMICRSetConfigFile function **s**ets the path and file name of the API configuration file. Default filename is "MICRDEV.INI".

VOID MTMICRSetConfigFile (
LPSTR lpszFile
);

Parameters

lpszFile
Null terminated string containing full path to the configuration file.

Return Values

NONE

Example

```
MTMICRSetConfigFile("c:\myconfig.txt");
```

Remark

The file must be verified to be accessible before calling this function. Any opened devices should first have its connection closed and the **MTMICRGetDevice** function will need to be recalled to retrieve the new device name list.

SECTION 4. COMMANDS SENT TO DEVICE

Section 4 describes device commands that can be sent to the device using the API function **MTMICRSendCommand** (see Section 3). The device reply will include the *CommandStatus* and/or *DeviceStatus* sections (and their corresponding Key-Value pairs) as described in later chapters.

The command is sent using a string with the following format:

DeviceCommand=[*command*]&[*parameter=value*]&[*parameter=value*]

Note *- The total maximum string length is 255 characters.*

MSR COMMAND

MSR commands that can be sent to the MagneSafe Reader

MSR command is sent using a string with the following format:
MSRCommand=[*Command Number*][*Data Length*][*Data*]*[Carriage Return 0x0D]**

*For detailed information on [Command Number], [Data Length] and [Data] you can refer to PN 99875398 “COMMANDS” section page 20.
**Each command and response is composed of a series of readable ASCII characters followed by the ASCII character CR (0x0D). For detailed information on command format, refer to PN 99875398 page 21.

The response from the MagneSafe Reader is saved in <DeviceInformation><CommandStatus><ReturnMsg> key.
The response format is [Result Code][Data Length][Data]*

* For detailed information on the result code refer to PN 99875398 “COMMANDS” section page 20

If the command is NOT correct, the response will be 0200

SETLED Command

This command allows the application to control the state of the LED’s when the device is in the idle state. This command overrides the normal/default LED states.

When the device leaves the idle state to execute a transaction, the device resumes control with the normal/default states for the LED’s.

LEDn Parameter

This parameter defines the state for each LED. Each LED is assigned an individual parameter as follows: LED1 (left LED), LED2 (middle LED), and LED3 (right LED).

Each LED state consists of 4 phases, and each phase can be 1 of 4 options: R=Red, G=Green, A=Amber, N=None/Off. The LED state is then defined by a 4-character string with one character (R, G, A or N) for each of the 4 phases.

Examples:

- To define a solid green state for the left LED, the following string is used: LED1=GGGG.
- To define slow red/green blinking for the middle LED, the following string is used: LED2=RRGG.
- To define fast red/green blinking for the right LED, the following string is used: LED3=RGRG

When the device is idle, the state reported for each LED by the *DeviceStatus* will be:
LED1 = NNNN
LED2 = GGGG
LED3 = NNNN

LDURn Parameter

This parameter controls the time duration of the LED state. Each LED is assigned an individual parameter as follows: LDUR1 (left LED), LDUR2 (middle LED), and LDUR3 (right LED).
For specific time duration, a number in ms must be specified (e.g., 5000 for 5 seconds). For a continuous time duration, the value ON is used. To relinquish control and return to normal/default LED states, the value DEFAULT is used.

Example

The SetLED command string can define the state of up to three LED's. As an example, assume the following requirements to define the state of the three LED's:

- LED1: steady green, stays on continuously
- LED2: off, relinquishing control to device for normal/default states
- LED3: slow red/off blinking, duration of 5 seconds.

The complete command string would be as follows:

DeviceCommand=SetLED&LED1=GGGG&LDUR1=ON&LED2=NNNN&LDUR2=DEFAULT&LED3=RRNN&LDUR3=5000

SECTION 5. KEYS SENT TO DEVICE

Section 5 describes all the Sections (and their corresponding Key-Value pairs) to be sent by the application to the device for document processing. The application can send information the device using the following sections: *Application, ProcessOptions, Endorser, and ImageOptions.*

The following keys are included in the *Application* section:

- Transfer
- DocUnits

The following keys are included in the *ProcessOptions* section:

- ReadMICR
- Endorse
- RespondEarly
- DblPickDet
- DocFeed
- DocFeedTimeout
- KVErrStop
- MICRFmtCode
- Sequence
- ScanOnce

The following keys are included in the *Endorser* section:

- PrintData
- PrintFrontData
- PrintFont
- PrintFrontfont
- PrintStyle
- PrintFrontStyle
- PrintRate
- PrintFontSize
- PrintFrontFontSize
- BackXPosition
- FrontXPositon
- BackYPosition
- FrontYPosition
- YPositionOffset
- Virtual

The following keys are included in the *ImageOptions* section:

- Number
- ImageColor#
- Resolution#
- Compression#
- FileType#
- ImageSide#
- FilterB
- FilterG
- JPECQC
- JPECQG
- CalculateSHA1
- ScanLED1
- ScanLED2

The following keys are included in the MICROptions section:

- Threshold
- Quality

SECTION = Application

The *Application* section includes keys to configure some required characteristics of the unit.

Transfer

This key determines the file transfer method. This key is supported by Excella and Excella STX.

Values	Value Description
HTTP	Default: Transfer files using HTTP protocol
FTP	Transfer files to FTP server

DocUnits

This key determines the unit of measurement. This key is supported by Excella and Excella STX.

Values	Value Description
ENGLISH	Default: Dimensions in thousandths of an inch
METRIC	Dimensions in millimeters

SECTION = ProcessOptions

The *ProcessOptions* section includes keys that control various operational options during the transaction.

ReadMICR

This key selects the MICR font to be read on the check. This key is supported by Excella and Excella STX.

Values	Value Description
E13B	Decode only E13B character set
CMC7	Decode only CMC7 character set
ALL	Auto-detect and decode both E13B and CMC7
NO	Suppress MICR reading

Endorse

This key determines what type of printing is required on the check. This key is supported by Excella and Excella STX.

Values	Value Description
NO	No printing required
BACK	Request printing on the back (endorsing)
FRONT	Request printing on the front (franking). This value is valid for Excella STX only.
BOTH	Request printing on the back and the front (endorsing and franking). This value is valid for Excella STX only.

RespondEarly

This key determines if an early response is required from the device. This key is supported by Excella and Excella STX.

Values	Value Description
YES	Request device to respond as early as possible. Images may not be ready.
NO	Device responds only when all images are completely processed

DblPickDet

This key determines if double-pick detection is required during check processing. This key is supported by Excella only.

Values	Value Description
NO	Disable double-pick detection
YES	Enable double-pick detection

DocFeed

This key determines the input tray to be used for document processing. This key is supported by Excella and Excella STX.

Values	Value Description
ALL	Auto-detect & process check, ID card and Magstripe card. This value is valid for Excella STX with MagneSafe Reader only
MANUAL	Process check from single-feed input tray
AUTOFEED	Process checks from the auto feeder. This value is valid for Excella only.
ID	Process ID card from card input tray. This value is valid for Excella STX only.
MSR	Process Magstripe card from MSR. This value is valid for Excella STX only.

DocFeedTimeout

This key specifies the waiting period for a document to be fed. This key is supported by Excella STX only.

Values	Value Description
numeric value	Specify a time period in seconds

KVErrStop

This key determines the type of key/value errors that can halt transaction operation. This key is supported by Excella and Excella STX.

Values	Value Description
NONE	No restriction
VALS	Invalid values halt the transaction
KEYS	Invalid keys halt the transaction
KEYVALS	Invalid keys and values halt the transaction

MICRFmtCode

This key specifies the pre-defined output format to be use for MICR data. This key is supported by Excella and Excella STX.

Values	Value Description
numeric string	Specifies the format code number (refer to Appendix A)

Sequence

The *Sequence* key is used to control the movement of the check document in the path. All available sequences or check moves are described below. Each check move is identified by the value of the *Sequence* key. If the *Sequence* key is not specified or an invalid value is use, the normal check process is used. This key is supported by Excella and Excella STX.

Values	Value Description
SEQ_0	The check move starts at the entry point. The check moves forward and it is pinched at the exit point. At the beginning of this sequence, the LED flashes green to indicate that a check must be inserted. When the check is inserted, the LED is turned off. If requested, the following operations can occur during this sequence: scanning front image, reading MICR line, endorsing, franking, and scanning back image. The data available after this sequence can include the MICR line, front image, and back image. Information on the images includes the image size and the URL string which the host can use to request the images.
SEQ_1	The check move starts at the entry point. The check moves forward and it stops at a point near the back printer. At the beginning of this sequence, the LED flashes green to indicate that a check must be inserted. When the check is inserted, the LED is turned off. If requested, the following operations can occur during this sequence: scanning front image and reading MICR line. The data available after this sequence can include the MICR line and the front image. Information of the front image includes the image size and the URL string which the host can use to request the image. Invalid conditions: • This sequence move is valid only if the path is clear at the beginning of the move. A jam will be reported if a check is detected in the path. The check must be cleared off the path before the sequence can begin. • This sequence move is valid only if there is no request for endorsing or franking. If there is a print request, error 306 ("Selected Print is Invalid For Selected Sequence") is returned. • This sequence move is also valid only if the requested image is the front side. If there is a request for the back side image, error 405 ("Scan Side is Invalid For Selected Sequence Move") is returned.

Values	Value Description
SEQ_2	The check move starts at a point near the back printer. The check moves in reverse for a short distance, it then moves forward again, and it is pinched at the exit point. If requested, the following operations can occur during this sequence: endorsing, franking, and scanning back image. The data available after this sequence can include the back image. Information of the back image includes the image size and URL string which the host can use to request the image. Invalid conditions: • This sequence move is valid only if there is a check in the path near the back printer. If there is no check found, error 208 ("Cannot find doc at location for this sequence move") is returned. • This sequence move is valid only if the request for the image is on the back side. If the front side image is requested, error 405 ("Scan Side is Invalid For Selected Sequence Move") is returned.
SEQ_3	The check move starts at a point near the back printer. The check moves in reverse, and it is pinched at the entry point. There is no data available from this sequence. Invalid conditions: • This sequence move is valid only if there is a check in the path near the back printer. If there is no check found near back printer location, error 208 ("Cannot find doc at location for this sequence move") is returned. • The host receives an OK status if the move is successful or an error code if an error occurs.
SEQ_4	The check move starts at the exit point. The check moves in reverse to the entry point, it then moves forward again, and it is pinched at the exit point. If requested, the following operations can occur during this sequence: scanning front image, and scanning back image. The data available after this sequence can include the front image and the back image. Information of the images includes the image size and the URL string which the host can use to request the image. The following operations do not occur in this sequence: MICR reading, endorsing or franking. Invalid conditions: • If there is request for endorsing and/or franking, error 306 ("Selected Print is Invalid For Selected Sequence") is returned. • This sequence move is valid only if there is a check in the path at the exit point. If there is no check found, error 208 ("Cannot find doc at location for this sequence move") is returned.

Values	Value Description
SEQ_5	The check move starts at the exit point. The check moves in reverse to a point near the back printer, it then moves forward again, and it is pinched at the exit point. If requested, the following operations can occur during this sequence: endorsing, franking, and scanning back image. The data available after this sequence includes the back image. Information of the back image includes the image size and URL string which the host can use to request the image. This sequence is similar to sequence SEQ_2**,** except the move starts at exit point. Invalid conditions: • This sequence move is valid only if there is a check at the exit point. If there is no check found, error 208 ("Cannot find doc at location for this sequence move") is returned. • This sequence move is valid only if the request for the image is on the back side. If the front side image is requested, error 405 ("Scan Side is Invalid For Selected Sequence Move") is returned.
SEQ_6	If there is check found in the check path, the check moves in the forward direction and it is pinched at the exit point. This operation will not take place if there is no check found in the check path. No scanning, printing or MICR reading operations are performed during this sequence. Invalid Conditions: • This operation will not take place if there is no check found in the check path. • A check that is pinched at the exit point (after check processing is completed) is considered to be out of the path.
SEQ_7	If there is check found in the check path, the check moves in the reverse direction and it is pinched at the entry point. This operation will not take place if there is no check found in the check path. No scanning, printing or MICR reading operations are performed during this sequence. Invalid Conditions: • This operation will not take place if there is no check found in the check path. • A check that is pinched at the exit point (after check processing is completed) is considered to be out of the path.
SEQ_8	If there is check found in the check path, the check moves in the forward direction towards the exit point, and the check leaves the unit. Motor is always turned on to ensure the path is completely clear. No scanning, printing or MICR reading operations are performed during this sequence. Invalid Conditions: • This operation will not take place if there is no check found in the check path. • A check that is pinched at the exit point (after check processing is completed) is considered to be out of the path.

Values	Value Description
SEQ_9	If there is check found in the check path, the check moves in the reverse direction towards the entry point, and the check leaves the unit. Motor is always turned on to ensure the path is completely clear. No scanning, printing or MICR reading operations are performed during this sequence. Invalid Conditions: • This operation will not take place if there is no check found in the check path. • A check that is pinched at the exit point (after check processing is completed) is considered to be out of the path.
SEQ_10	If there is check found in the check path, the check moves to a point near the back printer. No scanning, printing or MICR reading operations are performed during this sequence.

ScanOnce

This setting is only for MagUSB express mode.

If this setting is YES, the unit will automatically scan the next check.
If this setting is NO, the unit will scan only one check whether or not there are any other checks in the auto feed tray.

Values	Value Description
YES	If ExpressEnabled is TRUE, the unit will automatically scan the next check.
NO	If ExpressEnabled is TRUE, the unit will scan only one check regardless of the number of checks in the auto feed tray.

SECTION = Endorser

The *Endorser* section includes keys that control various options for printing on the check.

PrintData

This key specifies the text to be printed on the back of the check (i.e., endorsement message). This key is supported by Excella and Excella STX.

Values	Value Description
string	Specifies text for the endorsement message

PrintFrontData

This key specifies the text to be printed on the front of the check (i.e., franking message). This key is supported by Excella and Excella STX.

Values	Value Description
string	Specifies text for the franking message

PrintFont

This key determines the internal font to be used for the endorsement message. This key is supported by Excella and Excella STX.

Values	Value Description
INTFONT1	Selects internal Font 1 (5x7 bitmap)
INTFONT2	Selects internal Font 2 (7x10 bitmap)

PrintFrontFont

This key determines the internal font to be used for the franking message. This key is supported by Excella and Excella STX.

Values	Value Description
INTFONT1	Selects internal Font 1 (5x7 bitmap)
INTFONT2	Selects internal Font 2 (7x10 bitmap)

PrintStyle

This key determines the style of the selected font for the endorsement message. This key is supported by Excella STX only.

Values	Value Description
BOLD	Print Bold style
NORMAL	Print normal
WIDE	Print Wide style

PrintFrontStyle

This key determines the style of the selected font for the franking message. This key is supported by Excella STX only.

Values	Value Description
BOLD	Print Bold style
NORMAL	Print normal
WIDE	Print Wide style

PrintRate

This key determines the printing rate. This key can be used to control squeezing/condensing of printed characters. Default print rate is 100%. If the value of this key is larger than the default value, the number of characters per inch is increased. The Valid range is 50-250. This key is supported by Excella and Excella STX.

Values	Value Description
numeric string	Printing rate

Virtual

If value of this key is True, Excella will print without ink. It will use third party software to print on the image and not on the check itself.

This key is supported by Excella only, and will only function if the device is set to operate in Express Mode. See the DeviceCapabilities/**ExpressCapable** tag to determine whether the device can be configured to operate in Express Mode. ***See 99875310 EXCELLA MICR CHECK READER AND DUAL-SIDED SCANNER INSTALLATION AND OPERATION MANUAL*** for steps to configure the device to operate in Express Mode.

Values	Value Description
TRUE	The Excella will print its endorsement message on the image of the check, not on the check; no ink is required
FALSE	The Excella will print its endorsement message on the check; ink is required

Also, if **Virtual** is True, it supports these additional keys:

- PrintFontSize
- PrintFrontFontSize
- BackXPosition
- FrontXPositon
- BackYPosition
- FrontYPosition
- YPositionOffset

PrintFontSize

This key determines the font size of the selected font for the endorsement message. This key is supported by the Excella only if the **Virtual** key is set to True. This key is supported by Excella only.

Values	Value Description
numeric string	Font size, default setting is “30”

PrintFrontFontSize

This key determines the font size of the selected font for the franking message. This key is supported by the Excella only if the **Virtual** key is set to True. This key is supported by Excella only.

Values	Value Description
numeric string	Font size, default setting is “30”

BackXPosition

This key determines the X Position to be used for the endorsement message. This key is supported by the Excella only if the **Virtual** key is set to True. This key is supported by Excella only.

Values	Value Description

numeric string	Amount to set X position of endorsement message

FrontXPostion

This key determines the X Position to be used for the franking message. This key is supported by the Excella only if the **Virtual** key is set to True. This key is supported by Excella only.

Values	Value Description
numeric string	Amount to set X position of franking message

BackYPosition

This key determines the Y Position to be used for the endorsement message. This key is supported by the Excella only if the **Virtual** key is set to True. This key is supported by Excella only.

Values	Value Description
numeric string	Amount to set Y position of endorsement message

FrontYPosition

This key determines the Y Position to be used for the franking message. This key is supported by the Excella only if the **Virtual** key is set to True. This key is supported by Excella only.

Values	Value Description
numeric string	Amount to set Y position of franking message

YPositionOffset

This key determines the Y Position offset from the BOTTOM OR TOP for Virtual Endorsement. This key is supported by the Excella only if the **Virtual** key is set to True. This key is supported by Excella only.

Values	Value Description
numeric string	Amount to set Y position offset from the Bottom or Top for Virtual Endorsement

SECTION = ImageOptions

The *ImageOptions* section includes keys that control various options to process the check images. Some of the keys below include the symbol '#' to indicate a variable for the image number. The image number can be 1, 2, 3, or 4 (as specified in the *Number* key). Each image requires its own set of options controlled by the keys *ImageColor#, Resolution#, Compression#, FileType#, and ImageSide#*. For example, for image 2, the key names would be ImageColor2, Resolution2, Compression2, FileType2, and ImageSide2. If image number equals zero, scanning will not take place. All keys in the *ImageOptions* section are ignored. The zero image setting is supported by Excella STX only.

Number

This key determines how many images will be captured for each check that is processed. This key is supported by Excella and Excella STX.

Values	Value Description
numeric value	Specifies number of images to be captured: 1, 2, 3, or 4.
0	If this number equals zero, no image is captured (supported by Excella STX only.)

ImageColor#

This key determines the image rendition for the specified image number. This key is supported by Excella and Excella STX.

Values	Value Description
BW	Black and white (i.e., bitonal)
GRAY8	8-bit grayscale
COL24	24-bit color (supported by Excella STX only)

Resolution#

This key determines the image resolution for the specified image number. This key is supported by Excella and Excella STX.

Values	Value Description
100X100	100x100 DPI (dots per inch)
200X200	200x200 DPI (dots per inch)

Compression#

This key determines the algorithm used to compress the captured images. This key is supported by Excella and Excella STX.

Values	Value Description
GROUP4	Image will be compressed using Group4 compression type
JPEG	Image will be compressed using JPEG compression type
NONE	No compression

FileType#

This key determines the file type for the specified image number. This key is supported by Excella and Excella STX.

Values	Value Description
TIF	Image format type is tiff
JPG	Image format type is jpg
BMP	Image format type is bmp

ImageSide#

This key determines which side of the check will be associated with specified image number. This key is supported by Excella and Excella STX.

Values	Value Description
FRONT	Scan the front of image of the check
BACK	Scan the back image of the check

FilterB

This key can be used to change the sharpness and the file size of a black and white (B/W) image. The Default value of this key is 4. For a sharper image, set the value of this key to a smaller number. For a softer image, set the value of this key to a higher number. Too much sharpening can lead to a noisy image and a large file size. The valid range for this key is 1-6. This key is supported by Excella and Excella STX.

Values	Value Description
numeric string	Filtering B/W images

FilterG

This key can be used to change the sharpness and file size of a gray scale image. The default value of this key is 2. For a sharper image, set the value of this key to a smaller number. For a softer image, set the value of this key to a higher number. The valid range for this key is 1-6. This key is supported by Excella and Excella STX.

Values	Value Description
numeric string	Filtering gray scale images

JPEGQC

This key can be used to set the JPEG compression quality number for color images. The default value for this key is 50. Changing the default value of this key affects the image quality and size of a JPEG file. For higher quality images, set the value of this key to a higher number. The file size also increases when the quality number increases. The valid range for this key is 1-100. This key is supported by Excella and Excella STX.

Values	Value Description
numeric string	Quality number for color image

JPEGQG

This key can be used to set the JPEG compression quality number for grayscale images. The default value for this key is 50. Changing the default value of this key affects the image quality and size of a JPEG file. For higher quality images, set the value of this key to a higher number. The file size also increases when the quality number increases. The valid range for this key is 1-100. This key is supported by Excella and Excella STX.

Values	Value Description
numeric string	Quality number for grayscale images

CalculateSHA1

This key determines if SHA1 calculation is required for captured images. This key is supported by Excella and Excella STX.

Values	Value Description
YES	SHA1 calculation is required
NO	SHA1 calculation is NOT required

ScanLED1, ScanLED2

These keys select the illumination color used to scan the front (ScanLED1) and back (ScanLED2) of the check. The default illumination color is white and it provides the best image quality results. Other colors, such as Red, may be used to drop out a specific target color on specially designed checks. These keys are supported on Excella only (Firmware version 1.80J or above). The values for the available illumination colors are listed in the table below:

Table 5-1. Values for Scan Bar Illumination Colors

Value	Value Description
1	Green
2	Red
3	Green+Red
4	Blue
5	Blue+Green
6	Blue+Red
7	White

Example for setting up ImageOptions key-value pairs to obtain 4 Images

In the following example we are requesting four images:

1. Gray scale with resolution 100x100 for the FRONT of the document
2. Gray scale with resolution 100x100 for the BACK of the document
3. BW for the FRONT of the document
4. BW for the BACK of the document

Table 5-2 lists the current valid combination of Image Color, Resolution, Compression, and File Type.

Table 5-2. Possible Combination Values for Image Options

MagTek Device	ImageColor	Resolution	Compression	FileType
Excella & Excella STX	B/W	200x200	NONE	TIFF
Excella & Excella STX	B/W	200x200	GROUP4	TIFF
Excella & Excella STX	GRAY8	100x100	JPEG	JPG
Excella & Excella STX	GRAY8	200x200	JPEG	JPG
Excella & Excella STX	GRAY8	100x100	NONE	BMP
Excella & Excella STX	GRAY8	200x200	NONE	BMP
Excella STX only	COL24	100x100	JPEG	JPG
Excella STX only	COL24	200x200	JPEG	JPG

Table 5-3. Example for Section ImageOptions – 1 through 4

Number	4
ImageColor1	GRAY8
Resolution1	100x100
Compression1	JPEG
FileType1	JPG
ImageSide1	FRONT
ImageColor2	GRAY8
Resolution2	100x100
Compression2	JPEG
FileType2	JPG
ImageSide2	BACK
ImageColor3	BW
Resolution3	200x200
Compression3	GROUP4
FileType3	TIF
ImageSide3	FRONT
ImageColor4	BW
Resolution4	200x200
Compression4	GROUP4
FileType4	TIF
ImageSide4	BACK

SECTION = MICROptions

The MICR*Options* section includes keys that control various options to get MICR character.

Threshold

This value is the minimum signal value used to detect the start of a MICR symbol. This key is supported by Excella and Excella STX.

Values	Value Description
numeric value	The minimum signal value to detect the start of a MICR symbol. The default value is 15

Quality

This value is used to determine what quality is required for a character to be valid.
The best range is 85-92. This key is supported by Excella and Excella STX.

Values	Value Description
numeric value	Specify what quality is required for a character to be valid. The default value is 90

SECTION 6. KEYS RECEIVED FROM DEVICE

Section 6 describes all the Sections (and their corresponding Key-Value pairs) automatically reported by the device to the application after the requested document has been processed. The device reports information using the following sections: *CommandStatus, DocInfo, ImageInfo,* and *MSRInfo*.

Note

All Sections and Key-Value pairs described in this Section are also available from the device upon query by the application.

The following keys are included in the *CommandStatus* section:

- CheckDS
- ReturnCode
- ReturnMsg
- KVErrCnt
- KVErrCode#
- KVErrVal#

The following keys are included in the *DocInfo* section:

- DocUnits
- DocWidth
- DocHeight
- MICRFont
- MICRRaw
- MICRAcct
- MICRAmt
- MICRAux
- MICRBankNum
- MICRChkType
- MICRCountry
- MICRDecode
- MICREPC
- MICROnUs
- MICROut
- MICRSerNum
- MICRTPC
- MICRTransit
- MICRParseSts0
- MICRParseSts1

The following keys are included in the *ImageInfo* section:

- ImageSize#
- ImageURL#
- ImageSHA1Key#
- Number

The following keys are included in the *MSRInfo* section:

- CardType
- MPData
- MPStatus
- TrackData1
- TrackData2
- TrackData3
- TrackStatus1
- TrackStatus2
- TrackStatus3

SECTION = CommandStatus

This *CommandStatus* section includes keys that report various error conditions after a document has been processed.

CheckDS

This key is a general flag for critical device status, and it can be used to prompt applications to check the device status. This key is supported by Excella and Excella STX.

Values	Value Description
T	True: a change in the device status has been detected
F	False: everything is OK

ReturnCode

This key specifies the return error code reported by the device after each document transaction. This key is supported by Excella and Excella STX.

Values	Value Description
numeric value	For specific return code information refer to tables 6-1 through 6-8

ReturnMsg

This key specifies the return error message associated with the *ReturnCode* key. This key is supported by Excella and Excella STX.

Values	Value Description
String	For specific return message information refer to tables 6-1 through 6-8

KVErrCnt

This key specifies the number of key/value errors (i.e., syntax errors) reported by the device. This key is supported by Excella and Excella STX.

Values	Value Description
numeric value	Count of key/value errors detected (0-9)

KVErrCode#

This key specifies the key/value error code number reported by the device (up to 9 errors can be reported). This key is supported by Excella and Excella STX.

Values	Value Description
numeric value	For specific error code numbers refer to table 6-3

KVErrVal#

This key specifies the invalid key name or value that generated the error (up to 9 errors can be reported). This key is supported by Excella and Excella STX.

Values	Value Description
string	Name of the key or value that generated the error

RETURN CODES AND MESSAGES FROM EXCELLA AND EXCELLA STX

The following tables (Tables 6-1 through 6-8.) list the codes and messages returned by the device using the *ReturnCode* and *ReturnMsg* keys in the *CommandStatus* section.

Table 6-1. Operation Completed

Operation Completed (0-99)	
ReturnCode	**ReturnMsg**
0	"OK"

Table 6-2. Operation

Operation (100-149)	
ReturnCode	**ReturnMsg**
101	"Internal Device Failure"
102	"Unrecognized Operation"
103	"Command Format Error"
104	"Requested Operation Failed"
105	"Undetermined Path Error"

Table 6-3. Data Input

Data Input (150-199)	
ReturnCode	**ReturnMsg**
150	"Data Format Error"
151	"Illegal Value"
152	"Value Out of Range"
153	"String Too Long"
154	"Unrecognized Key"
155	"Unrecognized Section"
156	"Problem parsing XML"

Table 6-4. Path

Path (200-299)	
ReturnCode	**ReturnMsg**
201	“Access Guide Unlatched”
202	“No Doc Present”
203	“Path Not Clear”
204	“Document Jammed”
205	“Multiple doc feed detected”
206	“Manual detect during Autofeed”
207	“Illegal Doc Feed Option”
208	"Cannot find doc at location for this sequence move"
209	"Check Sequence Aborted by Operator"
210	"Doc Feed Path Error"
250	“Timed out waiting for document”

Table 6-5. Printer

Printer (300-349)	
ReturnCode	**ReturnMsg**
301	“Print Failed. Internal Error”
302	“Unable to Print Back. No Cartridge”
303	“Unable to Print Front. No Cartridge”
305	“Print Font Not Supported”
306	“Selected Print is Invalid For Selected Sequence ”
307	“Endorsing Failed. Internal Error”
308	“Franking Failed. Internal Error”

Table 6-6. MICR

MICR (350-399)	
ReturnCode	**ReturnMsg**
351	“MICR Failed. Internal Error”
352	“MICR Not Supported”

Table 6-7. Scan/Image

Scan/Image (400-499)	
ReturnCode	**ReturnMessage**
401	“Scan Failed. Internal Error”
402	“Bad Image”
403	“No Room to Store Image”
404	“Image Store Failed”
405	"Scan Side is Invalid For Selected Sequence"
421	“Scan Calib Black White Delta out of range”
422	“Scan Calib PGA Code out of range”
423	“Scan Calib DAC Code out of range”
424	“Scan Calib Consume and Park Check Failed”
425	“Scan Calib Save Data Failed”
426	“Scan Calib Shade Gain out of range”
427	“Scan Calib VPMAX out of range”
428	“Scan Calib VPAVG out of range”
429	“Scan Calib Get Data Failed, Factory Setting used”
430	“Scan Calib Get Calib Data Failed”
431	“Scan Calib LED Calib failed"
432	"Scan Calib check not found"
433	"Scan Calib Move Check to SB1 failed"
434	“Scan Calib Move Check to SB2 failed"
440	"Sensor Calib failed"

Table 6-8. Miscellaneous

Miscellaneous (500-599)	
Return Code	**Return Message**
501	"No RTC Support"
502	"RTC Battery Low"

SECTION = DocInfo

The *DocInfo* section includes keys that report on various attributes of the check document and information on the MICR data read from the check.

DocUnits

This key specifies the units of measurement used to report on check document dimensions (see the *DocWidth* and *DocHeight* keys below). This key is supported by Excella and Excella STX.

Values	Value Description
ENGLISH	Dimensions in thousands of an inch
METRIC	Dimensions in millimeters

DocWidth

This key specifies the width of the check document based on the scanned image. This key is supported by Excella and Excella STX.

Values	Value Description
numeric value	Width dimension of scanned document

DocHeight

This key specifies the height of the check document based on the scanned image. This key is supported by Excella and Excella STX.

Values	Value Description
numeric value	Height dimension of scanned document

MICRFont

This key specifies the MICR font detected and read from the check. This key is supported by Excella and Excella STX.

Values	Value Description
E13B	E13B font was detected on the check read
CMC7	CMC7 font was detected on the check read

Note
The parsed MICR fields below only apply to U.S. checks with E13B font.
CMC7 Checks are not parsed, and only the raw MICR line is reported.

MICRRaw

This key specifies the unformatted (as-is) MICR data read from the check. This key is supported by Excella and Excella STX.

Values	Value Description
string	Unformatted MICR data

MICRAcct

This key specifies the account number MICR field read from the check. This key is supported by Excella and Excella STX.

Values	Value Description
string	Account number MICR field

MICRAmt

This key specifies the amount MICR field read from the check. This key is supported by Excella and Excella STX.

Values	Value Description
string	Amount MICR field

MICRAux

This key specifies the Auxiliary On-Us MICR field read from the check. This key is supported by Excella and Excella STX.

Values	Value Description
string	Auxiliary On-Us MICR field

MICRBankNum

This key specifies the 4-digit ABA Identifier MICR field read from the check. This key is supported by Excella and Excella STX.

Values	Value Description
string	ABA Institution Identifier (bank number) MICR field

MICRChkType

This key specifies the check type based on the presence on the Auxiliary On-Us MICR field. This key is supported by Excella and Excella STX.

Values	Value Description
PERSONAL	Auxiliary On-Us MICR field NOT present
BUSINESS	Auxiliary On-Us MICR field present

MICRCountry

This key specifies the country of origin based on known MICR line formats. This key is supported by Excella and Excella STX.

Values	Value Description
USA	Check drawn in USA
CANADIAN	Check drawn in Canada
MEXICAN	Check drawn in Mexico

MICRDecode

This key indicates whether a MICR decode/read error was detected. This key is supported by Excella and Excella STX.

Values	Value Description
NONE	No MICR data detected
OK	MICR decode/read was successful
ERROR	Error detected in MICR decode/read operation

MICREPC

This key specifies the EPC MICR field read from the check. This key is supported by Excella and Excella STX.

Values	Value Description
string	EPC MICR field

MICROnUs

This key specifies the On-Us MICR field read from the check. This key is supported by Excella and Excella STX.

Values	Value Description
string	On-Us MICR field

MICROut

This key specifies the formatted MICR output data read from the check as defined by the *MICRFmtCode* key in Section=*ProcessOptions*. This key is supported by Excella and Excella STX.

Values	Value Description
string	Formatted MICR output data

MICRSerNum

This key specifies the sequence number MICR field read from the check. This key is supported by Excella and Excella STX.

Values	Value Description
string	Sequence number (or check number) MICR field

MICRTPC

This key specifies the TPC MICR field read from the check. This key is supported by Excella and Excella STX.

Values	Value Description
string	TPC MICR field

MICRTransit

This key specifies the 9-digit Transit MICR field read from the check. This key is supported by Excella and Excella STX.

Values	Value Description
string	Transit MICR field

MICRParseSts0

This key specifies a 4-digit status/error code reported by the device after parsing the MICR fields read from the check. This key is supported by Excella and Excella STX.

Values	Value Description
numeric value	For specific status/error code information refer to table 6-9

Table 6-9. MICRPARSESTS0

Digit	
1st Digit	0 – OK MICR
	1 – Low MICR
	2 – No MICR
2nd Digit	0 – Standard Check
	1 – Business Check
	2 – Mexican Check
	3 – Canadian Check
3rd Digit	0 – No Status
	1 – Amount Present
	2 – Short Account
	3 – Short Account + Amount Present
	4 – No Check#
	5 – No Check# + Amount Present
	6 – No Check# + Short Account
	7 – No Check# + Short Account + Amount Present
4th Digit	0 – No Errors
	1 – Chk#
	2 – Account
	3 – Account + Chk#
	4 – Transit
	5 – Transit + Chk#
	6 – Transit + Account
	7 – Transit + Account + Chk#

MICRParseSts1

This key specifies a 2-digit status/error code reported by the device after parsing the MICR fields read from the check. This key is supported by Excella and Excella STX.

Values	Value Description
numeric value	For specific status/error code information refer to table 6-10

Table 6-10. MICRPARSESTS1

PRIORITY	MICRParseSts1	TYPE	DESCRIPTION
10	01	Error	No MICR data: no transit and no account found
9	09	Status	Mexican check
8	08	Status	Canadian check
7	05	Error	No transit, bad character, bad length, bad check digit
6	07	Error	No account, bad character
5	04	Error	Bad character in check number
5	04	Status	No check number
4	12	Status	Short Account (maybe caused by mis-parsed check#)
3	03	Status	Low MICR signal, good read
2	10	Status	Business check
1	11	Status	Amount field present
0	00	Status	No error, check OK

Notes:

- The LED indicator will turn red on all error conditions.
- The absence of a check number is not considered an error.
- If a multiple error occurs, the error or status code with the highest priority is reported.
- All unreadable MICR characters are transmitted as an “?” ASCII character (hex 3F), except for Format 00xx.

SECTION = ImageInfo

The *ImageInfo* section includes keys that report on various attributes of the scanned images. Some of the keys below include the symbol '#' to indicate a variable for the image number. The image number can be 1, 2, 3, or 4 (up to 4 images as specified in the *Number* key below). The device reports image information using the keys *ImageSize#*, *ImageURL#*, and *ImageSHA1#*. For example, for image 2, the key names would be ImageSize2, ImageURL2, and ImageSHA1Key2.

The following tags are included in every image file (TIFF tags for black and white images; EXIF tags for JPEG files):

Tag #	Tag Name	Content
270	IMAGE DESCRIPTION	Contains raw MICR line
271	MAKE	"MagTek, Inc."
272	MODEL	"Excella (unit's serial number)"
305	SOFTWARE	[firmware version]
315	AUTHOR	Magtek,FB=#,FG=#,QC=#,QG=#. The # symbol refers to the settings for FilterB, FilterG, JPEGQC, JPEGQG (see the *ImageOptions* section in Section 5).

ImageSize#

This key specifies the size (in bytes) for the specified Image number. The number of scanned images and their availability is determined by the *Number* key in Section=*ImageOptions*. This key is supported by Excella and Excella STX.

Values	Value Description
numeric value	Number of bytes

ImageURL#

This key specifies the URL string needed to request the specified Image number using the **MTMICRGetImage** API function. This key is supported by Excella and Excella STX.

Values	Value Description
string	URL string to indicate internal file path for the Image

ImageSHA1Key#

This key contains the SHA1 key calculation for the specified Image number (if requested and available). This key is supported by Excella and Excella STX.

Values	Value Description
string	SHA1 Key string for the image

Number

This key specifies the number of scanned images ready and available from the device. This key is supported by Excella and Excella STX.

Values	Value Description
numeric value	Number of scanned images

SECTION = MSRInfo

The *MSRInfo* section includes keys that report on the information captured from the magnetic stripe card. This section is supported by Excella STX only.

CardType

This key specifies the standard encoding format detected on the magnetic stripe card. This key is supported by Excella STX only.

Values	Value Description
NONE	Unknown Card Type
ISO	ISO format
CADL	Old California Drive License format
AAMVA	American Association Motor Vehicle Administrators format

MPData

This key specifies the MagnePrint data read from the magnetic stripe card. This key is supported by Excella STX only.

Values	Value Description
string	Please contact MagTek for the use of MagnePrint data

MPStatus

This key specifies the MagnePrint status reported by the device. This key is supported by Excella STX only.

Values	Value Description
string	Please contact MagTek for the use of MagnePrint data

TrackData1

This key specifies the Track 1 data read from the magnetic stripe card. This key is supported by Excella STX only.

Values	Value Description
string	Track 1 data on magnetic stripe card

TrackData2

This key specifies the Track 2 data read from the magnetic stripe card. This key is supported by Excella STX only.

Values	Value Description
string	Track 2 data on magnetic stripe card

TrackData3

This key specifies the Track 3 data read from the magnetic stripe card. This key is supported by Excella STX only.

Values	Value Description
string	Track 3 data on magnetic stripe card

TrackStatus1

This key indicates whether a decode/read error was detected while reading Track 1. This key is supported by Excella STX only.

Values	Value Description
NONE	No status
OK	No read error detected
ERROR	Read error detected

TrackStatus2

This key indicates whether a decode/read error was detected while reading Track 2. This key is supported by Excella STX only.

Values	Value Description
NONE	No status
OK	No read error detected
ERROR	Read error detected

TrackStatus3

This key indicates whether a decode/read error was detected while reading Track 3. This key is supported by Excella STX only.

Values	Value Description
NONE	No status
OK	No read error detected
ERROR	Read error detected

EncryptedTrackData1

This key specifies the Encrypted Track 1 data read from the magnetic stripe card. The Encrypted Track data is only in MagneSafe Reader Security Level 3.

Values	Value Description
string	Encrypted Track 1 data on magnetic stripe card

EncryptedTrackData2

This key specifies the Encrypted Track 2 data read from the magnetic stripe card. The Encrypted Track data is only in MagneSafe Reader Security Level 3.

Values	Value Description
string	Encrypted Track 2 data on magnetic stripe card

EncryptedTrackData3

This key specifies the Encrypted Track 3 data read from the magnetic stripe card. The Encrypted Track data is only in MagneSafe Reader Security Level 3.

Values	Value Description
string	Encrypted Track 3 data on magnetic stripe card

DeviceSerialNumber

This key specifies the Device Serial Number from MagneSafe Reader.

Values	Value Description
string	Track 1 data on magnetic stripe card

EncryptedSessionID

This key specifies the Track Encrypted Session ID from MagneSafe Reader.

Values	Value Description
string	Track 2 data on magnetic stripe card

DUKPTserialnumber

This key specifies the DUKPT serial number from MagneSafe Reader.

Values	Value Description
string	Track 3 data on magnetic stripe card

SECTION 7. OTHER KEYS AVAILABLE FROM DEVICE

Section 7 describes other Sections (and their corresponding key-value pairs) available from the device upon query by the application. The device can report these additional sections: *DeviceUsage, DeviceCapabilities* and *DeviceStatus,*.

The following keys are included in the *DeviceUsage* section:

- ChecksRead
- DocsRead
- CardsRead
- CardsScanned
- HoursOp
- HoursOn
- InkUsed
- FrontInkUsed

The following keys are included in the *DeviceCapabilities* section:

- AutoFeed
- IDScan
- MagStripe
- MagnePrint
- Endorse
- Firmware
- Image
- MICR
- UnitSerialNumber
- Stamp
- Color
- MachineType
- USBDriver
- ExpressCapable

The following keys are included in the *DeviceStatus* section:

- State
- ManualFeeder
- AutoFeeder
- IDFeeder
- Lamp1
- Lamp2
- Ink
- FrontInk
- Path
- Printer
- FrontPrinter
- RTCBattery
- ScanCalibStatus
- SnsrCalibStatus
- AccessGuide
- ExpressEnabled
- USBSpeed
- StartTimeout
- RawSensors

SECTION = DeviceUsage

The *DeviceUsage* section includes keys that report general operational statistics of the device.

ChecksRead

This key specifies the number of checks read throughout the life span of the device. A check is recorded as read when the MICR line on the check is read, or the image of the check is captured, or both. If there is no RTC chip available in the device, the value in this key will be the same as in the *DocsRead* key. This key is supported by Excella and Excella STX.

Values	Value Description
numeric value	Number of checks read throughout life span

DocsRead

This key specifies the number of checks read since Maintenance reset. A check is recorded as read when the MICR line on the check is read, or the image of the check is captured, or both. This key is supported by Excella and Excella STX.

Values	Value Description
numeric value	Number of checks read since Maintenance reset

CardsRead

This key specifies the number of cards read throughout the life span of the device. A card is recorded as read when it is detected by card reader. This key is supported by Excella STX only.

Values	Value Description
numeric value	Number of cards read throughout life span

CardsScanned

This key specifies the number of cards scanned throughout the life span of the device. A card is recorded as scanned when the card image is captured. This key is supported by Excella STX only.

Values	Value Description
numeric value	Number of cards scanned throughout life span

HoursOp

This key specifies the number of operating hours throughout the life span of the device. If there is no RTC chip available in the device, the value in this key will be the same as in the *HoursOn* key. This key is supported by Excella and Excella STX.

Values	Value Description
numeric value	Number of operating hours throughout life span

HoursOn

This key specifies the number of operating hours since Maintenance reset. This key is supported by Excella and Excella STX.

Values	Value Description
numeric value	Number of operating hours since Maintenance reset

InkUsed

This key specifies the number of ink drops consumed since the back ink cartridge was installed. This key is supported by Excella and Excella STX.

Values	Value Description
numeric value	Number of ink drops since consumed

FrontInkUsed

This key specifies the number of ink drops consumed since the front ink cartridge was installed. This key is supported by Excella STX only.

Values	Value Description
numeric value	Number of ink drops since consumed

SECTION = DeviceCapabilities

The *DeviceCapabilities* section includes keys that report on the general capabilities of the device.

AutoFeed

This key indicates whether an auto-feed input tray is available on the device. This key is supported by Excella and Excella STX.

Values	Value Description
T	True: auto-feed input tray is available
F	False: auto-feed input tray is NOT available

IDScan

This key indicates whether the device is capable ID scanning ID cards. This key is supported by Excella and Excella STX.

Values	Value Description
T	True: ID card scanning is available
F	False: ID card scanning is NOT available

MagStripe

This key indicates whether an integrated MSR (Magnetic Stripe Reader) is available on the device. This key is supported by Excella STX only.

Values	Value Description
T	True: MSR is available
F	False: MSR is NOT available

MagnePrint

This key indicates whether an integrated MagnePrint reader is available on the device. This key is supported by Excella STX only.

Values	Value Description
T	True: MagnePrint is available
F	False: MagnePrint is NOT available

Endorse

This key indicates what print capabilities are available on the device. This key is supported by Excella and Excella STX.

Values	Value Description
BOTH	Device can print on the front and the back of the document
FRONT	Device can only print on the front of the document
BACK	Device can only print on the back of the document
NONE	Device does not have printing capabilities

Firmware

This key specifies the version of the firmware installed in the device. This key is supported by Excella and Excella STX.

Values	Value Description
string	Firmware version

Image

This key indicates what check scanning capabilities are available on the device. This key is supported by Excella and Excella STX.

Values	Value Description
BOTH	The device scans the front and back images of the check
FRONT	The device can only scan the front image of the check
BACK	The device can only scan the back image of the check
NONE	The device cannot scan any images

MICR

This key indicates whether the device is capable if reading MICR data. This key is supported by Excella and Excella STX.

Values	Value Description
T	True: device can read MICR data
F	False: device can NOT read MICR data

UnitSerialNumber

This key specifies the serial number of the device. This key is supported by Excella and Excella STX.

Values	Value Description
string	Device serial number

Stamp

This key indicates whether the device has stamping capability. This key is supported by Excella and Excella STX.

Values	Value Description
BOTH	The device has stamping capability on the front and the back printers
FRONT	The device has stamping capability on the front printer only
BACK	The device stamping capability on the back printer only
NONE	The device does not have stamping capability

Color

This key indicates whether the device is capable of scanning and producing color images. This key is supported by Excella and Excella STX.

Values	Value Description
BOTH	The device can produce color images of the front and back of the check
FRONT	The device can only produce color image of the front of the check
BACK	The device can only produce color image of the back of the check
NONE	The device cannot produce color image

MachineType

This key indicates the type of Excella device connected. This key is supported by Excella and Excella STX.

Values	Value Description
EXCELLA	The device connected is Excella
EXCELLASTX	The device connected is Excella STX

USBDriver

This key indicates the USB driver configuration for the Excella device. This key is supported by Excella and Excella STX USB.

Values	Value Description
MAGUSB	Excella is configured for MagTek USB drivers
RNDIS	Excella is configured for RNDIS USB drivers

ExpressCapable

This key specifies if Excella can run “express mode”. This key is supported by Excella only.

Values	Value Description
TRUE	The USB speed is USB2.0 - the device can run “express mode”
FALSE	The USB speed is USB1.1 - the device can NOT run “express mode”

SECTION = DeviceStatus

The *DeviceStatus* section includes keys that report general operational status of the device.

State

This key specifies the general status of the device. This key is supported by Excella and Excella STX.

Values	Value Description
ONLINE	Device is connected and online; device is ready to process check
ERROR	Device is not ready to process check

ManualFeeder

This key indicates whether a check is present in the manual-feed (i.e., single-feed) input tray. This key is supported by Excella and Excella STX.

Values	Value Description
NOTSUP	The connected device does not have a manual-feed input tray
EMPTY	No check present in the manual-feed input tray
DOCPRESENT	Check present in the manual-feed input tray

AutoFeeder

This key indicates whether a check is present in the auto-feed input tray. This key is supported by Excella and Excella STX.

Values	Value Description
NOTSUP	The connected device does not have an auto-feed input tray
EMPTY	No check present in the auto-feed input tray
DOCPRESENT	Check present in the auto-feed input tray

IDFeeder

This key indicates whether an ID card is present in the card input slot for scanning. This key is supported by Excella STX only.

Values	Value Description
NOTSUP	The connected device does not have an card input slot
EMPTY	No card present in the card input slot
DOCPRESENT	Card present in the card input slot

Lamp1

This key specifies status for the front image scan bar. This key is supported by Excella and Excella STX.

Values	Value Description
OK	Scan bar is ready
NOTSUP	Scan bar is not present
DIM	Light source is low
FAILED	Scan bar is not working

Lamp2

This key specifies status for the back image scan bar. This key is supported by Excella and Excella STX.

Values	Value Description
OK	Scan bar is ready
NOTSUP	Scan bar is not present
DIM	Light source is low
FAILED	Scan bar is not working

Ink

This key specifies status for the back ink cartridge used for endorsing. This key is supported by Excella and Excella STX.

Values	Value Description
EMPTY	The back ink cartridge is empty
NOTSUP	The back ink cartridge is not present
OK	The back ink cartridge is ready
LOW	The back ink cartridge has low ink

FrontInk

This key specifies status for the front ink cartridge used for franking. This key is supported by Excella STX only.

Values	Value Description
EMPTY	The front ink cartridge is empty
NOTSUP	The front ink cartridge is not present
OK	The front ink cartridge is ready
LOW	The front ink cartridge has low ink

Path

This key specifies status of the check path. This key is supported by Excella and Excella STX.

Values	Value Description
OK	Check path is clear
ERROR	Check path is not clear

Printer

This key indicates whether the back ink cartridge is installed. This key is supported by Excella and Excella STX.

Values	Value Description
NONE	No print cartridge installed
PRESENT	Print cartridge detected

FrontPrinter

This key indicates whether the front ink cartridge is installed. This key is supported by Excella STX only.

Values	Value Description
NONE	No print cartridge installed
PRESENT	Print cartridge detected

RTCBattery

This key specifies the status of the battery as reported by the RTC (Real Time Clock) chip in the device. This key is supported by Excella and Excella STX.

Values	Value Description
OK	RTC battery is OK
LOW	RTC battery is low or not installed
NOTSUP	RTC chip is not present in the device

ScanCalibStatus

This key specifies the status of the most recent scanner calibration performed on the device. It is highly recommended to keep the factory calibration. Any calibration performed by the user will take precedence over the factory calibration. If the user calibration is invalid, the device uses factory calibration. This key is supported by Excella and Excella STX.

Values	Value Description
NONE	The device is not calibrated
FACTORY	The device is using factory calibration
USER	The device is using the user calibration
ERROR	Neither the factory nor user calibration is valid

SnsrCalibStatus

This key specifies the status of the most recent sensor calibration performed on the device. This key is supported by Excella STX only.

Values	Value Description
NONE	The sensor is not calibrated
FACTORY	The sensor is using factory calibration data
ERROR	The calibration data is invalid

AccessGuide

This key specifies the status of the top cover on the Excella STX device or the status of the access guide on the Excella device. This key is supported by Excella and Excella STX.

Values	Value Description
UNKNOWN	Status is undetermined
LATCHED	The top cover is latched and installed
UNLATCHED	The top cover is unlatched or removed

ExpressEnabled

This key specifies if Excella is set as "express mode". This key is supported by Excella only.

Values	Value Description
TRUE	The device is set as express mode
FALSE	The device is set as normal mode*

*If the USB speed is USB1.1, the device is ALWAYS set as normal mode. This value is ALWAYS set as "FALSE".

USBSpeed

This key specifies the protocol of USB connection. This key is supported by Excella only.

Values	Value Description
HI	The current USB connection is USB2.0
LOW	The current USB connection is USB1.1

StartTimeout

This key specifies the firmware Timeout on check entry. This key is supported by Excella and Excella STX.

Values	Value Description
numeric value	Time, in milliseconds

RawSensors

This key contains a decimal number that can be masked to determine the status of a particular sensor or the status of a print cartridge or the status of the cover switch. When the value of a masked bit equals zero, this typically indicates a normal or default operational state. For example, depending on the masked bit, the zero value may indicate that the cover is detected, the ink cartridge is installed, or the sensor is unblocked. This key is supported by Excella and Excella STX.

Masked Values	Value Description for Excella STX
0x00001	0 = sensor at check entry 1 unblocked
0x00002	0 = sensor at check entry 2 unblocked
0x00004	0 = sensor at entry pinch unblocked
0x00008	0 = sensor at MICR head unblocked
0x00010	0 = sensor just past back printer unblocked
0x00020	0 = sensor just before front printer unblocked
0x00040	0 = sensor at exit pinch unblocked
0x00080	0 = sensor at exit 2 unblocked
0x00100	0 = sensor at exit 1 unblocked
0x00400	0 = back ink cartridge installed
0x00800	0 = front ink cartridge installed
0x10000	0 = cover detected

Masked Values	Value Description for Excella
0x0001	0 = sensor at manual feeder unblocked
0x0002	0 = sensor at MICR head unblocked
0x0004	0 = sensor just past back printer unblocked
0x0008	0 = sensor at exit pinch unblocked
0x0020	0 = sensor at auto feeder unblocked
0x0040	0 = access guide latched
0x0080	0 = sensor at operator button unblocked
0x0100	0 = back ink cartridge installed

SECTION 8. EXAMPLES OF KEY-VALUE PAIRS

This section contains examples of key-value pairs send to the Excella device and the key-value pairs returned from Excella device when requesting four images without endorsement. The corresponding key-value pairs for these examples are also provided in XML format. The examples include key-value pairs for querying the Excella device. These are key-value pairs in the *DeviceStatus*, *DeviceCapabilities*, and *DeviceUsage*.

EXAMPLE 1: REQUESTING TWO IMAGES WITH ENDORSEMENT AND FRANKING

Key-Value Pairs Sent by Host Application to Excella Device

Section: **Application**

Key	**Value**
Transfer	HTTP
DocUnits	ENGLISH

Section: **ProcessOptions**

Key	**Value**
ReadMICR	E13B
Endorse	BOTH
DocFeed	MANUAL
RespondEarly	YES

Section **ImageOptions**

Key	**Value**
Number	2
ImageColor1	COL24
Resolution1	100x100
Compression1	JPEG
FileType1	JPG
ImageSide1	FRONT
ImageColor2	COL24
Resolution2	100x100
Compression2	JPEG
FileType2	JPG
ImageSide2	BACK

Section: **Endorser**

Key	**Value**
PrintFont	INTFONT2
PrintData	VOID-VOID-VOID
PrintStyle	NORMAL
PrintFrontFont	INTFONT2
PrintFrontStyle	NORMAL
PrintFontData	VOID-VOID-VOID

Key-Value Pairs Sent by STXDemo Application to Excella Device in XML Format

```
<DeviceSettings>
    <ImageOptions>
        <Number>2</Number>
        <ImageSide1>FRONT</ImageSide1>
        <ImageSide2>BACK</ImageSide2>
        <ImageColor1>COL24</ImageColor1>
        <Resolution1>100x100</Resolution1>
        <Compression1>JPEG</Compression1>
        <FileType1>JPG</FileType1>
        <ImageColor2>COL24</ImageColor2>
        <Resolution2>100x100</Resolution2>
        <Compression2>JPEG</Compression2>
        <FileType2>JPG</FileType2>
    </ImageOptions>
    <Application>
        <Transfer>HTTP</Transfer>
        <DocUnits>ENGLISH</DocUnits>
    </Application>
−   <Endorser>
        <PrintStyle>NORMAL</PrintStyle>
        <PrintFrontStyle>NORMAL</PrintFrontStyle>
        <PrintFont>INTFONT2</PrintFont>
        <PrintFrontFont>INTFONT2</PrintFrontFont>
        <PrintData>VOID-VOID-VOID</PrintData>
        <PrintFrontData> VOID-VOID-VOID </PrintFrontData>
    </Endorser>
−   <ProcessOptions>
        <ReadMICR>E13B</ReadMICR>
        <Endorse>BOTH</Endorse>
        <DocFeed>MANUAL</DocFeed>
        <RespondEarly>YES</RespondEarly>
    </ProcessOptions>
</DeviceSettings>
```

Key-Value Pairs Returning from Excella Device

Section **CommandStatus**

Key	Value
BadData	NONE
CheckDS	F
ReturnCode	0
ReturnMsg	OK

Section **ImageInfo**

Key	Value
ImageSize1	40481
ImageSize2	21587
ImageURL1	/chkimg/FRONTCOL24Image1_0023.JPG
ImageURL2	/chkimg/BACKCOL24Image2_0023.JPG
Number	2

Section **DocInfo**

Key	Value
DocHeight	2990
DocUnits	ENGLISH
DocWidth	8000
MICRAcct	123456789
MICRAmt	
MICRAux	007751
MICRBankNum	0021
MICRChkType	BUSINESS
MICRCountry	USA
MICRDecode	OK
MICREPC	
MICRFont	E13B
MICROnUs	123456789U 11
MICROut	C/122000218/123456789/007751/0100
MICRParseSts0	0100
MICRParseSts1	11
MICRRaw	U007751U T122000218T123456789U 11
MICRSerNum	007751
MICRTPC	11
MICRTransit	122000218

Section **DeviceStatus**

Key	Value
AccessGuide	LATCHED
AutoFeeder	NOTSUP
ExpressEnabled	TRUE
FrontInk	OK
FrontPrinter	PRESENT
IDFeeder	EMPTY
Ink	OK
Lamp1	OK
Lamp2	OK
ManualFeeder	EMPTY
Path	OK
Printer	PRESENT
RTC Battery	OK
RawSensors	384
ScanCalibStatus	FACTORY
SnsrCalibStatus	FACTORY
StartTimeout	4000
State	ONLINE
USBSpeed	HI

Key-Value Pairs Returning From Excella Device In XML Format

```
<DeviceInformation>
        <CommandStatus>
                <BadData>OK</BadData>
                <CheckDS>F</CheckDS>
                <ReturnCode>0</ReturnCode>
                <ReturnMsg>OK</ReturnMsg>
        </CommandStatus>
```

<DeviceStatus>
<AccessGuide>**LATCHED**</AccessGuide>
<AutoFeeder>**NOTSUP**</AutoFeeder>
<ExpressEnabled>**TRUE**</ExpressEnabled>
<FrontInk>**OK**</FrontInk>
<FrontPrinter>**PRESENT**</FrontPrinter>
<IDFeeder>**EMPTY**</IDFeeder>
<Ink>**OK**</Ink>
<Lamp1>**OK**</Lamp1>
<Lamp2>**OK**</Lamp2>
<ManualFeeder>**EMPTY**</ManualFeeder>
<Path>**OK**</Path>
<Printer>**PRESENT**</Printer>
<RTCBattery>**OK**</RTCBattery>
<RawSensors>**384**</RawSensors>
<ScanCalibStatus>**FACTORY**</ScanCalibStatus>
<SnsrCalibStatus>**FACTORY**</SnsrCalibStatus>
<StartTimeout>**4000**</StartTimeout>
<State>**ONLINE**</State>
<USBSpeed>**HI**</USBSpeed>
</DeviceStatus>
<DocInfo>
<DocHeight>**2990**</DocHeight>
<DocUnits>**ENGLISH**</DocUnits>
<DocWidth>**8000**</DocWidth>
<MICRAcct>**123456789**</MICRAcct>
<MICRAmt></MICRAmt>
<MICRAux>**007751**</MICRAux>
<MICRBankNum>**0021**</MICRBankNum>
<MICRChkType>**BUSINESS**</MICRChkType>
<MICRCountry>**USA**</MICRCountry>
<MICRDecode>**OK**</MICRDecode>
<MICREPC></MICREPC>
<MICRFont>**E13B**</MICRFont>
<MICROnUs>**123456789U 11**</MICROnUs>
<MICROut>**C/122000218/123456789/007751/0100**</MICROut>
<MICRParseSts0>**0100**</MICRParseSts0>
<MICRParseSts1>**11**</MICRParseSts1>
<MICRRaw>**U007751U T122000218T123456789U 11**</MICRRaw>
<MICRSerNum>**007751**</MICRSerNum>
<MICRTPC>**11**</MICRTPC>
<MICRTransit>**122000218**</MICRTransit>
</DocInfo>
<ImageInfo>
<ImageSize1>**40481**</ImageSize1>
<ImageSize2>**21587**</ImageSize2>
<ImageURL1>**/chkimg/FRONTCOL24Image1_0023.JPG**</ImageURL1>
<ImageURL2>**/chkimg/BACKCOL24Image2_0023.JPG**</ImageURL2>
<Number>**2**</Number>
</ImageInfo>
</DeviceInformation>

EXAMPLE 2: DEVICE STATUS REPORTED BY EXCELLA DEVICE

Section = **DeviceStatus**

Key	Value
AccessGuide	LATCHED
AutoFeeder	NOTSUP
ExpressEnabled	TRUE
FrontInk	OK
FrontPrinter	PRESENT
IDFeeder	EMPTY
Ink	OK
Lamp1	OK
Lamp2	OK
ManualFeeder	EMPTY
Path	OK
Printer	PRESENT
RTCBattery	OK
RawSensors	384
ScanCalibStatus	FACTORY
SnsrCalibStatus	FACTORY
StartTimeout	4000
State	ONLINE
USBSpeed	HI

DEVICE STATUS REPORTED BY EXCELLA DEVICE IN XML FORMAT

```
<DeviceInformation>
  - <DeviceStatus>
        <AccessGuide>LATCHED</AccessGuide>
        <AutoFeeder>NOTSUP</AutoFeeder>
        <ExpressEnabled>TRUE</ ExpressEnabled >
        <FrontInk>OK</FrontInk>
        <FrontPrinter>PRESENT</FrontPrinter>
        <IDFeeder>EMPTY</IDFeeder>
        <Ink>OK</Ink>
        <Lamp1>OK</Lamp1>
        <Lamp2>OK</Lamp2>
        <ManualFeeder>EMPTY</ManualFeeder>
        <Path>OK</Path>
        <Printer> PRESENT </Printer>
        <RTCBattery>OK</RTCBattery>
        <RawSensors>384</RawSensors>
        <ScanCalibStatus>FACTORY</ScanCalibStatus>
        <SnsrCalibStatus>FACTORY</SnsrCalibStatus>
        <StartTimeout>4000</StartTimeout>
        <State>ONLINE</State>
        <USBSpeed>HI</USBSpeed>
    </DeviceStatus>
</DeviceInformation>
```

EXAMPLE 3: DEVICE CAPABILITIES REPORTED BY EXCELLA DEVICE

Section=**DeviceCapabilities**

Key	Value
AutoFeed	F

Color	BOTH
Endorse	BOTH
ExpressCapable	TRUE
Firmware	MS1.01.25K
IDScan	T
Image	BOTH
MICR	T
MachineType	ExcellaSTX
MagStripe	T
MagnePrint	T
Stamp	NONE
USBDriver	RNDIS
UnitSerialNumber	NONE

DEVICE CAPABILITIES REPORTED BY EXCELLA DEVICE IN XML FORMAT

```
- <DeviceInformation>
        - <DeviceCapabilities>
                <AutoFeed>F</AutoFeed>
                <Color>BOTH</Color>
                <Endorse>BOTH</Endorse>
                <ExpressCapable>TRUE</ ExpressCapable>
                <Firmware>MS1.01.25K</Firmware>
                <IDScan>T</IDScan>
                <Image>BOTH</Image>
                <MICR>T</MICR>
                <MachineType>ExcellaSTX</MachineType >
                <MagStripe>T</MagStripe>
                <MagnePrint>T</MagnePrint>
                <Stamp>NONE</Stamp>
                <USBDriver>RNDIS</USBDriver>
                <UnitSerialNumber>NONE</UnitSerialNumber>
          </DeviceCapabilities>
</DeviceInformation>
```

EXAMPLE 4: DEVICE USAGE REPORTED BY EXCELLA DEVICE

Section=**DeviceUsage**

Key	**Value**
CardsRead	15
CardsScanned	4
ChecksRead	39
DocsRead	47
HoursOn	162
HoursOp	2382
InkUsed	12301
FrontInkUsed	10894

DEVICE USAGE REPORTED BY EXCELLA DEVICE IN XML FORMAT

```
- <DeviceInformation>
-       <DeviceUsage>
              <CardsRead>15</CardsRead>
              <CardsScanned>4</CardsScanned>
              <ChecksRead>39</ChecksRead>
              <DocsRead>47</DocsRead>
              <FrontInkUsed>10894</FrontInkUsed>
              <HoursOn>162</HoursOn>
              <HoursOp>2382</HoursOp>
              <InkUsed>12301</InkUsed>
       </DeviceUsage>
</DeviceInformation>
```

APPENDIX A. FORMAT LIST

For check reading, the Excella device provides the flexibility to format the MICR fields and build a specific output string that will be transmitted to the Host. These output strings are referred to as Formats. The Excella device has a built-in list of Formats (described below) from which the user may select one to become the active Format every time a check is read. The Formats may be selected using the Key- Pair Values described in Section 8.

Each Format is assigned a 4-digit number. The first two digits indicate the Format number, and the last two digits are specific parameters used for various functions by each Format. For example, in Format "0415", the "04" refers to Format number 4 and the 15 refers the maximum number of characters allowed for the account field.

Note

The formats listed in this section apply only to U.S. and Canadian checks. The MICR line on checks from other countries will not be broken or parsed as described in these formats.

A complete description for each Format follows.

Fmt 00xx: Raw Data Format - sends the entire MICR line - where:

```
xx          - specify what symbol set to use.  Choose from the table
Add xx + 16 - change multiple spaces to one space
Add xx + 32 - Remove all spaces

Examples:

   MICR LINE:  T122000218T  1234 5678 9U   1321
      FC0001 - t122000218t  1234 5678 9o   1321
(+16) FC0017 - t122000218t 1234 5678 9o 1321
(+32) FC0033 - t122000218t123456789o1321.
```

xx	Transit	On-Us	Amount	Dash	Error
00	T	U	$	-	?
01	t	o	a	d	?
02	T	O	A	D	?
03	T	U	$	-	*
04	T	U	$	0	?
05	T	U	$	0	*
06	t	o	a	0	?
07	T	U	$	none	?

Fmt 01xx: Parsed Text Format

```
FC0100      - Parsed text with dashes
FC0101      - Parsed text, replace dashes with "d"
Field Labels - TR-transit, AC-account #, CK-check #, AM-amount, TP-tpc,
               EP-epc
```

```
            Example:    - PTTR444455556;AC 999-222-3;CK11045
```

Fmt 02xx: Parsed Text Format with Error Labels

```
            FC0200      - Parsed text with dashes
            FC0201      - Parsed text, replace dashes with "d"
            Error Labels - PE-parsed error, NE-no error, TR-transit error,
                           CK-chk # error, TC-transit check digit error,
                           AM-amount error, OU-on us/account# error, TP-tpc error
            Examples:   - PTTR444455556;AC999-222-3;CK11045/PENE
                        - PTTR111?11111;AC123456/PETR ("?" = unreadable character)
```

Fmt 03xx: **[acct #]**

- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - keep spaces and dashes

Fmt 04xx: **[acct #]**

- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes

Fmt 05xx: **[acct #]**

- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - replace spaces and dashes with zeros

Fmt 06xx: **[acct #]**

- **[acct #]:** - always xx characters, zero filled;
    when xx=00 all characters are sent
  - replace spaces and dashes with zeros

Fmt 07xx: **[acct #]**

- **[acct #]:** - always xx characters, zero filled;
    when xx=00 all characters are sent
  - remove spaces and dashes

Fmt 08xx: **[transit] [acct #]**

- **[transit]:** - all characters in the field
  - keep dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes

Fmt 09xx: [transit] [acct #]

- **[transit]:** - all characters in the field
  - keep dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - replace spaces and dashes with zeros

Fmt 10xx: [transit] [acct #]

- **[transit]:** - all characters in the field
  - keep dashes
- **[acct #]:** - always xx characters, zero filled; when xx=00 all characters are sent
  - replace spaces and dashes with zeros

Fmt 11xx: [transit] 'T' [acct #] 'A' [check #]

- **[transit]:** - all characters in the field
  - keep dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - all characters in the field

Fmt 12xx: [transit] 'T' [acct #] 'A' [check #]

- **[transit]:** - all characters in the field
  - keep dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - always 6 characters, zero filled

Fmt 13xx: [transit] 'T' [acct #] 'A' [check #] '000'

- **[transit]:** - all characters in the field
  - keep dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - always 6 characters, zero filled

Fmt 14xx: **[transit] [acct #] [check #]**

- **[transit]:** - all characters in the field
  - keep dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - always 6 characters, zero filled

Fmt 15xx: **[bank #] [acct #]**

- **[bank #]:** - all characters in the field
  - keep spaces and dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes

Fmt 16xx: **[bank #] [chk dgt] [acct #]**

- **[bank #]:** - all characters in the field
  - keep spaces and dashes
- **[chk dgt]:** - all characters (one character long)
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes

Fmt 17xx: **[transit] [acct #]**

- **[transit]:** - all characters in the field
  - keep dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - keep spaces and dashes

Fmt 18xx: **[acct #] "/" [check #]**

- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - keep spaces and dashes
- **[check #]:** - all characters in the field

Fmt 19xx: **[transit] [acct #] [check #]**

- **[transit]:** - all characters in the field
  - keep dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - replace spaces and dashes with zeros
- **[check #]:** - all characters in the field

Fmt 20xx: **[transit] [acct #] <CR> [check #]**

- **[transit]:** - all characters in the field
  - keep dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - replace spaces and dashes with zeros
- **[check #]:** - all characters in the field

Fmt 21xx: **[transit] [acct #] [check #]**

- **[transit]:** - all characters in the field
  - keep dashes
- **[acct #]:** - always xx characters, zero filled;
  when xx=00 all characters are sent
  - replace spaces and dashes with zeros
- **[check #]:** - all characters in the field

Fmt 22xx: **[bank #] [acct #] [check #]**

- **[bank #]:** - all characters in the field
  - keep dashes
- **[acct #]:** - always xx characters, zero filled;
  when xx=00 all characters are sent
  - replace spaces and dashes with zeros
- **[check #]:** - all characters in the field

Fmt 23xx: **[error #] [transit] [acct #] [check #] 'S'**

- **[error #]:** - one digit, always present
  - '0' read OK
  - '1' read error: bad char, empty field, invalid length, validation
- **[transit]:** - always 9 characters, zero filled
  - keep dashes
- **[acct #]:** - always xx characters, trailing spaces;
  when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - always 6 characters, zero filled
  - remove spaces and dashes

Fmt 24xx: **[transit] 'T' [acct #] 'A' [check #] 'C' [amount] '$'**

- **[transit]:** - all characters in the field
  - keep dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - always 6 characters, zero filled
- **[amount]:** - all characters in the field

Fmt 25xx: **'M' 'C' [transit] 'D' [acct #] 'E' [check #]**

- **[transit]:** - all characters in the field
  - remove dashes and keep spaces (contig spcs = 1 spc)
  - if the field is empty, remove 'C'
- **[acct #]:** - include leading characters
  - maximum of xx characters; when xx=00 all characters are sent
  - remove dashes and keep all spaces
  - if the field is empty, remove 'D'
- **[check #]:** - all characters in the field
  - if the field is empty, remove 'E'

Fmt 26xx: **[acct #]**

- **[acct #]:** - work with characters in acct and transit fields
  - a window of xx characters; xx must be greater than 00
  - remove spaces and dashes

Fmt 27xx: **[acct #]**

- **[acct #]:** - work with characters in the acct field only
  - a window of xx characters; xx must be greater than 00
  - remove spaces and dashes

Fmt 28xx: **[acct #]**

- **[acct #]:** - work with characters in the acct field only
  - a window of xx characters; xx must be greater than 00
  - minimum of 6 digits, fill with zeros if necessary
  - remove spaces and dashes

Fmt 29xx: **'C' '/' [transit] '/' [acct #] '/' [check #] '/' [status]**

- **[transit]:** - all characters in the field
  - keep dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - maximum of 6 digits
- **[status]:** - this is a programmable option that must be enabled (See Table 2-4).

Fmt 30xx: **[zero fill] [transit] [acct #]**

- **[zero fill]:** - if length of (transit+account) is less than xx; xx must be greater than 00
- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #]:** - all characters in the field
  - remove spaces and dashes

Fmt 31xx: **[transit] '/' [acct #] '/' [check #]**

- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - maximum of 10 digits
  - remove spaces and dashes
  - if no check number, remove preceding slash ('/')

Fmt 3200: **'^' [transit] '^' [acct #] '^' [check #] '^' [status]**

- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #]:** - all characters in the field
  - remove spaces and dashes
- **[check #]:** - all characters in the field
  - remove spaces and dashes
- **[status] :** - this is a programmable option that must be enabled (See Table 2-4).

Fmt 3300: **'=' [transit] '=' [acct #] '=' [check #] '=' [status]**

- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #] :** - maximum of 14 digits
  - remove spaces and dashes
- **[check #]:** - maximum of 8 digits
  - remove spaces and dashes
- **[status]:** - this is a programmable option that must be enabled (See Table 2-4).

Fmt 34xx: **[transit] [acct #] [zero fill]**

- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #]:** - all characters in the field
  - remove spaces and dashes
- **[zero fill]:** - zero filled up to xx; xx must be greater than 00

Fmt 3500: **MA [aux] B [epc] C [tran] D [acct] E [chk] F [tpc] G [amt]**

This format is defined specifically for Target Test Checks. A description of the Target Test Check must be loaded in the exception table.

- **[aux], [epc], [tran], [chk], [tpc], [amt]:**
  - all characters in the field
  - keep spaces and dashes
- **[acct]:** - all characters in the field
  - keep spaces and remove dashes

Fmt 36xx: Read OK : **[transit] [acct #] [check #] '/'**
Read error: **'0' '/'**

- **[transit]:** - all characters in the field
  - remove spaces and dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - always 6 characters, zero filled
  - remove spaces and dashes

Fmt 37xx: **[ABA] [chk dgt] [acct #]**

- **[ABA], [chk dgt]:**
  - all characters in the field
  - keep spaces and dashes

- **[acct #]:**
  - work with characters in the acct field only
  - window of xx characters; xx must be greater than 00
  - remove spaces and dashes

Fmt 38xx: **'T' [transit] 'A' [acct #] 'C' [check #]**

- **[transit]:**
  - all characters in the field
  - keep dashes

- **[acct #]:**
  - maximum of xx characters; when xx=00 all characters are sent
  - include leading characters
  - keep spaces and dashes

- **[check #]:**
  -all characters in the field

Fmt 39xx: **[transit] <CR> [acct #]**

- **[transit]:**
  all characters in the field
  - remove dashes

- **[acct #]:**
  - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and keep dashes

Fmt 40xx: **[country code] [transit] [acct #]**

- **[country code]:**
  - '1' for US checks
  - '2' for Canadian checks

- **[transit]:**
  - all characters in the field
  - remove dashes

- **[acct #]:**
  - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes

Fmt 4100: **'S' 'T' [transit] 'A' [acct #] 'C' [check #]**

- **[transit]:**
  - all characters in the field
  - remove dashes

- **[acct #]:**
  - all characters in the field
  - place a slash ('/') after 10th character
  - if 10 characters or less, precede with a slash ('/')
  - remove spaces and dashes
- **[check #]:**
  - always 6 characters, zero filled
  remove spaces and dashes

Fmt 42xx: US check : **[transit] [acct #]**

Can check: **'9' [transit] [acct #]**

- **[transit]:** - all characters in the field
  - remove dashes

- **[acct #]:** - always xx characters; zero filled;
  when xx=00 all characters are sent.
  - remove spaces and dashes

Fmt 43xx: **[check #] <CR> <CR> [transit] <CR> [acct #]**

- **[check #]:** - maximum of 6 digits
  - remove spaces and dashes

- **[transit]:** - all characters in the field
  - remove dashes

- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes

Fmt 44xx: **[transit] [acct #]**

- **[transit]:** - all characters in the field
  - if Canadian check, replace dash with a space

- **[acct #]:** - always xx characters, trailing spaces,
  when xx=00 all characters are sent
  - remove spaces and dashes

Fmt 45xx: **[transit] <CR> [acct #] <CR> [check #]**

- **[transit]:** - all characters in the field
  - remove dashes

- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces, dashes and leading zeros

- **[check #]:** - all characters in the field

Fmt 46xx: **[transit] [acct #] [check #]**

- **[transit]:** - all characters in the field
  - remove dashes

- **[acct #]:** - always xx characters, zero filled;
  when xx=00 all characters are sent
  - remove spaces and dashes

- **[check #]:** - always 6 characters, zero filled
  - remove spaces and dashes

Fmt 47xx: **[transit] 'T' [acct #] 'A' [check #]**

- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - all characters in the field

Fmt 48xx: **[transit] 'T' [acct #] 'A'**

- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes

Fmt 49xx: **[transit] '/' [acct #] '/' [check #] '/' [check type]**

- **[transit]:** - always 9 characters, zero filled
  - remove dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - maximum of 9 digits
- **[check type]:**- personal checks ('1'); commercial checks ('2')

Fmt 50xx: **'T' [transit] 'T' 'O' [acct #] 'O' [check #]**

- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - all characters in the field

Fmt 51xx: **'=' [transit] '=' [acct #] '='**

- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes

Fmt 52xx: **'T' [transit] 'T' [acct #] 'A' [check #]**

- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - all characters in the field
  - remove dashes and spaces

Fmt 53xx: **'/' [transit] '/' [acct #] '/' [check #] '/' [tpc] '/' [status] '/'**

- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - all characters in the field
- **[tpc]:** - all characters in the field
- **[status]:** - this is a programmable option that must be enabled (See Table 2-4)

Fmt 54xx: **[transit] [acct #] [check #] [status]**

- **[transit]:** - always 12 characters, zero filled
  - remove dashes
- **[acct #]:** - always xx characters, zero filled;
  when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - always 12 characters, zero filled
  - remove dashes and spaces
- **[status]:** - this is a programmable option that must be enabled (See Table 2-4)

Fmt 55xx: **'C' '/' [acct #] '/' [transit] '/' [check #] '/' 0000000000**

- **[acct #]:** - always xx characters, zero filled;
  when xx=00 all characters are sent
  - remove spaces and dashes
- **[transit]:** - all characters in the field
  - remove dashes
- **[check #]:** - always 6 characters, zero filled
  - remove dashes and spaces

Fmt 56xx: **[transit] <CR> [acct #] <CR> [check #] <CR> [amount]**

- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - all characters in the field
  - remove dashes and spaces
- **[amount]:** - all characters in the field
  - remove dashes and spaces

Fmt 57xx: **[acct #] <CR> [amount]**

- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[amount]:** - all characters in the field
  - remove dashes and spaces

Fmt 58xx: **[short transit] [acct #] ':'**

- **[transit]:** - 3 rightmost characters
  - remove dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes

Fmt 59xx: **[transit] [acct #] <TAB> [check #] [amount]**

- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - always 9 characters, zero filled
  - remove dashes and spaces
- **[amount]:** - all characters in the field
  - remove dashes and spaces
  - insert decimal point ('.') before 2nd rightmost digit

Fmt 60xx: **[transit] '/' [acct #] '/' [check #] '/' [check type]**

- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - maximum of 10 characters
  - remove spaces and dashes
  - if no check #, remove preceding slash ('/')
- **[check type]:**- personal checks ('1'); commercial checks ('2')

Fmt 61xx: **[transit] <TAB> [acct #] <TAB> [check #] <TAB>**

- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces, dashes and leading zeros
- **[check #]:** - all characters in the field

Fmt 62xx: **'T' [transit] 'T' [acct #] 'A' [check #] 'S' [status]**

- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - all characters in the field
  - remove dashes and spaces
- **[status]:** - this is a programmable option that must be enabled (See Table 2-4).

Fmt 63xx: **[transit] [acct #] [check #]**

- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - always 4 characters, zero filled
  - remove spaces and dashes

Fmt 64xx: **[transit] [acct #] [check #] [amount]**

- **[transit]:** - all characters in the field
  - keep dashes
- **[acct #]:** - always xx characters, trailing spaces;
  when xx=00 all characters are sent
  - keep spaces and dashes
- **[check #]:** - always 6 characters (N is on quick-init check), trailing spaces
  - remove spaces and dashes
- **[amount]:** - all characters in the field
  - remove spaces and dashes
  - insert decimal point ('.') before 2nd rightmost digit

Fmt 65xx: **'!' [transit] '/' [acct #] '/' [check #] '/' [amount]**

- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - all characters in the field
  - remove dashes and spaces
- **[amount]:** - all characters in the field
  - remove dashes and spaces

Fmt 66xx: **[transit] [acct #] <CR> '7' '1' <CR>**

- **[transit]:** - all characters in the field
  - keep dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes

Fmt 67xx: **<CR> <CR> [check #]**

- **[check #] :** - maximum of xx characters; when x=00 all characters are sent
  - remove spaces and dashes

Fmt 68xx: **[transit] <TAB> [acct #] <TAB> [check #] <TAB> [amount] <TAB>**

- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - all characters in the field
  - remove dashes and spaces
- **[amount]:** - all characters in the field
  - remove dashes, spaces and leading zeros
  - insert decimal point ('.') before 2nd rightmost digit

Fmt 69xx: Read OK : **[transit] [acct #] [check #]**

Read error: **'0' '/'**

- **[transit]:** - all characters in the field
  - remove dashes
- **[acct #]:** - always xx characters, trailing spaces;
  when xx=00 all characters are sent
  - remove spaces and dashes
- **[check #]:** - always 6 characters, zero filled
  - remove dashes and spaces

Fmt 70: **[transit] ',' [acct #] ',' [check #] ',' [amount]**

- **[transit]:** - all characters in the field
  - keep dashes
- **[acct #]:** - always N characters (N is on quick-init check), space filled
  - remove spaces and dashes from the account
- **[check #]:** - always 8 characters, zero filled
  - remove dashes and spaces
- **[amount]:** - all characters in the field
  - remove dashes and spaces
  - if amount is not present, remove last ','

Fmt 71: **[acct #] '?' [check #]**

- **[acct #]:** - work with a window of N characters in the acct field
  - always N characters (N is on quick-init check), zero filled
  - remove spaces and dashes

- **[check #]:** - maximum of 4 characters
  - remove spaces and dashes

Fmt 72: **[transit] <TAB> [acct #]**

- **[transit]:** - all characters in the field
  - remove dashes

- **[acct #]:** - maximum of N characters (N is on quick-init check)
  - remove spaces and dashes

Fmt 73: **[transit] <CR> [acct #] <CR> [check #]**

- **[transit]:** - all characters in the field
  - remove dashes

- **[acct #]:** - maximum of N characters (N is on quick-init check)
  - remove spaces and dashes

- **[check #]:** - all characters in the field
  - remove dashes and spaces

Fmt 74: **[transit] [acct #] [check #]**

- **[transit]:** - all characters in the field
  - remove dashes

- **[acct #]:** - always N characters (N is on quick-init check), zero filled
  - remove spaces and dashes

- **[check #]:** - always 8 characters, zero filled
  - remove spaces and dashes

Fmt 75xx: **[transit] <CR> [acct #] <CR> [check #] <CR> [status]**

- **[transit]:** - always 9 characters, zero filled
  - keep dashes; remove spaces

- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove dashes and spaces

- **[check #]:** - maximum of 12 characters
  - remove dashes and spaces

Fmt 76xx: **'T' [transit] 'A' [acct #] 'C' [check #] 'M' [raw data]**

- **[transit]:** - all characters in the field
  - remove dashes and spaces
- **[acct #]:** - maximum of xx characters; when xx=00 all characters are sent
  - remove dashes and spaces
- **[check #]:** - all characters in the field- remove dashes and spaces
- **[raw data]:** - translate MICR symbols to t,o,a,d

APPENDIX B. ERROR CODES AND MESSAGES

The following are Error Codes and Error Messages from Excella device API MTXMLMCR.dll.

Error Code	Error Message	Error Code	Error Message
0	"Success"	25	“Error connecting to the device.”
1	"Unknown Error"	26	“Error device is not open.”
2	"Bad Parameter"	27	“Error getting the pointer to DOM.”
3	“Operation Pending”	28	"Error loading XML."
4	“Operation Failed”	29	"Error finding key number."
5	“OverFlow”	30	"Error connecting to internet."
6	“Device Not Found”	31	"Error HTTP Open."
7	“Access Denied”	32	"Error HTTP Send."
8	“Device Not Responding”	33	"Error creating event."
9	“Device Not Ready”	34	"Error DOM: creating node."
10	“Hopper Empty”	35	"Error DOM: query interface."
11	“Error Copying File”	36	"Error DOM: add key."
12	“Error Reading Check”	37	"Error DOM: append child."
13	“No Images Specified”	38	"Error DOM: get document element."
14	“No Image Found”	39	"Error DOM: get xml."
15	“No Checks to Process”	40	"Error DOM: get item."
16	“Not Enough Memory”	41	"Error DOM: get child node."
17	“Key not found”	42	"Error DOM: get base name."
18	“Section not found”	43	"Error DOM: get length."
19	“Invalid Section”	44	"Error DOM: get element by tag name."
20	“Error Bad Buffer Length”	45	"Error DOM: get text."
21	“Function not supported”	46	"Error DOM: put text."
22	“Error allocating memory”	47	"Error HTP Query Info."
23	“Request timed out.”	48	"Error insufficient data."
24	“Error getting the length of the data returned from the device.”	49	“Error Bad HTTP Connection”

Error Code	Error Message	Error Code	Error Message
50	“Error Get Zero Content Length”		
51	“Error Bad Device Name”		
52	“Error Bad Doc Info Buffer”		
53	“Error Bad Section Name”		
54	“Error Bad Key Name”		
55	“Error Bad Value Buffer”		
56	“Error Bad Buffer Length”		
57	“Error Bad Query Parameter”		
58	“Error Bad Image Name”		
59	“Error Bad Buffer”		
60	“Error Bad Buffer Size”		
61	"Error Bad Connect Request Timeout."		
62	"Error Insufficient Disk space."		
63	"Error MSXML Not Found."		
64	"Error Query Content failed."		
65	"Error Internet Connection failed."		
66	"Error Bad IP or Domain name."		
67	"Error USB Get Data Failed."		
68	"Error WinInet Get Data Failed."		
69	"Error HTTP Header Not Found."		
70	"Error USB Send Request Failed."		

APPENDIX C. HOW TO PROCESS CHECKS AND GET IMAGE QUALITY ASSURANCE

Note

This procedure is only for Excella /Excella STX Software installed with ImageScore optional feature.

The following steps describe the process of scanning a check to perform a set of tests on the scanned image to ensure the image qualifies a number of quality and usability criteria specified by the X9.37 standard:

1. Find the attached Excella/Excella STX device by using function **MTMICRGetDevice**.

2. Use function **MTMICROpenDevice** to open the device.

3. Prepare a process recipe for the device to read the check. This process recipe specifies the feeder used to feed the check, type of image, resolution of image, and text for endorsing and/or franking, etc. The process recipe is prepared using the **MTMICRSetValue** and **MTMICRSetIndexValue** function. See Section 2, **MTMICRSetValue** for more information**.**

4. The Image QA tests are performed on black & white images only. To get the black & white image, the process must have "BW" as the value for **ImageColor** key. The valid file type for a black & white image is a "TIF".

5. The Image QA tests are performed on the front side and back side of the check. The process recipe, therefore, must specify a value of 2 or more for the **Number** key.

6. The process recipe is sent to the device using the function **MTMICRProcessCheck**. Excella/Excella STX reads the check with the given process recipe and returns the results to the Host application. The result is stored in the third parameter of the **MTMICRProcessCheck** function. The result contains only information of the process and information of the device. Information such as size of scanned images and image identifier are included. It does not contain scanned data.

7. After Excella/Excella STX reads the check, images are retrieved from the device. The image information can be found in the buffer specified in the third parameter of **MTMICRProcessCheck** function, when the function is returned. This buffer is contained in the **ImageInfo** section. The **ImageInfo** section contains key/value pairs consisting of **ImageSize** and **ImageURL**. The value of a key **ImageURL** is the ID of an image. The **MTMICRGetValue** and **MTMICRGetIndexValue** function are used to retrieve the value of image ID and the size of an image from the **ImageInfo** section.

8. Images are retrieved using function **MTMICRGetImage** or **MTMICRGetImages**.

9. When images are ready to test, the process for Image QA Tests can start. The operation for Image QA must be setup before the tests can run. Use **IS_Initialize** function to initialize the operation. This function must be executed only once prior to running any Image QA tests.

10. Test metrics for Image QA Tests must be setup before running the tests. A default copy of "SampleSetup.iss" is installed in the installation directory. This file "SampleSetup.iss" contains default values for all Image QA test metrics. For more information on these test metrics, see ImageScore manual (Please contact Help Desk).

11. Use function **IS_LoadSetup** to load the pre-defined setup file. When a setup file is loaded, any change to this file will not be taken into effect until you unload the old setup file. Use function **IS_DeleteSetup** to unload the old setup file then use function **IS_LoadSetup** to load the new setup file.

12. Use function **IS_LoadImage** to load the image of the front and back side of check. If the return value of this function is less than 0 then the images are not qualified for Image QA Tests. If the return value of **IS_LoadImage** equals to 0 then the image is ready to run Image QA tests.

13. Use function **IS_Run** to perform all Image QA tests. If the return value of this function is less than 0 then the tests failed to perform on the specified setup file and/or specified loaded images. If the return value of **IS_Run** function equals to 0 then the results of Image QA tests are ready for retrieval.

14. Use function **IS_GetResult** to retrieve the results of the Image QA test whether the test results in a pass or fail. This function must be executed for each type of Image QA test run.

15. When all Image QA tests are completed, use function **IS_DeleteImage** to unload the old images and to regain resources used on the old images.

16. To run tests on a new image or to rescan a failed image, go to step 3 to repeat the check processing.

17. When application is terminated, the **IS_Terminate** function must be executed once to regain resources used for Image QA operation.

INDEX