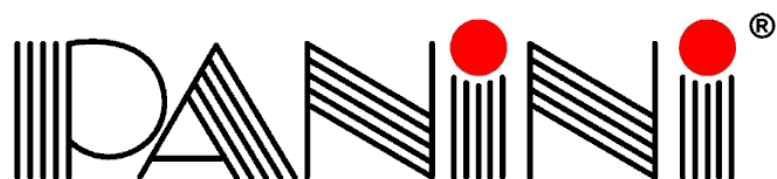


Panini Vision API



Reference Manual

Version 4.5.0

Panini Vision API 4.5.0
Reference Manual Revision 1

DATE	ISSUE	AMENDMENT DESCRIPTION
March 8, 2013	4.2.0 R0	Added new Extended parameters for UV images enhancement. Added Appendix H describing API Startup mechanism.
March 21, 2013	4.2.0 R1	Added info about Feeding mode. Added info about Images @600 DPI for VisionNeXt.
April 2013	4.2.0 R2	Added info about GetDeviceFeature.
July 2013	4.2.0 R3	Added info about Vision NeXt erros codes.
March 2014	4.2.2 R0	Added <code>OCR1_DATAMATRIX</code> code. Added new <code>VapiWaitForInit()</code> function. Added new <code>ERR_DUE_TO_PREFEED_FAILURE</code> Sorter Error for VisionNeXt.
April 2014	4.2.2 R1	Added new FW parameter for Vision NeXt.
September 2014	4.4.0 R0	Added new barcode formats inside <code>GetOCRCodeline()</code> function (<code>OCR1_QRCODE</code> , <code>OCR1_BC93</code> , <code>OCR1_DATABAR</code> , <code>OCR1_PATCH</code> , <code>OCR1_BCAUTO1D</code> , <code>OCR1_BCAUTO2D</code>). Added Recapture image after printing section (Appendix D: VisionX).
February 2015	4.4.1 R3	Added new AGP-14 Cap & Clean section inside Appendix G: VisionNext. Added new <code>VapiLoad/VapiUnload</code> in <i>VisionAPI layers Functions</i> section. Added new Appendix I describing new API Logging mechanism.
April 2015	4.4.2 R0	Rewritten the Compressed Image for Decision section on every device Appendix. Added OEM logging description on Appendix I.
June 2015	4.4.2 R1	Added <code>DEVICE_ERR_LIFT</code> description
July 2015	4.4.3 R0	Changed the behaviour of <code>GetMagCardResult</code> and added a new <code>GetMagCardResultString</code> function for VX.

Panini Vision API 4.5.0
Reference Manual Revision 1

Dec 2016	4.5.0 R0	<p>Added <code>GetDeviceStatus</code> API call. Added new VisionNext Cap & Cleaning logic. (VN only)</p> <p>Added support for PNG image formats.</p> <p>Added new Appendix J describing PDF document creation support and functions (cheques only).</p> <p>Added new Appendix K describing VisionAPI Multiple Application support.</p> <p>Implemented new MICR+OCR behavior.</p>
May 2016	4.5.0 R1	<p>Added True Color (cards only) support for VN.</p>

Contents

Contents	3
Overview.....	9
Device States	12
Device cycle through State description.....	13
On Line Mode (Processing Document Mode).....	13
Off Line (Diagnostic Mode)	15
Device Parameters Structure.....	15
Compatibility between <i>DeviceParameters</i> options.....	21
Image and Snippet retrieved structure Description.....	24
Firmware Parameters Description.....	24
Pockets Handling.....	25
Smart Jet	27
True Color	28
UV Images	28
MICR+OCR	29
New behavior (API 4.5+)	29
Reverse gear	30
Virtual Endorsement.....	31
IQA	36
Extended parameters	38

Error Handling.....	43
USB Errors.....	44
Device Errors	45
API Errors.....	48
Sorter Errors.....	49
Function Calls	51
VisionAPI layers functions	51
VApiGetVersion.....	51
VApiSetDeviceEngine	52
VApiGetDeviceList.....	52
VApiSelectDevice	53
VApiGetError.....	54
VApiGetErrorString	54
VApiWaitForInit	55
VApiLoad.....	55
VApiUnload	56
VisionAPI errors.....	56
Device layer functions	57
Information Functions	57
GetApiRelease	57
GetEngineApiRelease	57
GetDriverRelease	58
GetFWVersion.....	59
GetSerialNumber.....	59
ReadCryptedIDCard	60
IDCard and capabilities request functions	61
GetIDCardDescription.....	61
UpgradeIDCard	62
GetDeviceFeature	63
Error Request Functions	67
GetUsbError	67
GetUsbErrorString.....	68
GetDeviceError	68
GetDeviceErrorString.....	69
GetApiError.....	70
GetEngineApiError	70
GetApiErrorString	71
GetEngineApiErrorString.....	72

GetSorterError	73
GetSorterErrorString	73
GetIQALastError	74
GetIQALastErrorString	75
Device States Control Functions	76
GetDeviceState	76
GetDeviceStateString	76
StartUp	77
ShutDown	78
ChangeParameters	78
OnLine	79
OffLine	80
GetDeviceStatus (VN only)	80
Parameters Managing Functions	82
SetSorterParameter	82
GetSorterParameter	82
SetMicrOcrFontOverride	83
SetDeviceParameters	84
GetDeviceParameters	85
SetDocProcessingParam	85
SetImageAdjustment	94
SetMaxDpm	95
GetMaxDpm	96
SetAGPLines	96
SetFeederLimit	97
SetHandDropDly	98
GetAvailablePockets	98
SetMaxPocket	99
SetFeedingMode	100
SetVirtualEndorsementProperty	101
GetVirtualEndorsementProperty	101
SetIQAParameter	103
SetExtendedParameter	107
GetExtendedParameter	108
On Line Functions	109
IsFeederEmpty	109
IsFeederEmpty (VN only)	109
StartFeeding	110
StopFeeding	111
FreeTrack	112
SetPocket	112
GetDocumentLength	113
GetMicrCodeline	114
GetOCRCodeline	116

Panini Vision API 4.5.0
Reference Manual Revision 1

SendPrinterData.....	118
SetVirtualEndorsement	120
FreeImagesBuffer	122
FreeSnippetBuffer	123
DisableDownstreamImage	123
GetDoubleFeedingResult	125
GetFullPocketStatus	126
GetFullPocketStatus (VN only).....	127
GetDocumentType	128
EnableFranking	129
IsTrackFree.....	130
GetIQAResults	131
GetIQAValues	132
ProcessPage	132
Serial Functions	134
Rs232SetBaud	134
Rs232Write.....	134
Rs232GetLen.....	135
Rs232Read.....	136
Maintenance Functions	136
ReadPrinterDropsCounter	136
ResetPrinterDropsCounter.....	137
GetPrinterCartridgeInfo.....	137
GetHistoricalCounter.....	138
ResetHistoricalCounter	139
ServicePrinterHead (VN only)	140
Magnetic card reader	141
GetMagCardResult, GetMagCardResultString	141
Appendix A: Devices comparison.....	144
Interface functions	144
Implemented messages list	146
Device parameters	146
Appendix B: Vision S	148
USB connection.....	148
USB driver.....	148
Vision S Messages.....	149
Full pocket handling	152
Flow-Mode remarks	153
Firmware parameters	153
Appendix C: I:Deal.....	155
Document life cycle.....	155

DeviceParameters	157
SetPocket	158
Compressed Images for Decision	158
Sorter errors	164
Connection attempt timeout	165
Exception errors	165
Appendix D: VisionX	166
Compressed Images for Decision	166
Devices with maximum DPM greater that 100	168
Recapture image after printing	168
GetDeviceFeature	169
Appendix E: wl:Deal	170
Document life cycle	171
DeviceParameters	173
SetPocket	174
Compressed Images for Decision	175
Sorter errors	180
Connection attempt timeout	181
Exception errors	181
Appendix F: VisionX Page Scanner Extension	183
Document life cycle	183
DeviceParameters	185
SetPocket	185
Compressed Images for Decision	186
Sorter errors	188
Connection attempt timeout	189
Exception errors	189
Gamma Correction	190
Appendix G: VisionNext	191
Document life cycle	191
DeviceParameters	193
SetMaxPocket	194
SetPocket	194
Sorter errors	195
Image resolutions	198
GetFullPocketStatus (VN only)	198
IsFeederEmpty (VN only)	201
Mobile AGP-14 printer	202
AGP-14 Inkjet Cartridge Maintenance Best Practices	207
Firmware parameters	209
Appendix H: API StartUp mechanism	211

Appendix I: API Logging mechanism	213
DebugView logging	213
File logging.....	213
OEM logging.....	214
 Appendix J: PDF document support.....	 215
Functions to set the format of the PDF output	215
Function to get the PDF output	216
 Appendix J: Multiple Applications support	 218
Background	218
Vision API changes	218
Applications changes.....	219
PaniniUDS service	219

Overview

The VisionAPI is the software interface to drive the Panini's devices. The VisionAPI is the "standard" API for every machine manufactured by Panini. This API will be able to drive different kind of machine.

It organizes the software interface in two layers. It's composed of an "Interface" library and a "device engine" library that contains the specific code of a specific device.

It's organized as a set of Microsoft Win32 DLLs.

Panini Vision API has been created to supply our customers' requests of an easy-to-use and very specific Interface. In fact, using very simple and direct function calls, every software programmer is able to use every low level feature of our devices with all the benefits that this brings (e.g. the complete access to the image options, or the possibility to use our powerful diagnostic tools using our offline function).

With high level functions it is possible to download the images from the scanners, to recognize the MICR String, to send string or bitmap to the printer, and so on.

Panini Vision API philosophy is very simple: we provide our C++ header file with the prototypes of the function, the .dll and the .lib API files. The include file is written and tested with Visual C++ Microsoft compiler.

The interface is composed of two files.

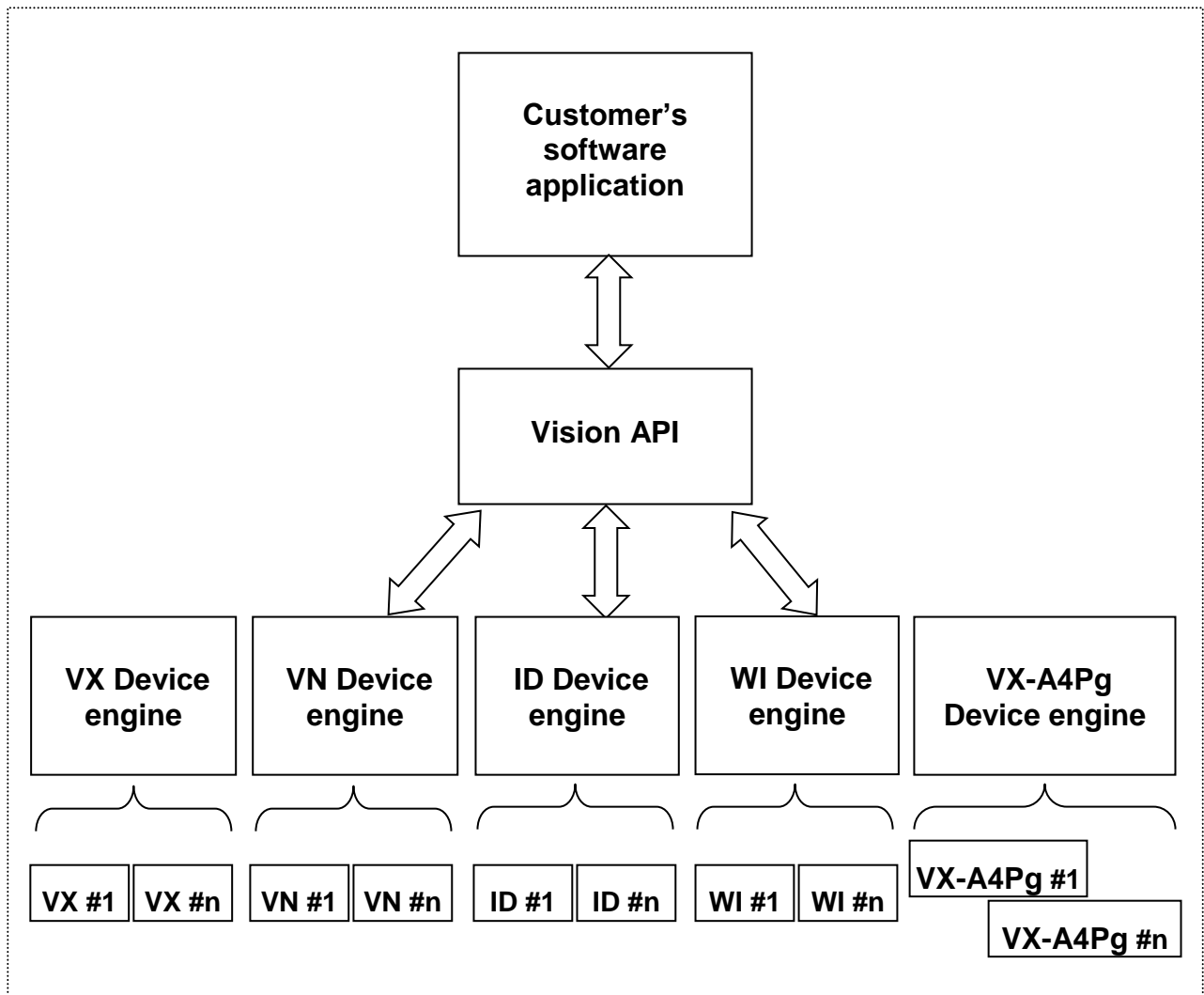
1. *VisionAPI.dll* is the software interface exposed to the customers' application.
2. *<Engine>.dll* is the "device engine" that manages all the specific operations with the physical device (MyVisionX or VisionX or I:Deal or wI:Deal or VisionX Page Scanner Extension [Vx-A4Pg] or VisionNext).

The release of the 3.6.1 Version of the Panini VisionAPI allows the application to manage simultaneously more than one device at the same time. For all the installed engine, the list of the connected devices can be retrieved using the function call **VApiGetDeviceList**. At this point calling the function **VApiSelectDevice**, before the StartUp, the specific device can be selected. From this point the device will be addressed with its personal DeviceID. This can be repeated for up to 16 devices. If the **VApiSelectDevice** function is not called, the API will work in the compatibility mode, so all the software written for the previous version of the software is still compatible.

Calling the **StartUp** the windows handle to the dialog need to be passed with the window message that will be fired. This will allow the application to have more dialogs to managed the same window message or have one dialog to handle all the different message coming from the different devices.

The device ID will remain valid until the device is shutdown, even if it is disconnected. So if it is connected again the device will be addressed with the same device ID.

The following chart shows the actual software organization:



The entire set of file needed to install VisionAPI is composed of the following list of files:

- VisionAPI.dll;
- One or more device engine library (VxEngine.dll, IDEngine.dll, WiEngine.dll and VxA4Engine.dll, X2Engine.dll);
- Baroc.dll (OCR engine, optional);
- AxBar32.dll (Barcode engine, optional);
- PaniniABY.dll and PaniniABY folder (ABBYY engine wrapper, optional)
- PaniniOCR.dll, MICR_OCR.dll, IQA_Engine.dll, ImageDecoder.dll (OCR engine for MICR+OCR and IQA, optional);
- PaniniOCRParms folder (data for the PaniniOCR engine, optional);
- PaniniOCR depends on few OpenCV v2.4.0 libraries: opencv_core246.dll, opencv_calib3d246.dll, opencv_features2d246.dll, opencv_flann246.dll, opencv_highgui246.dll, opencv_imgproc246.dll, opencv_legacy246.dll, opencv_ml246.dll, opencv_video246.dll.;
- Microsoft WHQL USB Driver (the Panini.INF file, PaniniUSB.dll and PaniniUSB.sys);

Panini Vision API 4.5.0 Reference Manual Revision 1

- PaniniExtension.dll, PaniniSNMPclient.dll, pnnXtns.cfg (Panini Avantor engine, optional);
- PaniniULD.dll (Panini USB driver utilities).

The VisionAPI maintains all the functions defined in the previous MyVisionX API library. Panini will maintain this interface unchanged in order to obtain the application's backward compatibility. The interface will change just adding new functions, without modifying the existing ones.

The exported functions can be grouped in two different set. One group is named as “interface layer function” and the other is named as “device engine layer functions”. All the functions are defined in the “VApiInterface.h” header file.

The interface layer functions are named with the prefix VApi- and are the ones related to the VisionAPI.dll layer.

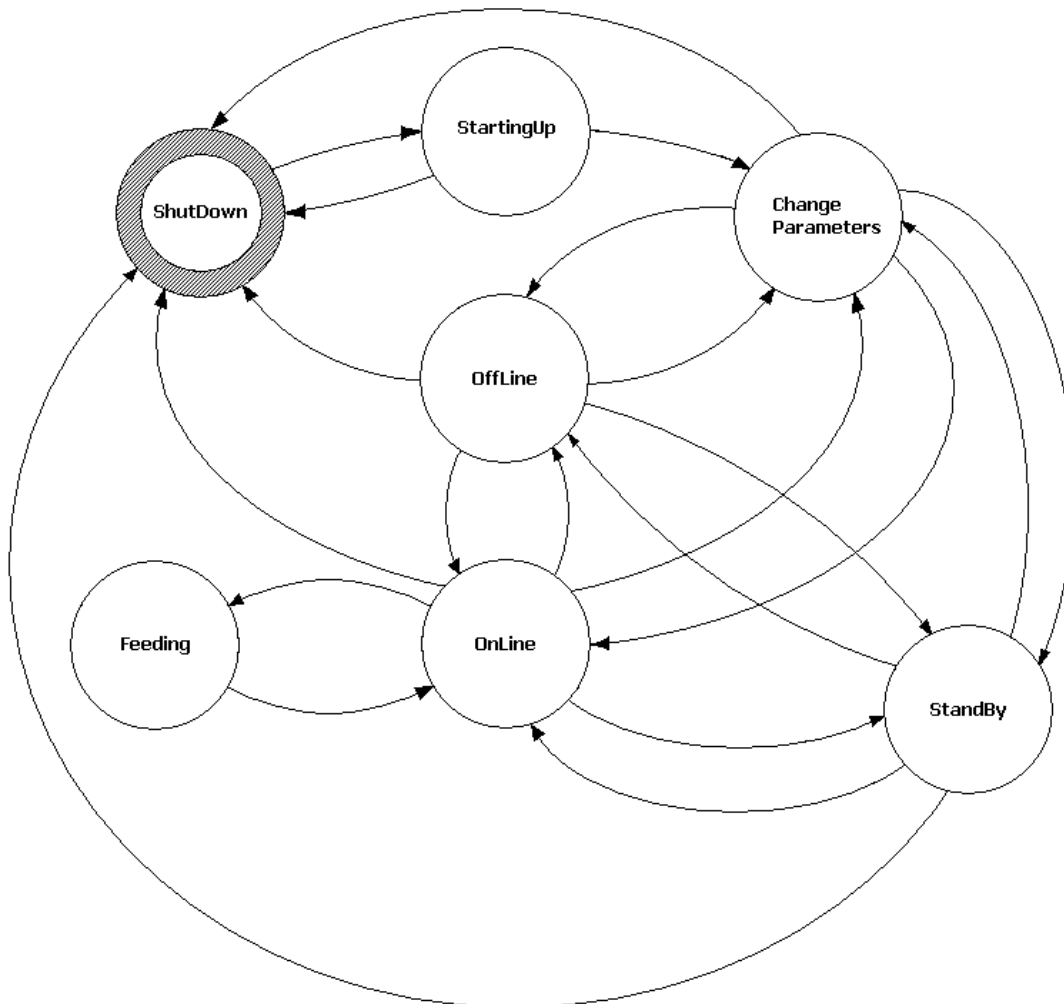
The device layer functions are related to the engine layer. A Panini device does not necessary support all the engine functions. A VisionX device supports all the engine's functions. In the future a new device could support a subset of them. If this happens, the not supported functions will return an error and VisionAPI error “function non available” is set. See the appendix in order to identify which the supported for each kind of devices.

The “device” abstraction is substantially a Finite State Machine, in which every state describes the physical status of the reader. With this assumption, it's easier to understand the real behaviour of the machine. The actual state of the Device can be queried at any time using the *GetDeviceStateString* function.

The device communicates with the user application by common Windows Messages, which describes the reader connection status and provide real-time document processing status notification. Each Panini Vision API application should define its own message handlers, which will trap all relevant conditions; for this purpose, the usual technique could be defining and opening an invisible window whose handle will be registered as the messages default destination.

Panini Vision API will not save or read information to or from any .ini files, just to have no possible overlapping between what the API does internally and what the customers should do in their applications. In Panini Vision API philosophy, it's an application's responsibility to store all default parameters and send them to the API when needed.

Device States



The device state can be retrieved with the functions ***GetDeviceStateString*** and ***GetDeviceState***. Here is a brief description of the State of the FSM:

- ***DeviceShutDown (0)***: it's the “not operative” state
- ***DeviceStartingUp (1)***: it's the initialization state of the device, the first reached after the application calls the ***StartUp*** function. This state automatically switches to the next one when the sorter is connected.
- ***DeviceChangeParameters (3)***: this is the state in which it's allowed to change the options of the device.
- ***DeviceOnLine (5)***: this is the operative state, in which the device can be used to process documents.

- **DeviceOffline (6):** this is the state in which the offline functions (testing motors, photocell calibration check...) can be called.
- **DeviceStandBy (7):** this state is forced on and off when the rear button of the device is pressed for at least a second.
- **DeviceFeeding (8):** the reader is processing documents.
- **DeviceLocked (10):** an identification problem occurred, the device is unusable.

Device cycle through State description

The Start Up sequence follows these steps:

1. The device is in **DeviceShutDown** state.
2. Call the **StartUp** function, and the Finite State Machine goes in the **DeviceStartingUp** state.
3. When the API manages to connect the device, the message **WMPAR_SORTER_CONNECTED** is sent to the application message handler and the current state become **DeviceChangeParameters** state. If the connection can't be obtained, after a while the message **DEVICE_ERR_CONNECTION_TIMEOUT** is sent to the application.
4. At this point **OnLine** or **OffLine** function can be called and the Device will be forced to the corresponding state.

The Shut Down sequence can begin from almost every state (except for the **DeviceFeeding** State), simply call the **ShutDown** function and the device will be forced into that state.

Note that a message won't be sent to confirm that the device is not operative; the **WMPAR_SORTER_DISCONNECTED** is sent when the reader is unplugged or there are problems in the communication.

Press for at least one second the reader's rear button and the Device is forced into **DeviceStandBy** state and the **WMPAR_STANDBY** message is sent (the LPARAM is TRUE). To exit this state, press the button again and the device will return to the previous state, the **WMPAR_STANDBY** message will be re-sent with the LPARAM set to FALSE.

NOTICE: the driver doesn't support PC Stand by and Hibernation Mode. So if one of these events happens it will be necessary to unplug the device and re-plug it again to have the driver to be reloaded correctly.

On Line Mode (Processing Document Mode)

In order to use the machine in the processing Document Mode, the Device must be in the **DeviceOnLine** state (this can be reached using the **OnLine** function from the **DeviceChangeParameters** state, after the necessary changes to the processing parameters have been done by the application).

The Feeding Options are selected in the *DeviceChangeParameters* state and can be set between “Hand Drop Mode” and “Hopper Feed Mode”, and also between single document feeding and multi document feeding.

Call *StartFeeding* function to begin the processing of the documents, *StopFeeding* to end it; if the device is working in One Document mode or if an exception occurred, it is not necessary to call *StopFeeding*. During the document transport, if a jam occurs, use the *FreeTrack* function to eject the document from the track.

The progress of the document in the track is followed by the messages received from the application as described below:

1. *WMPAR_SORTER_NEW_DOCUMENT* is sent when the document enters the track, the LPARAM relates to this message.
2. *WMPAR_SORTER_MICR_AVAILABLE* is sent when the MICR Codeline has been fully recognized (if it had been requested), and it is ready for the application to use; to obtain it call the *GetMicrCodeline* function. The LPARAM of this message reports the document number.
3. *WMPAR_SET_DOWNSTREAM_OPTIONS* is sent when the application has to decide some specific downstream operations. It's a specific message used by the I:Deal engine. See the related appendix for more details.
4. *WMPAR_COMPRESSED_IMG_FOR_DECISION* is sent when the application has enabled the decision based on the compressed image and a franked front image has been requested too. . LPARAM returns a structure containing the requested images (see the description of Image and Snippet Structure).
5. *WMPAR_SORTER_SET_ITEM_OUTPUT* is sent when the document is ready to be pocketed; call the *SetPocket* function to set the destination pocket (this must be called even if the reader has only one pocket!). In this state, the data that has to be printed on the next document must be sent to the API through the function *SendPrinterData* (see function definition). The LPARAM of this message reports the document number.
6. *WMPAR_SORTER_DOCUMENT_INSIDE_POCKET* is sent when the document has been pocketed. The LPARAM of this message reports the document number.
7. *WMPAR_IMAGE_READY* is sent when the images from the scanners are available for the application. LPARAM returns a structure containing the requested images (see the description of Image and Snippet Structure).
8. *WMPAR_SNIPPET_READY* is sent when the snippet is ready for the application. This message will be received twice, the first time for the front snippets, and the second time for the rear ones; in this way, the front snippets are available very quickly for OCR recognition without waiting for the rear ones. LPARAM field of this message is a structure similar to the one used for the images.
9. *WMPAR_DOC_COMPLETED* is sent when all the processing phases have been completed. LPARAM is the document number.

Different document messages could be mixed, use the LPARAM to retrieve the document ID to clearly track the process of the document.

Off Line (Diagnostic Mode)

All the diagnostic functions must be called in the *DeviceOffLine* state (that can be reached using the *OffLine* function from the *DeviceChangeParameters* state).

NOTE: In this release of the Panini Vision API manual OffLine functions won't be documented.

Device Parameters Structure

This structure is the way in which the application communicates to the Panini Vision API the settings of the processing options. Its definition can be found in the header file.

It must be passed to the API in the *DeviceChangeParameters* state as parameter of the function *SetDeviceParameters*.

A description of all the options of the structure *DeviceParameters* follows:

Type	Name	Description
BOOL	<i>bMICREnable</i>	Enabling of the MICR Recognition
UINT	<i>nMICRFont</i>	Select the MICR font: <ul style="list-style-type: none">- MICR_FONT_CMC7 for CMC7 documents;- MICR_FONT_E13B for E13B documents;- MICR_FONT_AUTO for auto-recognition of both types of codeline;- MICR_FONT_E13B_OCR for E13B recognition reinforced by a “merge” with an OCR result;- MICR_FONT_CMC7_OCR for CMC7 recognition reinforced by a “merge” with an OCR result;- MICR_FONT_AUTO_OCR for MICR Automatic recognition reinforced by a “merge” with an OCR result.
BOOL	<i>bMICRSaveSamples</i>	Save MICR waveforms to file (the files will be store in the directory MICR Waveforms, automatically created by the API in the working directory). The name of the file will report the document number and the date it was written. The file contains, in its header, the recognized codeline.
UINT	<i>nMICRSpaces</i>	Sets the number of spaces returned in the MICR codeline. <ul style="list-style-type: none">- SPACES_NONE: no spaces returned among the codeline fields;- SPACES_ONE: one space returned;- SPACES_ALL: all the spaces are returned following the MICR spacing specification.

Panini Vision API 4.5.0
Reference Manual Revision 1

char	<i>cRejectSymbol</i>	Sets the symbol for reject characters (the default is '?'). This value must be a printable character and cannot be set to the MICR symbols (digit from '0' to '9', ':', ';', '<', '>', '=').
UINT	<i>nReserved</i>	Must be set to 0.
BOOL	<i>bReserved</i>	Must be set to FALSE.
IMAGE_PROPERTIES	<i>ImagePropertiesFront1</i>	Image property structure for the first front image (see description below).
IMAGE_PROPERTIES	<i>ImagePropertiesFront2</i>	Image property structure for the second front image (see description below).
IMAGE_PROPERTIES	<i>ImagePropertiesRear1</i>	Image property structure for the first rear image (see description below).
IMAGE_PROPERTIES	<i>ImagePropertiesRear2</i>	Image property structure for the second rear image (see description below).
SNIPPET_PROPERTIES	<i>SnippetProperties</i>[10]	Vector containing the property structure of the ten available snippets (see description below).
BOOL	<i>bPrintEnable</i>	<p>Enable the Ink-Jet Printer.</p> <p>Valid values for this parameters are:</p> <ul style="list-style-type: none"> - PRINTER_DISABLE (printer disabled); - PRINTER_ENABLE_LEADING (enable printer device and the position is referred to the leading edge of the document); - PRINTER_ENABLE_TRAILING (enable printer device and the position is referred to the trailing edge of the document). <p>These three values are defined in the header file.</p> <p>For the AGP Printer there are available 3 options to modify the printing quality and decrease the ink consumption.</p> <p>The options are the following:</p> <ul style="list-style-type: none"> - PRINTER_AGP_QUALITY_HIGHQ (it's the best quality, 100% of ink used to print); - PRINTER_AGP_QUALITY_NORMAL (about 66% of ink used to print); - PRINTER_AGP_QUALITY_DRAFT (about 33% of ink used to print). <p>Example:</p> <pre>bPrintEnable = PRINTER_ENABLE_LEADING PRINTER_AGP_QUALITY_NORMAL;</pre> <p>This example enables Printer, with leading edge alignment and normal printer quality.</p>

Panini Vision API 4.5.0
Reference Manual Revision 1

		<p>To enable the Smart Jet printer the application has to add to the bPrintEnable field the following flag:</p> <ul style="list-style-type: none"> - PRINTER_ENABLE_SMART_JET. <p>Example:</p> <pre> bPrintEnable = PRINTER_ENABLE_LEADING PRINTER_AGP_QUALITY_NORMAL PRINTER_ENABLE_SMART_JET </pre>
BOOL	<i>bOneDoc</i>	If TRUE, just one document will be processed when <i>StartFeeding</i> has been called. At the end of the process the state automatically will return to OnLine.
UINT	<i>nFeedingMode</i>	<p>Select the feeder mode.</p> <p>HOPPER_FEED for Main Hopper Feeding Mode, i.e. the feeder will recognize if a document is present, and in negative case return an error.</p> <p>DROP_FEED for Hand Drop Feeding Mode i.e. the feeder, even if it is empty, wait for document(s) to be inserted.</p>

Image property structure description:

Type	Name	Description
UINT	<i>Format</i>	<p>Set the image compression format. The available image formats are:</p> <p>FORMAT_NONE: no image requested</p> <p>FORMAT_GIV: Group 4 TIFF</p> <p>FORMAT_JPEG: gray-level JPEG</p> <p>FORMAT_BMP: gray-level uncompressed bitmap</p> <p>FORMAT_NATIVE_BMP: gray-level bitmap of the native image (as it is acquired from the scanner before any processing). It's intended to be used just for diagnostic purposes.</p> <p>FORMAT_JPEG_COLOR: color JPEG acquired with Panini Fast Color Technology.</p> <p>FORMAT_BMP_COLOR: color bitmap acquired with Panini Fast Color Technology.</p> <p>FORMAT_GIV_DROP_OUT_RED: G4 TIFF image acquired with the drop out of the red color.</p> <p>FORMAT_GIV_DROP_OUT_GREEN: G4 TIFF image acquired with the drop out of the green color.</p> <p>FORMAT_GIV_DROP_OUT_BLUE: G4 TIFF image acquired with the drop out of the blue color.</p> <p>FORMAT_BMP_TRUE_COLOR: True Color bitmap acquired with True Color mode.</p>

		<p>FORMAT_JPEG_TRUE_COLOR: color JPEG acquired with True Color mode.</p> <p>FORMAT_GIV_DOR_TRUE_COLOR: G4 TIFF image acquired with True Color mode for red drop out.</p> <p>FORMAT_GIV_DOG_TRUE_COLOR: G4 TIFF image acquired with True Color mode for green drop out.</p> <p>FORMAT_GIV_DOB_TRUE_COLOR: G4 TIFF image acquired with True Color mode for blue drop out.</p> <p>FORMAT_GIV_DROP_OUT_IR: G4 TIFF image acquired with the Infrared light</p> <p>FORMAT_JPEG_UV: gray-level JPEG acquired with Ultraviolet light.</p> <p>FORMAT_JPEG_UV_NEGATIVE: gray-level JPEG acquired with UV light and inverted.</p> <p>FORMAT_GIV_UV_NEGATIVE: G4 TIFF image acquired by UV light and inverted before binarization.</p> <p>FORMAT_GIV_UV: G4 TIFF image acquired with UV light .</p> <p>FORMAT_PNG: gray-level PNG (lossless)</p> <p>FORMAT_PNG_COLOR: color PNG acquired with Panini Fast Color Technology.</p> <p>FORMAT_PNG_TRUE_COLOR: color PNG acquired with True Color mode.</p> <p>Please, note that the Drop-Out Acquisition is incompatible with color acquisition and with G4 format, so, for instance, if the first required image is a FORMAT_GIV_DROP_OUT_BLUE image, the second can't be a color or a group 4 image.</p> <p>Fast color acquisition uses the nominal track speed. The True Color acquisition needs a track speed set to one third of the nominal speed.</p> <p>The IDCard enable some of these image formats.</p> <p>A <code>DEVICE_ERR_INVALID_PARAMETERS</code> error will be obtained if not compatible images are requested.</p>
UINT	Paging	<p>Set the paging option in order to create a multipage TIFF image.</p> <p>PAGING_ONLY_SINGLE: this image is not included in the multi-page TIFF</p> <p>PAGING_ONLY_MULTI: this image is included it in the multi-page TIFF and the single image buffer isn't returned</p> <p>PAGING_SINGLE_MULTI: this image is included in the TIFF and returned as a single one.</p>

Panini Vision API 4.5.0
Reference Manual Revision 1

		<u>NOTE:</u> Bitmaps and PNG images will not be included in the multi-page TIFF.
UINT	<i>Resolution</i>	<p>Set the resolution of the image (100, 200 or 300 DPI are allowed).</p> <p>If the requested image is a native bitmap, this will be ignored and the image will always be the maximum (300 DPI for Vision X, 200 DPI for MyVisionX).</p> <p>If the required images are either at the resolution of 100 DPI (for gray acquisition, i.e. no drop out and no color), they will be acquired with that resolution from the scanners, with an improvement of the throughput (in particular with USB 1.1 connection).</p>
UINT	<i>ColorDepth</i>	<p>Sets the number of gray level for the images, it can be set to 16, 64 or 256 grey levels (for color and dropped out images must be set to 256).</p> <p>For historical reason all values are still defined, but it's supported only the 256.</p> <p>If an application sets a different value, it will be ignored.</p>
UINT	<i>Threshold</i>	<p>Set the quality factor for JPEG images (1-99), the G4 density (1-9) or PNG compression factor (1-9). For bitmap images it's ignored.</p> <p>Lower JPEG quality means more compression and lower quality.</p> <p>Higher JPEG quality means less compression and higher quality.</p> <p>Lower G4 density means lighter images.</p> <p>Higher G4 density means darker images.</p> <p>PNG compression setting higher than average (6) will delay processing but often not result in a significantly smaller file size (PNG is a lossless image compression format).</p>

Snippet Property structure description:

Type	Name	Description
BOOL	<i>Enable</i>	<p>Enable the snippet.</p> <p>Valid values for this parameter are:</p> <p>SNIPPET_DISABLE – Snippet disable</p> <p>SNIPPET_ENABLE – Snippet enabled</p> <p>SNIPPET_ENABLE_FOR_DECISION – Snippet enabled for decision (used to decide pocket and/or to decide printing data from the image's information: OCR, ICR, CAR/LAR...)</p>
BOOL	<i>Front</i>	TRUE if the snippet is located on the front of the document, FALSE on the rear.
Snippet	<i>Properties</i>	Snippet structure (see below).

Snippet Structure description:

Type	Name	Description
------	------	-------------

Panini Vision API 4.5.0
Reference Manual Revision 1

UINT	<i>Xposition</i>	X position of the Snippet. The horizontal position reference is the right edge of the front side, and the left edge of the rear side.
UINT	<i>Yposition</i>	Y position of the Snippet. The reference is the bottom edge.
UINT	<i>Width</i>	Snippet Width
UINT	<i>Height</i>	Snippet Height
UINT	<i>Orientation</i>	Snippet Orientation The orientations are: <ul style="list-style-type: none"> - SNIPPET_ORIENTATION_NORMAL The orientation is the same as the document (from the upper-left corner, by horizontal scans, to the lower-right corner) - SNIPPET_ORIENTATION_CCW_90_DEG_ROT The snippet is vertical, and it is scanned from the upper-right corner to the lower-left corner, by vertical lines. - SNIPPET_ORIENTATION_UPSIDE_DOWN The snippet is horizontal, and it is scanned from the lower-right corner to the upper-left corner, by horizontal lines. - SNIPPET_ORIENTATION_CW_90_DEG_ROT The snippet is vertical, and it is scanned from the lower-left corner to the upper-right corner, by vertical lines. - SNIPPET_ORIENTATION_VERTICAL_MIRROR The orientation is the same as the document (from the upper-left corner, by horizontal scans, to the lower-right corner, but stored in memory from the last scan line to the first). - SNIPPET_ORIENTATION_CCW_90_DEG_ROT_VERT_MIR The snippet is vertical, and it is scanned from the upper-right corner to the lower-left corner, by vertical lines, but stored in memory from the last scan line to the first. - SNIPPET_ORIENTATION_UPSIDE_DOWN_VERT_MIRROR The snippet is horizontal, and it is scanned from the lower-right corner to the upper-left corner, by horizontal lines, but stored in memory from the last scan line to the first. - SNIPPET_ORIENTATION_CW_90_DEG_ROT_VERT_MIR The snippet is vertical, and it is scanned from the lower-left corner to the upper-right corner, by vertical lines, but stored in memory from the last scan line to the first.
UINT	<i>Color</i>	Snippet Color SNIPPET_COLOR_GREY_LEVEL for grey level bitmap SNIPPET_COLOR_BLACK_WHITE for monochromatic bitmap SNIPPET_COLOR_COLOR for color bitmap Note that if the image isn't acquired in color mode and a color snippet is required an exception will be fired.
UINT	<i>Compression</i>	Snippet image compression Please refer to the macro SET_SNIPPET_COMPRESSION include in VApiInterface.h.
BOOL	<i>Millimeters</i>	If TRUE the position and the dimensions are mm. If FALSE they are in mils.

NOTE: If X, Y, Width and Height are set to zero, the returned snippet includes the entire image. If the dimensions exceed the document size, the snippet will only include the available part of the image. Snippets are always returned as uncompressed bitmap.

Compatibility between *DeviceParameters* options

The following table shows the incompatibilities between images formats.

Image Format	Incompatible with...
FORMAT_GIV	FORMAT_GIV_DROP_OUT_RED FORMAT_GIV_DROP_OUT_GREEN FORMAT_GIV_DROP_OUT_BLUE
FORMAT_JPEG	-
FORMAT_BMP	-
FORMAT_NATIVE_BMP	FORMAT_BMP_COLOR FORMAT_JPEG_COLOR FORMAT_GIV_DROP_OUT_RED FORMAT_GIV_DROP_OUT_GREEN FORMAT_GIV_DROP_OUT_BLUE FORMAT_JPEG_UV
FORMAT_JPEG_COLOR	FORMAT_GIV_DROP_OUT_RED FORMAT_GIV_DROP_OUT_GREEN FORMAT_GIV_DROP_OUT_BLUE FORMAT_JPEG_UV
FORMAT_BMP_COLOR	FORMAT_GIV_DROP_OUT_RED FORMAT_GIV_DROP_OUT_GREEN FORMAT_GIV_DROP_OUT_BLUE FORMAT_JPEG_UV
FORMAT_GIV_DROP_OUT_RED	FORMAT_GIV FORMAT_NATIVE_BMP FORMAT_BMP_COLOR FORMAT_JPEG_COLOR FORMAT_GIV_DROP_OUT_GREEN FORMAT_GIV_DROP_OUT_BLUE FORMAT_JPEG_UV
FORMAT_GIV_DROP_OUT_GREEN	FORMAT_GIV FORMAT_NATIVE_BMP FORMAT_BMP_COLOR FORMAT_JPEG_COLOR FORMAT_GIV_DROP_OUT_RED FORMAT_GIV_DROP_OUT_BLUE FORMAT_JPEG_UV
FORMAT_GIV_DROP_OUT_BLUE	FORMAT_GIV FORMAT_NATIVE_BMP FORMAT_BMP_COLOR

Panini Vision API 4.5.0
Reference Manual Revision 1

	FORMAT_JPEG_COLOR FORMAT_GIV_DROP_OUT_RED FORMAT_GIV_DROP_OUT_GREEN FORMAT_JPEG_UV
FORMAT_BMP_TRUE_COLOR	FORMAT_BMP_COLOR FORMAT_JPEG_COLOR FORMAT_GIV_DROP_OUT_RED FORMAT_GIV_DROP_OUT_GREEN FORMAT_GIV_DROP_OUT_BLUE FORMAT_JPEG_UV
FORMAT_JPEG_TRUE_COLOR	FORMAT_BMP_COLOR FORMAT_JPEG_COLOR FORMAT_GIV_DROP_OUT_RED FORMAT_GIV_DROP_OUT_GREEN FORMAT_GIV_DROP_OUT_BLUE FORMAT_JPEG_UV
FORMAT_GIV_DOR_TRUE_COLOR	FORMAT_BMP_COLOR FORMAT_JPEG_COLOR FORMAT_GIV_DROP_OUT_RED FORMAT_GIV_DROP_OUT_GREEN FORMAT_GIV_DROP_OUT_BLUE FORMAT_JPEG_UV
FORMAT_GIV_DOG_TRUE_COLOR	FORMAT_BMP_COLOR FORMAT_JPEG_COLOR FORMAT_GIV_DROP_OUT_RED FORMAT_GIV_DROP_OUT_GREEN FORMAT_GIV_DROP_OUT_BLUE FORMAT_JPEG_UV
FORMAT_GIV_DOB_TRUE_COLOR	FORMAT_BMP_COLOR FORMAT_JPEG_COLOR FORMAT_GIV_DROP_OUT_RED FORMAT_GIV_DROP_OUT_GREEN FORMAT_GIV_DROP_OUT_BLUE FORMAT_JPEG_UV
FORMAT_GIV_DROP_OUT_IR	FORMAT_GIV FORMAT_JPEG FORMAT_BMP_COLOR FORMAT_JPEG_COLOR FORMAT_GIV_DROP_OUT_RED FORMAT_GIV_DROP_OUT_GREEN FORMAT_GIV_DROP_OUT_BLUE FORMAT_BMP_TRUE_COLOR FORMAT_JPEG_TRUE_COLOR FORMAT_GIV_DOR_TRUE_COLOR FORMAT_GIV_DOG_TRUE_COLOR FORMAT_GIV_DOB_TRUE_COLOR FORMAT_JPEG_UV

FORMAT_JPEG_UV	FORMAT_NATIVE_BMP FORMAT_BMP_COLOR FORMAT_JPEG_COLOR FORMAT_GIV_DROP_OUT_RED FORMAT_GIV_DROP_OUT_GREEN FORMAT_GIV_DROP_OUT_BLUE FORMAT_BMP_TRUE_COLOR FORMAT_JPEG_TRUE_COLOR FORMAT_GIV_DOR_TRUE_COLOR FORMAT_GIV_DOG_TRUE_COLOR FORMAT_GIV_DOB_TRUE_COLOR FORMAT_GIV_DROP_OUT_IR
----------------	--

Table 1 - Incompatibilities between 1st and 2nd image format

The Smart-Jet is not allowed when:

- True Color image are requested and the MICR+OCR is enabled;
- True Color image are requested and one or more snippets is enabled with the value SNIPPET_ENABLE_FOR_DECISION.

A color snippet is not allowed when the image format is different from a color acquisition (Fast color or True Color).

The Virtual Endorsement is not allowed when:

- the rear image is disabled;
- Smart-jet is based on a rear snippet
- Smart-jet is used with rear Compressed Image for Decision

Image and Snippet retrieved structure Description

On Messages `WMPAR_IMAGES_READY` and `WMPAR_SNIPPETS_READY`, the `LPARAM` will be pointed to a structure containing the images themselves, and some information related with them; for a more simple way of managing this, for both snippets and images the same structure is used. Beware that these structures are allocated internally, and must be released by the application with ***FreeImagesBuffer*** and ***FreeSnippetBuffer*** methods.

The following is the description of ***ImagesStruct*** structure:

- **DWORD DocNumber** : the document number.
- **CompressedImage * Images** : the pointer to a vector of `CompressedImage` structure: 5 elements for images (in order 1st front image, 2nd front image, 1st rear image, 2nd rear image and multi-page TIFF) and 10 for snippets.
- **BOOL SnippetFront** : only for snippets (TRUE if the snippet is on the front of the document).

The `CompressedImage` structure has two members:

- **BYTE * pBuffer** : the buffer pointer.
- **int BufferLen** : the length of the buffer for the compressed images or snippets.

If an image is not requested the corresponding buffer pointer is `NULL` and the buffer length is 0.

Firmware Parameters Description

These parameters are sent to the device with the function call ***SetSorterParameter***, and are retrieved from it with ***GetSorterParameter***. When the device connects the default values of the sorter parameters are restored.

The parameters that are available are as following:

- ID = 0 Double-Feeding Detection Enabling (0 – 1, default is 0).
Enable the Double-Feeding Detection.
- ID = 1 Power for Double-Feed Detection (2 – n, default is 5).
Set the power value of the photocell detecting the Double-Feeding. The maximum value of this parameter depends on the manufacturing calibration.
- ID = 2 Delay For Double-Feed Detection (10 mm – 150 mm from leading edge, default is 50).
Set the delay from leading edge where the Double-Feeding is tested.
- ID = 3 Confidence for Double-Feed Detection (2 – 10, default is 7).
Higher indicates greater reliability of detection.
- ID = 4 Hole Filter Length (5 mm – 10 mm, default is 5 mm).
To avoid holes in the paper indicating document end.

- ID = 5 Delay To Start Endorsement (0 mm – 220 mm, default is 0 mm).
Set the delay in print starting, referred to the leading edge of the document.
- ID = 6 Max symbols in MICR code-line (10 – 80, default is 80).
This parameter can reduce the MICR reading length. The value represents the maximum number of MICR symbols accepted.
- ID = 7 Min document length (80 – 240 mm, default is 80).
This value is the minimum document length accepted.
- ID = 8 Max document length (80 – 240 mm, default is 240).
This value is the maximum document length accepted.
- ID = 9 Enable Full-Pocket detection (0-1, default is 0)
Enables the Full Pocket detection for a My Vision X endowed with 2 pocket. When a Full Pocket condition is detected, the machine doesn't stop and the application receives a `DEVICE_ERR_FULL_POCKET` exception. It's an application responsibility to manage this situation (*Stopfeeding* call, switch the destination pocket, User Interface warning...).

Note that the default values for the Double-Feeding detection are optimized for standard documents and are calibrated during the manufacturing process.

NOTE: See Appendixes for specific FW parameters information that applies only to that particular engine.

Pockets Handling

This API version is able to manage a device endowed with 2 pockets. With this kind of machine the application can decide the destination pocket of a document. Basically there are two manners to decide the document's destination:

1. using the MICR information;
2. Using the image (OCR, ICR, CAR/LAR...) information.

The pocket is decided by the application calling the API function *SetPocket*. This function has to be called during the `SET_ITEM_OUTPUT` message.

To decide the pocket using the MICR information, the application has to enable the MICR option. As already explained in this document, when the application receives the `SET_ITEM_OUTPUT` message, the application has to call the *SetPocket* function to decide the document's destination pocket. The `SET_ITEM_OUTPUT` is always sent after the `MICR_AVAILABLE` message. Thus, during the `SET_ITEM_OUTPUT` message, the application already has the MICR information to decide the pocket.

To decide the pocket using the image information, the application has to enable at least one snippet *for decision*. Both the front and the rear side can be used to extract the snippet for decision. The enable *for decision* means that the pocket is decided after the image acquisition, when the snippet is sent to the application. Compressed images cannot be used to decide the

pocket. To enable the snippet for decision you have to set the **Enable** field of the SNIPPET_STRUCTURES to the **SNIPPET_ENABLE_FOR_DECISION** value.

Enabling this kind of snippet, the API sends the SET_ITEM_OUTPUT message after the SNIPPET_READY message. Thus, during the SET_ITEM_OUTPUT message, the application already has the image information (OCR, ICR, CAR/LAR....) to decide the pocket for the document.

When a snippet is requested with ENABLE only, the sequence of the document messages is:

- NEW_DOC;
- MICR;
- **SET_ITEM_OUTPUT;**
- IN_POCKET;
- **SNIPPET_READY;**
- IMAGE_READY;
- DOC_COMPLETE.

Using the ENABLE_FOR_DECISION value the sequence will be:

- NEW_DOC;
- MICR;
- **SNIPPET_READY;**
- **SET_ITEM_OUTPUT;**
- IN_POCKET;
- IMAGE_READY;
- DOC_COMPLETE.

It's important to notice the position of SET_ITEM_OUTPUT and SNIPPET_READY messages. In the first case the snippet message is considered as a down-stream message (not used for decision) and is signaled after pocket decision. In the second case it is an up-stream message (used for decision) and is signaled before the SET_ITEM_OUTPUT that is the message where the application decides the document's pocket.

The MICR-based decision permits to the application to maintain the right DPM performance. The performance could be affected by a very long delay of the application before calling *SetPocket ()*. The machine cannot feed a new document if the application doesn't call the *SetPocket ()* function for the previous one. Then, later is the call and more is the reduction of the DPM performance.

The 2 pocket machine introduces the reverse gear of the document in the track. In some cases, the machine needs to retreat the document in the track to complete the operations. If the application takes a long time to decide the pocket, the machine will stop the document just after the image capture device waiting for the decision. If the destination is the default pocket, the machine restarts track moving forward the document in the pocket. If the destination is the second pocket, the machine makes a reverse gear of the document, stop the motor when the leading edge is positioned before the mechanical switch, then open the switch and then restart the motor to put the document in the pocket.

The Image-based decision usually cannot maintain the right DPM (except for MVX 30 DPM). For example, when the pocket is selected by the image information and the destination pocket is the second, the device have always to retreat the leading edge of the document, before the mechanical switch position, and then restart the track to introduce the document in the pocket.

Smart Jet

Smart Jet is a more flexible manner to manage the Printing operations. When the Printer is used like as an up-stream device, the application has to decide the information to print on the document before the document is fed.

Instead Smart Jet set the Printer as a down-stream device. This means that the application can decide the printing information based on the MICR or on the Image information (OCR, ICR, CAR/LAR...).

To enable this kind of printer the application has to add a new flag (PRINTER_ENABLE_SMART_JET) inside the *bPrintEnable* field of the *DeviceParameters* structure.

When Smart-Jet is enabled, the application has to call the *SendPrinterData* function only during the SET_ITEM_OUTPUT event, before the *SetPocket* call.

Example

```
DWORD DocID;
char ApiErrorString[200];

case WMPAR_SORTER_SET_ITEM_OUTPUT:
    DocID = (DWORD) LPARAM;

    // When Smart-Jet enabled this function has to be called
    // only here, before SetPocket() call.
    SendPrinterData(DeviceID, ... )
    SetPocket( DeviceID, DocID, 1 );
    break;
```

To decide the printing data using the MICR the application has to enable the MICR option.

To decide the printing data using the Image the application has to enable has to enable at least one snippet for decision (explained in Pocket Handling section). Both the front and the rear side of document can be used to extract the snippet for decision (like for a 2 pocket machine).

The Smart Jet could affect the DPM performance, especially when based on image information. The Smart Printer could require (surely in case of printing from image) a reverse gear of the document. This happens when the position of the document is after the printing position decided by the application. In this case the machine has to retreat the document to recover the right position and start the printing operation.

There is a mechanical limit for the reverse gear of the document. When the application takes a very long time to decide the pocket, the machine stops the document in the track after the end of the MICR signal acquisition. When the printer data are sent to the My Vision X, the firmware retreats the document in the track to recover the right printing position, stop the

track, and the restart the motor to make the printing and conclude the document processing. If the printing position could not be recovered, the firmware signals an ERR_PRINTER error.

True Color

This API has the capabilities of capture TRUE COLOR images. TRUE COLOR images are different from the FAST COLOR images because the image data don't undergo any pre-processing between the acquisition process and the compression process.

FAST COLOR images are captured at the nominal track speed. This means that the acquisition process isn't a real true color acquisition, but the compression process creates a compressed true color image and the machine can maintain the DPM throughput.

True Color images are captured at a lower track speed. So the acquisition is a real true color acquisition, but the DPM performance is reduced.

The TRUE COLOUR option could require a reverse gear of the track. As the MICR and the Printing needs the nominal track speed, if they are enabled, the TRUE COLOUR acquisition cannot slow down the track until they have terminated their operations. When they (MICR and printer) have finished their operations, if the document is in front of the image capture sensor, the device have to move back the document and restart the track to the right speed (one 3rd of the nominal) to acquire the TRUE COLOUR image.

To capture TRUE COLOR images the application has to enable the new image format options.

To capture TRUE COLOR images the application has to enable the new image format options **FORMAT_BMP_TRUE_COLOR** and **FORMAT_JPEG_TRUE_COLOR**.

The TRUE COLOR permits you to obtain a finer image quality compared with the FAST COLOR option, at a lower throughput.

In addition it is possible to apply a Color Drop-out process based on the TRUE COLOR acquisition. This option is enabled by the values **FORMAT_GIV_DOR_TRUE_COLOR** (Red drop out), **FORMAT_GIV_DOG_TRUE_COLOR** (Green drop out) and **FORMAT_GIV_DOB_TRUE_COLOR** (Blue drop out).

UV Images

This API has the capabilities of capture images using Ultraviolet light. Of course the device has to mount a UV-capable images sensor.

UV images are captured at the nominal track speed and the device can maintain the DPM throughput.

To capture UV images the application has to enable one of the following image format options: **FORMAT_JPEG_UV**, **FORMAT_JPEG_UV_NEGATIVE** (gray-level JPEG acquired with UV light and inverted), **FORMAT_GIV_UV** (G4 TIFF) or **FORMAT_GIV_UV_NEGATIVE** (G4 TIFF image acquired by UV light and inverted before binarization).

The UV option makes the user capable of detect UV ink anti-fraud patterns.

MICR+OCR

This API introduces the capabilities to read the MICR code-line using the magnetic head signal and the image information. This means that an OCR algorithm is used to create the best MICR result. This option can strongly reduce the rejects rate of a device.

This option doesn't change the behaviour of your applications, because they always receive the MICR_AVAILABLE message with MICR code-line. Internally the API makes a further process "merging" the MICR result with the OCR one.

NOTE: This option could slightly reduce the DPM throughput.

To enable it, the application has to set one of MICR_FONT_E13B_OCR, MICR_FONT_CMC7_OCR and MICR_FONT_AUTO_OCR options in the *nMICRFont* field of the *DeviceParameters* structure.

New behavior (API 4.5+)

Starting from API 4.5.0 a new set of requirements have been implemented for MICR+OCR:

- All device engines will use +OCR by default, meaning that all current applications having selected E13B only or E13B+OCR will have the +OCR enabled.
- The OCR DLL will always be copied during installations. If the OCR DLL is missing the device engine will refuse to start.
- +OCR can be disabled without deleting files or setting registry entries, but a new API call will be used to disable +OCR and must be called before *SetDeviceParameters()* – see below
- Upgrade to OpenCV 2.4.6 for all OCR libraries

To disable/Enable the MICR+OCR enforcing (which by default is enabled on API 4.5+ for all Engines), please use the following new API call:

BOOL *SetMicrOcrFontOverride* (DWORD DeviceID, BOOL bEnabled)

Valid in State : *DeviceChangeParameters*

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

BOOL **bEnabled** - FALSE (disable) / TRUE (enable)

Return Value : TRUE if successful otherwise FALSE (see logs for errors)

NOTE: This function must be called before the *SetDeviceParameters()*

Example

```
DeviceParameters DevPar;  
char ApiErrorString[200];  
  
// to call DeviceChangeParameters method you must be in  
// ChangeParameters state  
if( ChangeParameters( m_DeviceID ) )  
{  
    // Set the MICR+OCR font override *before* calling the  
    // SetDeviceParameters()  
    if( m_DevPar.bMICREnable )  
    {  
        SetMicrOcrFontOverride( m_DeviceID, bMicrOcrFontOverride );  
    }  
  
    if( !SetDeviceParameters( m_DeviceID, DevPar ) )  
    {  
        // Report error  
        GetApiErrorString( ApiErrorString, 200 );  
        MessageBox(...);  
        ...  
    }  
    ...  
}
```

Reverse gear

Enabling some *DeviceParameters* option the machine could need to implement a reverse gear of the document in the track. This kind of behaviour is present only on machine endowed with 2 pockets or with mechanical reversibility.

There are essentially 4 cases:

1. ***Printer with Trailing edge alignment.***

If the printing data are Trailing edge aligned, the machine needs to know the length of the document to decide where is the printer start position. When the trailing edge leaves the first photocell sensor, the machine computes the print position. If this position is beyond the printer head, the machine has to make a reverse gear of the document to bring the document to the right position. This operation is very fast with minimum lack of DPM performance.

2. ***Smart Printer.***

The Smart Printer could require (surely in case of printing from image) a reverse gear of the document. This happens when the position of the document is beyond the printing position decided by the application. In this case the machine has to retreat the document to recover the right position and start the printing operation.

3. ***True Color.***

The reverse gear is necessary when the machine has to wait for the end of MICR and/or Printer operations. As these two peripherals need the nominal track speed, the machine has to terminate their operations and, if the leading edge is in front of the image sensor, stop the track, do a reverse gear of the document, restart the track setting the right speed to capture the True Color images.

4. ***Pocket decision.***

The reverse gear is requested when the leading edge of the document is beyond the mechanical switch, inside the default pocket, and the application chooses the second

pocket for that document. If the image capture is enabled, the machine has to wait for the end of the image operations, then stops the track, makes a reverse gear of the document and finally put the document in the second pocket.

This kind of reverse gear could happen combined. You can see the 1 combined with 3, or the 2 with 4, the 3 with 4, etc...

The reverse gear affects the DPM performance depending on the option enabled and the application behaviour.

Virtual Endorsement

The Virtual endorsement is the capability of the device to create a graphical printing overlapped to the rear image of the processed document. Instead of using the physical printing head the programmer can decide to generate the endorsement data by software. Using the virtual endorsement the paper won't be inked at all: in fact this capability excludes the printing head. The virtual endorsement doesn't need an IDCard license and it is available even if the printer is disabled. It has been developed in order to follow the ANSI standard **X9.100-111-2004**. It can be used with all the image formats (except for the native bitmap image).

Virtual endorsement defines up to eight different areas on the document image. They are the followings:

- Payee area
- Bank Of First Deposit (BOFD) area
- Transit area (subsequent banks)
- Custom area #1
- Custom area #2
- Custom area #3
- Custom area #4
- Custom area #5

The first three areas are specifically defined following the ANSI X9 standard for US Checks. The others five are not defined and could be used to create areas that don't follow the ANSI standard.

The following figure shows the standard definition of the firsts three areas:

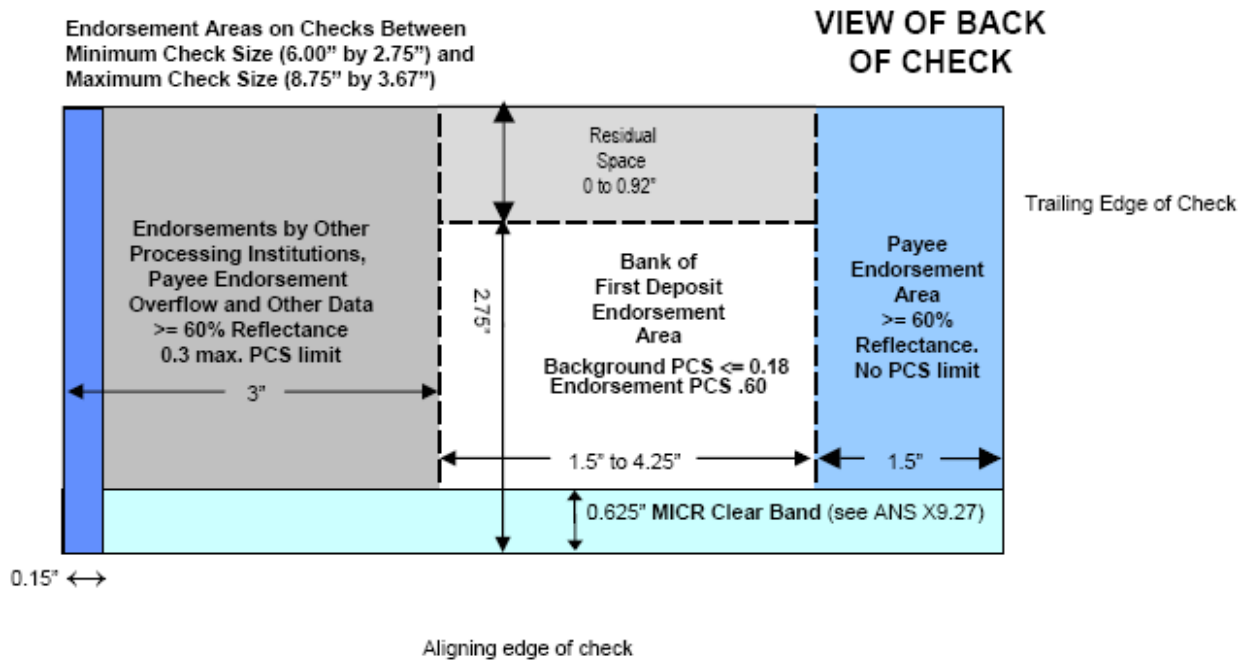


Figure 1 – ANSI layout for back of check

For each area the application can print a multi line text and/or a bitmap. If the text and the bitmap extension exceed the area boundaries, they are cut within them.

Each area is defined by the following parameters:

- AREA RECTANGLE

It defines the position of the area referred to the aligning edge (bottom edge of the image), the leading edge (left edge of the image) and the trailing edge (right edge of the image). It's defined as a Win32 RECT structure. The values can be defined as mils (thousands of inch) or millimeters.

The bottom and the top fields are referred to the aligning edge of the document. They represent a distance from the aligning edge of the document.

Bottom defines the lower edge of the area and Top is the upper part. Bottom has to be lower than Top. If Top exceeds the height of the document it's automatically limited to the height.

The Left field sets the distance from the Leading edge of the document.

The Right field sets the distance from the Trailing edge of the document.

The following table shows the default values of each endorsement area:

Area ID	Bottom	Top	Right	Left	Unit
Payee	625	4250	-1500	0	mils
BOFD	625	2750	1500	3000	mils

Panini Vision API 4.5.0
Reference Manual Revision 1

Transit	625	4250	0	-3000	mils
Custom #1	0	0	0	0	mils
Custom #2	0	0	0	0	mils
Custom #3	0	0	0	0	mils
Custom #4	0	0	0	0	mils
Custom #5	0	0	0	0	mils

The Payee area is defined at a distance of 1500 mils from the trailing edge. It's independent from the document length. This is the reason why the left field is not used (set to 0) and the Right value is negative.

The same is for the Transit area, but it's defined from the leading edge. The left field is negative and the right field is not used.

The BOFD area instead has a variable length dependent on the document length. It defines the distance from the vertical borders.

Every standard area has a bottom edge that skips the MICR clear band (625 mils). The top edge is defined by the ANSI specifications.

- UNIT

This parameter defines the unit of length set in the rectangle structure. They can be expressed in mils (thousandth of an inch) or millimeter. The default setting is mils.

- FONT

The font of the text is defined as a Win32 LOGFONT structure. The default font for every area is Arial, the size is 10 and the weight is 600 (FW_SEMIBOLD). The font size has to be defined greater than 0 (negative isn't accepted). The size is expressed in points (1/72 of an inch). The font size is automatically scaled to the image resolution.

- COLOR

The color of the text is defined as a Win32 COLORREF type. The default settings is black (RGB(0,0,0)). The color is automatically converted to the right color-space of the image (True Color, Gray levels and Black&White)

- FLAGS

There is available a set of flags in order to define the horizontal text alignment and the word wrap. The default setting is center aligned and word wrap turned off. The flags are defined in the VApiInterface.h file. The flags bits-field is defined as Win32 DWORD type.

The followings figures show the result of printing when word wrap is off or on:

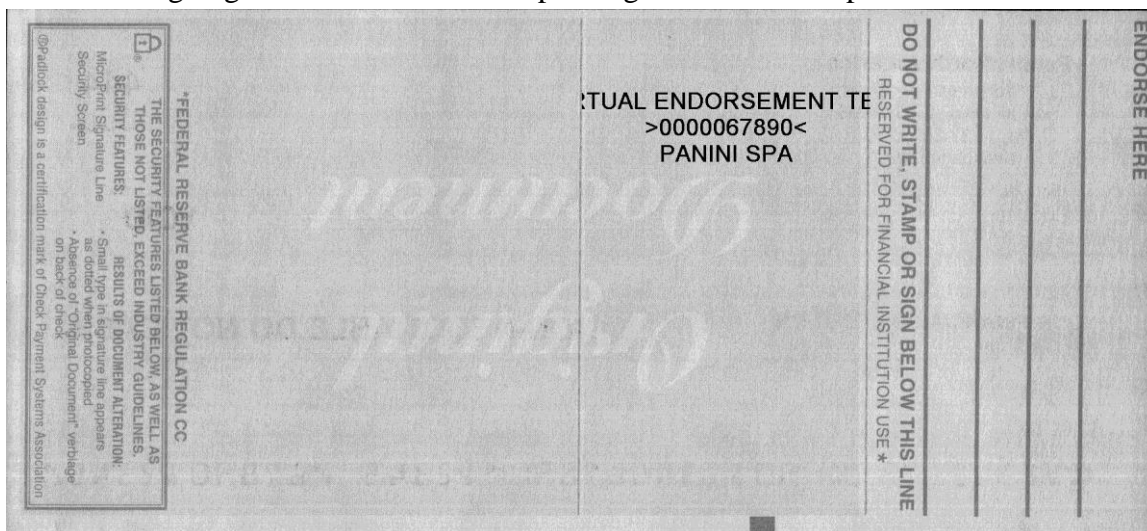


Figure 2 – Word wrap off

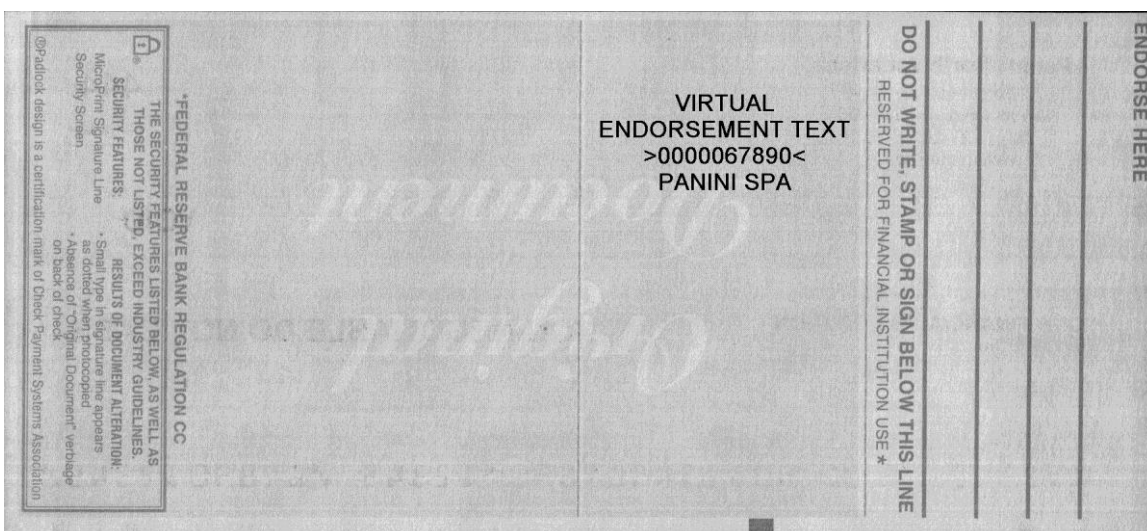


Figure 3 – Word wrap on

- ROTATION

There are four possible counterclockwise rotations. They are 0, 90, 180 and 270 degrees. The default setting is 0 degree except for the Payee area where it is set to 270 degrees.

The application has to call the functions `SetVirtualEndorsementAreaProperty` or `GetVirtualEndorsementAreaProperty` in order to set or get one parameter of an area.

Example – How to set the area, the font, the color, the rotation and the flags of the BOFD area

```
ChangeParameters( DeviceID );

// Rectangle
RECT Area = {1000,4250,2000,625};
SetVirtualEndorsementAreaProperty( DeviceID, ENDORSEMENT_AREA_BOFD,
```

Panini Vision API 4.5.0 Reference Manual Revision 1

```
ENDORSEMENT_PARAM_RECT, (LPVOID)&Area );

// Font
LOGFONT font;
memset( &font, 0, sizeof(font) );

GetVirtualEndorsementAreaProperty( DeviceID, ENDORSEMENT_AREA_BOFD,
                                   ENDORSEMENT_PARAM_FONT, (LPVOID)& font );
font.lfHeight = 14;
font.lfWeight = FW_BOLD;
strcpy(font.lfFaceName , "Courier New");
SetVirtualEndorsementAreaProperty( DeviceID, ENDORSEMENT_AREA_BOFD,
                                   ENDORSEMENT_PARAM_FONT, (LPVOID)&font );

// Color
COLORREF Color = RGB(128,0,0); // Dark red
SetVirtualEndorsementAreaProperty( DeviceID, ENDORSEMENT_AREA_BOFD,
                                   ENDORSEMENT_PARAM_COLOR, (LPVOID)&Color );

// Rotation
DWORD Rot = ENDORSEMENT_ROTATE_270;
SetVirtualEndorsementAreaProperty( DeviceID, ENDORSEMENT_AREA_BOFD,
                                   ENDORSEMENT_PARAM_ROTATION, (LPVOID)&Rot );

// Flags - Left horizontal alignment and word-wrap on
DWORD Flags = (ENDORSEMENT_FLAG_LEFT | ENDORSEMENT_FLAG_WORD_WRAP);
SetVirtualEndorsementAreaProperty( DeviceID, ENDORSEMENT_AREA_BOFD,
                                   ENDORSEMENT_PARAM_FLAGS, (LPVOID)&Flags );
```

In order to enable the virtual endorsement the application has to set a flag in the `bPrintEnable` field.

If the Virtual Endorsement flag is enabled the parameters regarding the printing quality and the alignment are ignored.

The Smart Jet is available when Virtual endorsement is requested.

Example: How to enable the Virtual Endorsement

```
DeviceParameters params;
BOOL SmartJetEnabled = TRUE;

params.bPrintEnable = (PRINTER_ENABLE_LEADING_EDGE | PRINTER_VIRTUAL_ENABLE);

if( SmartJetEnabled )
    params.bPrintEnable |= PRINTER_ENABLE_SMART_JET;
```

Once the Virtual endorsement is enabled the application has to the function `SetVirtualEndorsement` in order to set the text and a bitmap of a specific area. The call sequence of this function is exactly the same requested for the `SendPrinterData` function. Since the physical printer is not available when the Virtual is enabled the application has to `SetVirtualEndorsement` instead of `SendPrinterData`.

A further capability is that the application is able to call the `SetVirtualEndorsement` multiple times in order to print into different areas for each documents processed.

Example: How to print a text into BOFD and a BMP into Payee on the same document

```
case WMPAR_SORTER_SET_ITEM_OUTPUT:
...
SetVirtualEndorsement( DeviceID, ENDORSEMENT_AREA_BOFD, "BOFD text",
                        NULL, 0 );
SetVirtualEndorsement( DeviceID, ENDORSEMENT_AREA_PAYEE, NULL,
                        "Payee.bmp", ENDORSEMENT_BMP_FILE );
```

IQA

Panini IQA has been implemented in order to facilitate FSTC Image quality certification testing on Panini Devices. The library is based on the FSTC Specifications and requirements for the image quality analysis. The testing process is intended to give a validation to the reliability of the image that will be used for processing.

The IQA processing parameters are enabled and defined by the application.

If enabled, the tests are executed inside the Panini API. At the end of the process, the application can retrieve the compressed images and the results of the tests executed on them.

The IQA Test System is based on sixteen tests. Here is a brief description of all the tests:

1. **Undersize Image** – Detects images with dimensions that are too low
2. **Folded or Torn Document Corner** – Detects the presence of folded or torn corners on the document
3. **Folded or Torn Document Edge** – Detects the presence of folded or torn edges on the document
4. **Document Framing Error** – Detects the presence of black lines at the borders of the image
5. **Excessive Document Skew** – Detects excessive skewing on the document
6. **Oversize Image** – Detects the images which have dimensions higher than the acceptable threshold
7. **Piggyback Document** – Detects the presence of more than one document in the image
8. **Image Too Light** – Detects images that are too light
9. **Image Too Dark** – Detects images that are too dark
10. **Horizontal Streaks in Image** – Detects the presence of horizontal streaks in the image
11. **Below Minimum Compressed Image Size** – Detects if the dimension of the file is too low

- 12. **Above Maximum Compressed Image Size** – Detects if the dimension of the file is too high
- 13. **Excessive Spot Noise in Image** (Only BW) – Detects the presence of spot noise in the image
- 14. **Front And Rear Image Dimension Mismatch** – Detects the differences between the front and the rear image dimensions
- 15. **Carbon Strip Detected** – Detects the presence of a carbon strip
- 16. **Image Out Of Focus** (Only Grey Level)– Detects images that are out of focus

Each test can be enabled or disabled and a set of parameters is used to define the defect detection thresholds.

Each test returns one the following results:

- 0 : the condition has not been tested;
- 1 : the condition is tested, and the defect is present;
- 2 : the condition is tested, and the defect is not present.

The IQA interface functions are:

- SetIQAParameters;
- GetIQAResults;
- GetIQAValues;
- GetIQALastError;
- GetIQALastErrorString.

These functions are defined in VApiInterface.h.

The IQA parameters are defined in VApiIQAInterface.h

Extended parameters

VisionAPI implements a set of optional parameters to run additional processing to the document data in order to help customer's applications to handle particular processing conditions (documents with defects, operator without experience, not jogged document...).

These additional parameters are disabled by default.

The current release supports these parameters for the VisionX/MyVisionX device.

The "Extended parameters" interface functions are:

- SetExtendedParameter;
- GetExtendedParameter.

These functions are defined in VApiInterface.h.

The extended parameters are defined in VApiExtendedParameters.h.

Here is a description of each parameter:

- **"Ignore cropping error"**

Name: EP_IP_IGNORE_CROPPING_ERROR

Type: UINT32

Min: 0 (Disabled)

Max: 1 (Enabled)

Default: 0

Requirements: Image capture enabled

Description:

This parameter forces the API to ignore image cropping error when document with folded or torn corner are processed. The API avoid to stop the processing with a "Compression error" and the application has to know that the image must be checked to ensure the document has been correctly captured.

- **"Auto orientation"**

Name: EP_IP_ENABLE_AUTO_ORIENTATION

Type: UINT32

Min: 0 (Disabled)

Max: 1 (Enabled)

Default: 0

Requirements: MicrFont set to E13B+OCR or CMC7+OCR

Description:

This parameter enables an image pre-processing task that is able to identify the document orientation and to correct it if necessary. When the document is inserted in the feeder upside-down, the API detects the wrong orientation condition and returns to the application an image with the right orientation.

The processing is based on the image capture information.

This parameter supports the following image formats: MICR+OCR is selected (E13B+OCR or CMC7+OCR). The parameter cannot be used with AUTO+OCR

options.

- **“Auto de-skew”**

Name: EP_IP_ENABLE_AUTO_DESKEW

Type: UINT32

Min: 0 (Disabled)

Max: 1 (Enabled)

Default: 0

Requirements: Image capture enabled with image format (G4, JPEG and BMP gray level)

Description:

This parameter enables an image pre-processing task that is able to identify the document skew and to correct it if necessary. When the operator inserts the documents in the feeder without jogging them, some documents could be fed with a certain skew. When the API detects a skew greater than 0.5 degrees, it rotates (de-skew) the image in order to return to the application a correctly aligned image.

This parameter supports the following image formats: FORMAT_GIV, FORMAT_JPEG, FORMAT_BMP. The other image formats cannot be used with this option.

- **“Allow MICR with OCR only”**

Name: EP_MICR_ALLOW_OCR_ONLY

Type: UINT32

Min: 0 (Disabled)

Max: 1 (Enabled)

Default: 0

Requirements: One of the parameters “Auto-Orientation” and “Auto-Deskew” enabled.

Description:

This parameter enables the possibility to decode the entire MICR codeline via OCR (in case of excessive skew or flipped document).

This behaviour changes the way VisionAPI implements the MICR+OCR. Normally the OCR result is used to correct the MICR rejects. This parameter allows the application to receive the OCR results only when the MICR information is wasted due to a wrong document position (upside-down or skew).

It requires that one parameter among “Auto-Orientation” and “Auto-Deskew” is enabled. It also requires that one of the MICR+OCR font is selected.

When a document with wrong orientation is detected, if enabled, the API returns the OCR result of the MICR codeline.

- **“Minimum MICR length”**

Name: EP_MICR_MIN_LENGTH

Type: UINT32

Min: 0

Max: 200

Default: 20

Requirements: One of the parameters “Auto-Orientation” and “Auto-Deskew”

enabled. MICR+OCR font selected.

Description:

This parameter sets a minimum number of characters expected from the MICR result.

This parameter aid to detect a wrong orientation or skew.

When the number of characters is less than this value, the API returns the MICR codeline recognition returned by the OCR.

This parameter is used only when “Auto orientation” or “Auto-Deskew” are enabled.

- **“MICR Maximum acceptable skew”**

Name: EP_MICR_MAX_ACCEPTABLE_SKEW

Type: UINT32

Min: 10

Max: 50

Default: 15

Requirements: Parameters “Auto-Deskew” and “MICR allow with OCR only” enabled. MICR+OCR font selected.

Description:

This parameter sets a maximum acceptable document skew to return the MICR result.

When the skew is greater than this value, the API returns the MICR codeline recognition returned by the OCR.

It is expressed in “tenth of degrees”. Example: 15 means 1.5 degrees.

This parameter is used only when “Auto-Deskew” and “MICR allow with OCR only” are enabled.

- **“Machine tapes processing”**

Name: EP_ENABLE_MACHINE_TAPE

Type: UINT32

Min: 0 (Disabled)

Max: 1 (Enabled)

Default: 0

Requirements: When enabled the application has to set the followings device parameters values:

- MICR disabled
- Printer disabled
- Rear image disabled
- Front image format: FORMAT_JPEG or FORMAT_GIV
- No limit on the image resolution

Description:

This parameter enables the processing of machine tapes document up to 10 feet length.

- **“Machine tapes maximum length”**

Name: EP_MACHINE_TAPE_MAX_LEN

Type: UINT32

Min: 80 (mm)

Max: 3048 (mm)

Default: 1524 mm

Requirements: “Machine tapes processing” enabled.

Description:

This parameter sets the maximum length of a machine tape document. It might be used to improve jam condition detection on this kind of document. The default value is 1524 mm (5 feet).

- **“Front Snippet resolution”**

Name: EP_SNIPPET_FRONT_RESOLUTION

Type: UINT32

Min: 100

Max: 600

Default: 200

Requirements: None

Description:

This parameter sets the front side snippets resolution in DPI. The admissible values are 100, 200, 300 or 600 (if the device type supports such resolution)

- **“Rear Snippet resolution”**

Name: EP_SNIPPET_REAR_RESOLUTION

Type: UINT32

Min: 100

Max: 600

Default: 200

Requirements: None

Description:

This parameter sets the rear side snippets resolution in DPI. The admissible values are 100, 200, 300 or 600 (if the device type supports such resolution)

- **“UV image Brightness adjustment”**

Name: EP_IP_UV_BRIGHTNESS

Type: UINT32

Min: 0

Max: 200

Default: 100

Requirements: None

Description:

This parameter is used to adjust the UV image brightness. Values are in percentage, in the range 0 to 200, where 100 will preserve original brightness (the range 0-99 will apply a brightness decrease, while 101-200 a brightness increase).

- **“UV image Contrast adjustment”**

Name: EP_IP_UV_CONTRAST

Type: UINT32

Min: 0

Max: 200

Default: 100

Requirements: None

Description:

This parameter is used to adjust the UV image contrast. Values are in percentage, in the range 0 to 200, where 100 will preserve original contrast (the range 0-99 will apply a contrast decrease, while 101-200 a contrast increase).

- **“UV image background removal”**

Name: EP_IP_UV_BACKGROUND_REMOVAL

Type: INT32

Min: 0

Max: 100

Default: 0 (disabled)

Requirements: None

Description:

This parameter sets fraction of the dark areas to suppress. Values are in percentage, in the range 1 to 100. Set to 0xFFFFFFFF to use the mean image or to 0 to disable the removal.

Error Handling

The error query is available with specific functions, for each of them there are two versions, one retrieving the Error Code, the other returning the string that describes the error.

The types of the errors that can occur are:

- **USB Errors** that depends on communication problems between the device and the PC through the USB connection. Details on the error can be retrieve with the function *GetUsbError* and *GetUsbErrorString*.
- **Device Errors** typically depend on mechanical or peripheral problems. Their related functions are *GetDeviceError* and *GetDeviceErrorString*. A Device Error, could be due to a USB Error, so the nature of the cause will require further investigation.
- **API Errors** are errors concerning software problems, use *GetApiError* and *GetApiErrorString* to get details on the error which occurred. An API Error can be a USB Error or a Device Error.

When a function call returns FALSE, it is an **API Error** (that could be a USB or a Device error), so the function *GetApiError* or *GetApiErrorString* must be used for first analysis.

When the message WMPAR_SORTER_EXCEPTION is received, it is a **Device Error** and so *GetDeviceError* or *GetDeviceErrorString* must be used. In addition the LPARAM of the message contains in the lower 2 bytes the error code of the error that occurred, in the higher 2 bytes the number of the current document (if it is significant, otherwise it is set to zero).

For further analysis, more functions are available like *GetSorterError* and *GetSorterErrorString*. They obtain the error code directly from the firmware (this can be useful, for example, to know the exact point where a jam occurred). See the function definitions in the next chapter for more details.

USB Errors

USB ERROR LABEL	ERROR CODE	DESCRIPTION
USB_ERR_NONE	0	No error
USB_ERR_INVALID_DEVICE_NAME	1	Internal error
USB_ERR_OPEN_DRIVER	2	Internal error
USB_ERR_CLOSE_DRIVER	3	Internal error
USB_ERR_GET_PIPE_INFO	4	Internal error
USB_ERR_GET_DEVICE_DESCRIPTOR	5	Internal error
USB_ERR_DEVICE_IO_CONTROL	6	Internal error
USB_ERR_WRITE_BULK	7	Internal error
USB_ERR_READ_BULK	8	Internal error
USB_ERR_WRITE_CONTROL	9	Internal error
USB_ERR_READ_CONTROL	10	Internal error
USB_ERR_GET_LINK_STATE	11	Internal error
USB_ERR_SEND_LINK_MESSAGE	12	Internal error
USB_ERR_RECEIVE_LINK_MESSAGE	13	Internal error
USB_ERR_PROGRAM_FPGA	14	Internal error
USB_ERR_CHECK_FPGA	15	Internal error
USB_ERR_WRITE_E2PROM	16	Internal error
USB_ERR_READ_E2PROM	17	Internal error
USB_ERR_SET_TIME_LEDS	18	Internal error
USB_ERR_SET_GAINS	19	Internal error
USB_ERR_SET_OFFSETS	20	Internal error
USB_ERR_WRITE_DMA_BULK	21	Internal error
USB_ERR_READ_DMA_BULK_PARAM	22	Internal error
USB_ERR_READ_DMA_BULK_SETUP	23	Internal error
USB_ERR_READ_DMA_BULK_PHASE1	24	Internal error
USB_ERR_READ_DMA_BULK_PHASE2	25	Internal error
USB_ERR_READ_DMA_BULK_CLEAR	26	Internal error
USB_ERR_READ_ID_CARD	27	Internal error
USB_ERR_WRITE_ID_CARD	28	Internal error
USB_ERR_SET_DPM	29	Internal error
USB_INVALID_PARAM	30	Internal error
USB_INVALID_DEVICE	31	Internal error
USB_SYSTEM_ERROR	32	Internal error

Device Errors

DEVICE ERROR LABEL	DEVICE ERROR CODE	DESCRIPTION
DEVICE_ERR_NONE	0	No error
DEVICE_ERR_UNKNOWN_SORTER_ERROR	1	Unknown sorter error
DEVICE_ERR_USB	2	USB error
DEVICE_ERR_WAIT_FEED_DONE_TIMEOUT	4	Timeout waiting for feed done event
DEVICE_ERR_WAIT_LAST_DOC_POCKETED_TIMEOUT	5	Timeout waiting for last document in pocket
DEVICE_ERR_READ_SORTER_STATUS	6	Sorter status read failed
DEVICE_ERR_SORTER_ERROR_PENDING	7	Sorter error pending
DEVICE_ERR_COMMUNICATION_FAILURE	8	Communication failure reported by the sorter
DEVICE_ERR_FEED_FAILURE	9	Document feeding failed
DEVICE_ERR_FULL_POCKET	10	Full pocket detected
DEVICE_ERR_SAFETY	11	The covers safety switch are locking the device
DEVICE_ERR_GET_LAST_DOC_POCKETED_ID	12	Failure reading last document pocketed ID
DEVICE_ERR_FEED_DOCUMENT	13	Failure sending Feed document command
DEVICE_ERR_TRACK_NOT_CLEAR	14	Feeding command failure due to a document in the track
DEVICE_ERR_SET_POCKET	15	SetPocket failed
DEVICE_ERR_GET_LAST_DOC_ID	16	Last doc ID reading failed
DEVICE_ERR_READ_MICR_SIZE	17	MICR size reading failed
DEVICE_ERR_READ_MICR_WAVEFORM_CHUNK	18	MICR data transfer failed
DEVICE_ERR_DIFFERENT_MICR_SIZE	19	MICR internal error
DEVICE_ERR_MICR_BUFFER_OVERFLOW	20	MICR internal error
DEVICE_ERR_MICR_READING	21	MICR reading failed
DEVICE_ERR_ACQUISITION_FAILED	22	Image acquisition failed
DEVICE_ERR_COMPRESSION_ERR	23	The image compression process failed
DEVICE_ERR_READ_ERROR	24	Read sorter error failed
DEVICE_ERR_CREATE_COMPENSATION_TABLES	25	Error creating image compensation tables
DEVICE_ERR_ID_CARD	26	The IDCard is corrupted
DEVICE_ERR_DEVICE_ENABLING	27	The app has requested an option that is not enabled by the IDCard
DEVICE_ERR_LOADING_OCR	28	The OCR engine module

Panini Vision API 4.5.0
Reference Manual Revision 1

		cannot be loaded because is not present
DEVICE_ERR_CONNECTION_TIMEOUT	29	The connection attempt timeout elapsed
DEVICE_ERR_FEEDER_EMPTY	30	The device feeder is empty
DEVICE_ERR_FREE_TRACK_FAILED	31	The free track process failed
DEVICE_ERR_SOFTWARE_OVERLOAD	32	Internal error that signals fw/sw misbehaviors
DEVICE_ERR_STANDBY	33	The device cannot process documents because is in standby mode
DEVICE_ERR_NOT_COMPATIBLE_FW	34	The fw version is not compatible with the API
DEVICE_ERR_INVALID_SEND_PRINTER	35	Invalid or missing SendPrinterData function call
DEVICE_ERR_UNSUPPORTED_ID_CARD	36	The IDCard is not supported by the current API version.
DEVICE_ERR_INVALID_PARAMETERS	37	The app is setting one or more wrong device parameters
DEVICE_ERR_FEEDER_LIMIT	38	Feeder limitation triggered
DEVICE_ERR_WAITING_FEEDER_EMPTY	39	The device is waiting for the empty feeder due to the feeder limitation.
DEVICE_ERR_FEEDER_LIMIT_RESET	40	Feeder limit has been reset
DEVICE_ERR_NOT_AVAILABLE_MAX_DPM	41	The app requested a not available DPM
DEVICE_ERR_NOT_CALIBRATED_IMAGE	42	Image calibration parameters are missing
DEVICE_ERR_UPGRADE_UNSUCCESSFUL	43	IDCard upgrade failed
DEVICE_ERR_NO_PREREQUISITE_FOR_UPGRADE	44	The IDCard upgrade does not match the requested prerequisite
DEVICE_ERR_BUTTON_STOP	47	Stop feeding requested pressing the operator's button (Vision-S)
DEVICE_ERR_BUTTON_JAM	48	Jam triggered by the operator's button. (Vision-S)
DEVICE_ERR_INVALID_STATE	49	A function has been called during an invalid device state. (Vision-S)
DEVICE_ERR_DEFAULT_POCKET	50	The document has been

Panini Vision API 4.5.0
Reference Manual Revision 1

		sent in the default pocket ignoring the application pocket selection. (Vison-S)
DEVICE_ERR_US_DFD_SELF_TEST	51	The ultrasounds double feed sensor has failed the self-test. (Vision-X)
DEVICE_ERR_MISSING_FRANKING	52	The EnableFranking() call is missing when it is requested (I:Deal)
DEVICE_ERR_FRANKING_FAILED	53	The franking operation failed because the franking roller is not correctly mounted or is missing (I:Deal)
DEVICE_ERR_EXTENDED_PARMS	54	A wrong extended parameter has been used.
DEVICE_ERR_LIFT	55	The device has detected an error during the movements of the lift (VN with mobile AGP-14 only)

API Errors

API ERROR LABEL	API ERROR CODE	DESCRIPTION
API_ERR_NONE	0	No error
API_ERR_DEVICE_CREATE	1	Device creation failure
API_ERR_BRIDGE_THREAD	2	Internal error
API_ERR_POLLING_THREAD	3	Internal error
API_ERR_COMPRESSION_THREAD	4	Internal error
API_ERR_OCR_MANAGER	5	Generic OCR error
API_ERR_OCR_NOT_ENABLED	6	The OCR font is not available
API_ERR_BUFF_ALLOC	7	Buffers allocation failed
API_ERR_NO_USB_PORT_AVAILABLE	8	No USB ports available during the API start-up
API_ERR_LOG	9	Log error
API_ERR_INVALID_DEV_ID	10	API method called with an invalid Device ID
API_ERR_INVALID_DEV_OBJ	11	Internal error
API_ERR_INVALID_PARAM	12	Invalid parameter passed to API
API_ERR_INVALID_STATE	13	Method cannot be called during the current device state
API_ERR_DEVICE	14	Device Error
API_ERR_USB	15	USB Error
API_ERR_FPGA_PROGRAMMING	16	Internal error
API_ERR_E2PROM_WRITE	17	Internal error
API_ERR_E2PROM_READ	18	Internal error
API_ERR_SET_TIME_LEDS	19	Internal error
API_ERR_SET_GAINS	20	Internal error
API_ERR_SET_OFFSETS	21	Internal error
API_ERR_SERIAL_PORT	22	Internal error
API_ERR_SEND_SERIAL_COMMAND	23	Internal error
API_ERR_NOT_AVAILABLE_DEVICE	24	Device not available
API_ERR_TRANSFER_THREAD	25	Internal error
API_ERR_INTERNAL	26	Internal error
API_ERR_IQA	27	IQA Internal error
API_ERR_SNMP_REQUESTS_THREAD	28	Failure creating or resuming SNMP request Thread during API starting-up or shutting-down
API_ERR_DEVICE_ALREADY_IN_USE	29	Device already in use (connected) by other APP (StartUp() method)

Sorter Errors

NOTE: see *Appendix G*: Sorter errors (page 195) for Vision Next error codes
They are defined inside VApiInterface.h include file.

The error word (4 byte value) is composed in the following way:

Byte 3 (MSB)	- Error Class
Byte 2	- Peripheral involved in the error
Byte 1	- Error Code
Byte 0 (LSB)	- Jam Point

Error Class

ERR_GENERAL	0x00 : General error
ERR_COMM	0x01 : Error in Communication with PC
ERR_INIT	0x02 : HW initialization Error
ERR_PERIPHERAL	0x03 : Device error
ERR_TRANSPORT	0x04 : Error during paper transport

Peripheral Codes

GENERAL	0x00 : No Specific Peripheral
COMMUNICATION	0x01 : Error in Communication
READER_MODULE	0x02 : Error in MICR Reading module
SORTER_MODULE	0x04 : Error in Sorter module
IMAGE	0x06 : Error in Scanner module
PRINTER1	0x07 : Error in Printer module

Error Codes

ERR_NONE	0x00 : No errors
----------	------------------

Error codes related to ERR_COMM

ERR_CHECKSUM	0x01 : Checksum verification failure
ERR_UNKNOWN_CMD	0x02 : Unknown Command
ERR_INVALID_CMD	0x03 : Invalid Command
ERR_DOC_ID	0x04 : The Doc ID sent already exists
ERR_POCKET	0x05 : Invalid destination pocket requested
ERR_PREV_POCKET	0x07 : Missing pocket for previous document on feed request
ERR_PARAM	0x08 : Error reading/writing a parameter

Error codes related to ERR_INIT

ERR_INIT_PHOTOS	0x01 : Photocell initialization error
-----------------	---------------------------------------

Error codes related to ERR_PERIPHERAL

ERR_OPENED_COVER 0x01 : Open cover

Error codes related to ERR_TRANSPORT

ERR_DOC_LOST	0x01 : Docs never reached a photocell or lost in front of one of the photocells
ERR_DOC_LENGTH	0x02 : Document too long
ERR_DOC_FEED	0x03 : Missing document in the feeder or feed failure
ERR_FREE_TRACK	0x04 : Free track not completed
ERR_DOC_DEST	0x05 : Document without destination pocket
ERR_FULL_POCKET	0x06 : Pocket is full
ERR_DOUBLE_FEEDING	0x08 : Double-feeding detected
ERR_BUSY_PHOTO	0x0A : Photo busy during process start
ERR_PRINTER	0x0B : The printer device has detected a problem
ERR_IMAGE	0x0C : The image device has detected a problem
ERR_BUFFER_BUSY	0x0D : There's a buffer overwriting error
ERR_DOC_SHORT	0x11 : Document too short
ERR_FRANKING	0x12 : Franking failed
ERR_NO_MICROPERF	0x13 : Not Micro-perforated document detected
ERR_TWO_DOC_FEED	0x14 : Feeding two docs in a single doc device (Vision 1 only)
ERR_V1_REPOCKETING	0x15 : Re-pocketing error (Vision 1 only)

Function Calls

VisionAPI layers functions

Here are the interface layer functions. They are used to manage the VisionAPI interface layer. They are declared in the “*VApiInterface.h*” file.

These functions are defined with a new prototypes style. The MyVisionX API defined the functions in this manner:

```
BOOL FunctionName(Parameters)
```

Every function returns a BOOL value in order to signal operations end successfully or not. Then the application has to call the “GetError” functions to detect the reason of the problem.

The VisionAPI interface functions, instead, are defined in this manner:

```
ERR_CODE FunctionName(Parameters)
```

The difference is that these new set of functions directly return the eventual error code.

VApiGetVersion

```
VAPI_RET_TYPE VApiGetVersion( char* sVersion, DWORD MaxLen )
```

This function is used to obtain the VisionAPI interface release, which is returned as a NULL terminated string.

Arguments: **char* sVersion** – the pointer to a user allocated char buffer where the API version will be copied.

BYTE MaxLen – the length of the user allocated buffer.

Return Value: VAPI_ERR_NONE if successful.

Example:

```
char sVersion[250];
VAPI_RET_TYPE RetCode;
.....
// Get the VisionApi laver version
RetCode = VApiGetRelease( sVersion, sizeof(sVersion) );
if( RetCode != VAPI_ERR_NONE )
{
    // Function failed, RetCode contains the error code
}
```

VApiSetDeviceEngine

VAPI_RET_TYPE **VApiSetDeviceEngine**(DWORD EngineSelector)

This function is used to load an underlying device engine. This function is intended for future use, in order a different device engine to drive a specific Panini device. Now the unique supported function device is the MyVisionX / Vision|X. If this function is not called the VisionAPI automatically load the Vision|X device layer.

It must be called as the first function or after a “Shutdown” call, in other words when no devices are connected.

Available engine values are (see VapiInterface.h for an updated list):

VAPI_ENGINE_VX	0	// Vision X device engine selector
VAPI_ENGINE_VS	1	// Vision S device engine selector
VAPI_ENGINE_ID	2	// I-Deal device engine selector
VAPI_ENGINE_WI	3	// WIDeal device engine selector
VAPI_ENGINE_VXA4	4	// MFD device engine selector
VAPI_ENGINE_RESERVED1	5	// RFU
VAPI_ENGINE_VN	6	// VisionNext device engine selector

Arguments: **DWORD EngineSelector** – set the engine to be loaded. The valid values for this parameter are defined in the header file.

Return Value: VAPI_ERR_NONE if successful.

Example:

```
DWORD EngineSelector = VAPI_ENGINE_VX;
.....
// Load the desired device engine layer
RetCode = VApiSetDeviceEngine( EngineSelector );
if( RetCode != VAPI_ERR_NONE )
{
    // Function failed, RetCode contains the error code
}
.....
```

VApiGetDeviceList

VAPI_RET_TYPE **VApiGetDeviceList**(PDEVICELISTSTRUCT pDevList,
DWORD Length, DWORD * DetectedDevices)

This function is called to retrieve the information about the connected devices.

Arguments: **PDEVICELISTSTRUCT pDevList** – The pointer to the vector of structures that will be filled with the information about the connected devices.

DWORD Length – Length of the vector of structure pointed by pDevList

DWORD *DetectedDevices – Pointer to the variable in which the number of connected devices will be returned.

Return Value: VAPI_ERR_NONE if successful.

Structure DEVICELISTSTRUCT definition

DWORD DeviceType – Type of the device
char DeviceSerialNumber[MVX_SN_SIZE] – Serial Number of the device
DWORD DeviceID – The deviceId that will be used for the function calls
DWORD InternalDeviceID – not used
BOOL Connected – if it's connected or not

Example:

```
DEVICELISTSTRUCT DeviceListStr[MAX_DEVICE_NUMBER];  
DWORD DeviceIDs[MAX_DEVICE_NUMBER];  
.....  
VApiGetDeviceList( DeviceListStr, MAX_DEVICE_NUMBER,  
                   &DeviceCount );  
for (Ind=0; Ind<DeviceCount; Ind++)  
{  
    VApiSelectDevice (DeviceListStr[Ind].DeviceSerialNumber,  
                      DeviceListStr[Ind].DeviceType );  
    DeviceIDs[Ind] = StartUp (m_hWnd, WM_PANINI_MESSAGE);  
}
```

VApiSelectDevice

VAPI_RET_TYPE **VApiSelectDevice**(char *SerialNumber, DWORD DeviceType)

This function select the device to be started up, the start up called after it connects the device that has been selected.

Arguments: **char *SerialNumber** – Serial Number to be started up
DWORD DeviceType – Decice type of the machine to be strated up.

Return Value: VAPI_ERR_NONE if successful.

Example:

```
DEVICELISTSTRUCT DeviceListStr[MAX_DEVICE_NUMBER];  
DWORD DeviceIDs[MAX_DEVICE_NUMBER];  
.....  
VApiGetDeviceList( DeviceListStr, MAX_DEVICE_NUMBER,  
                   &DeviceCount );  
for (Ind=0; Ind<DeviceCount; Ind++)
```

```
{  
    VApiSelectDevice(DeviceListStr[Ind].DeviceSerialNumber,  
        DeviceListStr[Ind].DeviceType );  
    DeviceIDs[Ind] = StartUp(m_hWnd,WM_PANINI_MESSAGE);  
}
```

VApiGetError

VAPI_RET_TYPE *VApiGetError*(void)

This function gets the last occurred error. It has to be called after a function call failure to know the reason of the problem. Use this function with the other device engine layer to understand the reason of a function call failure.

Arguments: -

Return Value: The error code.

Example:

```
VAPI_RET_TYPE VApiError = 0;  
.....  
// Get the last occurred error  
VApiError = VApiGetError();  
if(VApiError != VAPI_ERR_NONE )  
{  
    // Manage the error code  
}  
else  
{  
    //Call the device engine layer "GetError" //function  
}  
.....
```

VApiGetErrorString

VAPI_RET_TYPE *VApiGetErrorString*(ERR_CODE ErrorCode, char* sErrorString,
int MaxLen)

This function gets an error code description. After a call to the VApiGetError the programmer can use this function to obtain a description of the error.

Arguments: ERR_CODE ErrorCode - The error code returned by the VApiGetError function
char *sErrorString – a user allocated string that receive the description
int MaxLen – the size of the user allocated string

Return Value: VAPI_ERR_NONE if successful.

Example:

```
VAPI_RET_TYPE VApiError = 0;
Char sErrorString[250];
// Get the last occurred error
VApiError = VApiGetError();
if(VApiError != VAPI_ERR_NONE )
{
    VApiGetErrorString( VApiError, sErrorString,
                        sizeof(sErrorString) );
    // Manage the error code
}
else
{
    //Call the device engine layer "GetError" //function
}
```

VApiWaitForInit

VAPI_RET_TYPE *VApiWaitForInit*(void)

This function blocks waiting for all engine threads to be up & running before returning. It has been introduced to allow an application to execute a LoadLibrary, wait for initialization to complete and then call FreeLibrary. It implicitly calls the *VApiLoad* if needed (i.e. the initialization process has not been started yet).

Arguments: -

Return Value: VAPI_ERR_NONE after all threads have been successfully started.

VApiLoad

BOOL *VApiLoad*(void)

This function initializes API and loads theEngine DLLs, waiting for all engine threads to be up & running before returning. After successfully calling this function, it is safe to call all other VAPI functions and operate on devices. The *VApiLoad*() can be called by the App after loading the VisionAPI dll (explicitly via LoadLibrary or implicitly via compile time linking) or will be automatically called the first time that VApiGetDeviceList, VApiSetDeviceEngine, VApiSelectDevice, VApiWaitForInit and Startup is executed.

Arguments: -

Return Value: TRUE if successful otherwise FALSE (see API logs for errors)

VApiUnload

BOOL *VApiUnload*(void)

This function unloads all Engines and prepares the API to be closed, blocking until the full process is completed. The *VApiUnload*() can be explicitly called by the App before executing the *FreeLibrary* or it will be automatically called inside *VAPI ExitInstance* (triggered by Windows while freeing the VisionAPI dll).

Arguments: -

Return Value: TRUE if successful otherwise FALSE (see API logs for errors)

VisionAPI errors

VisionAPI define its set of error codes related to its software layer. They are defined in “*VApiInterface.h*”. Here is the list of the errors:

- VAPI_ERR_NONE
- VAPI_ERR_ENGINE_LOAD_FAILED
- VAPI_ERR_FUNCTION_NOT_SUPPORTED
- VAPI_ERR_INVALID_PARAM
- VAPI_ERR_INTERNAL
- VAPI_ERR_DEVICEID
- VAPI_ERR_API_LOCKED

These error's codes are returned directly from the VisionAPI functions (*VApiGetVersion*, *VApiSetDeviceEngine*, *VApiGetError* and *VApiGetErrorString*) or calling *VApiGetError* after a device function call failure.

The previous functions *GetApiError*, *GetDeviceError*, *GetUsbError* and *GetSorterError* have still to be used.

Device layer functions

Here are the device layer functions. They are used to drive the physical device.

Information Functions

GetApiRelease

BOOL *GetApiRelease*(char* sVersion, BYTE MaxLen)

This function is used to obtain the API release, which is returned as a NULL terminated string. This is an OBSOLETE function working just if the function **VApiSelectDevice** is not called so the application is working in single “device mode”. If not the function will return an FALSE. Call **GetEngineApiRelease** instead.

Valid in State : All

State Transition: None

Arguments: char* **sVersion** – the pointer to a non-NULL char buffer where the API version will be stored.

BYTE **MaxLen** – the length of the user allocated buffer.

Return Value: TRUE if successful.

Example:

```
char sVersion[250];
BOOL bOk;
.....
// Read the Api Release
bOk = GetApiRelease( sVersion, 250 );
.....
```

GetEngineApiRelease

BOOL *GetEngineApiRelease* (int DeviceType, char* sVersion, BYTE MaxLen)

This function is used to obtain the API release of a defined Engine, which is returned as a NULL terminated string.

Valid in State : All

State Transition: None

Arguments: int **DeviceType** – Type of the engine for which the release is required.

char* sVersion – the pointer to a non-NULL char buffer where the API version will be stored.

BYTE MaxLen – the length of the user allocated buffer.

Return Value: TRUE if successful.

Example:

```
char sVersion[250];
BOOL bOk;
.....
// Read the Api Release
bOk = GetEngineApiRelease(DeviceType ,sVersion, 250 );
.....
```

GetDriverRelease

BOOL GetDriverRelease(char* sVersion, BYTE MaxLen)

This function is used to obtain the driver release, which is returned as a NULL terminated string.

Valid in State : All

State Transition: None

Arguments: **char* sVersion** – the pointer to a non-NULL char buffer where Driver version will be stored.

BYTE MaxLen – the length of the user allocated buffer.

Return Value: TRUE if successful.

Example

```
char sVersion[250];
BOOL bOk;

.....
// Read the Driver Release
bOk = GetDriverRelease( sVersion, 250 );
.....
```

GetFWVersion

BOOL *GetFwVersion*(DWORD DeviceID, char* sVersion, BYTE MaxLen)

This function is used to obtain the Firmware version, which is returned as a NULL terminated string. It can be called only after the device is connected, because it is retrieved directly from the device.

Valid in State : *DeviceChangeParameters, DeviceOnLine, DeviceOffLine*

State Transition: None

Arguments: **DWORD DeviceID** – the Identification number of the device.

char* sVersion – the pointer to a non-NULL char buffer where the Firmware version will be stored.

BYTE MaxLen – the length of the user allocated buffer.

Return Value: TRUE if successful.

Example

```
char sVersion[250];
BOOL bOk;
.....
// Read the Firmware version
bOk = GetFwVersion( m_DeviceID, sVersion, 250 );
.....
```

GetSerialNumber

BOOL *GetSerialNumber*(DWORD DeviceID, char* pSerialNumber,
 BYTE MaxLen)

This function is used to obtain the Serial number of the connected reader, which is returned as a NULL terminated string. It can be called only after the device is connected, because it is retrieved directly from the device.

Valid in State : *DeviceChangeParameters, DeviceOnLine, DeviceOffLine*

State Transition: None

Arguments: **DWORD DeviceID** – the Identification number of the device.

char* pSerialNumber – the pointer to a non-NULL char buffer where the serial number will be stored.

BYTE MaxLen – the length of the user allocated buffer.

Return Value: TRUE if successful.

Example

```
char sSerial[15];  
BOOL bOk;  
.....  
// Read the Serial Number  
bOk = GetSerialNumber( m_DeviceID, sSerial, 15 );  
.....
```

ReadCryptedIDCard

BOOL *ReadCryptedIDCard*(DWORD DeviceID, BYTE *pIDCard, BYTE *pLen)

This function is used to obtain the IDCard encrypt code.

Valid in State : *DeviceChangeParameters*

State Transition: None

Arguments: DWORD **DeviceID** – the Identification number of the device.

 BYTE ***pIDCard** – the pointer to a Byte buffer where the encrypt IDCard will be stored.

 BYTE ***pLen** – Input: contain the pIDCard length (at least 128 bytes)

 Output: contain the bytes copied in pIDCard.

 VisionX and VisionS require 128 bytes.

 I-Deal requires 512 bytes. Since pLen is a BYTE the application has to set this parameter to 512/8.

Return Value: TRUE if successful.

Example

```
BYTE IDCardBuffer[200];  
BOOL bOk;  
BYTE RealLen;  
.....  
// Read the crypted IDCard  
bOk = ReadCryptedIDCard( m_DeviceID, IDCardBuffer, &RealLen );  
.....
```

IDCard and capabilities request functions

GetIDCardDescription

BOOL *GetIDCardDescription*(DWORD DeviceID, char **pDescription, int *pLen,
DEVICES *Devices)

This function is used to obtain the description of the device from the IDCard, which stores all information about rights and licenses. The buffer for the description is a pointer address supplied by the application and will be internally allocated, so remember to free it with VirtualFree() when it is no longer needed. The DEVICES structure will contain the features of the device. The DEVICES structure members are the following:

char SerialNumber[11]	: Serial number 53xxxxxx
BYTE FeederLimit	: Valid values are 0(Unlimited), or 30
BYTE MaxDPM	: Document per minute
BOOL MicrE13B	: TRUE means device available
BOOL MicrCMC7	: TRUE means device available
BOOL MicrAUTO	: TRUE means device available
BOOL Inkjet	: TRUE means device available
BOOL InkjetMultiline	: TRUE means device available
BOOL InkjetGraphic	: TRUE means device available
BYTE InkjetLines	: Valid values are 0 (Unlimited), 1, 2 and 3 lines.
BOOL ImageFront	: TRUE means device available
BOOL ImageFrontClr	: TRUE means device available
BOOL ImageFrontDropOut	: TRUE means device available
BOOL ImageRear	: TRUE means device available
BOOL ImageRearClr	: TRUE means device available
BOOL ImageRearDropOut	: TRUE means device available
BOOL OcrAB	: TRUE means device available
BOOL OcrMicr	: TRUE means device available
BOOL OcrBarcode1D	: TRUE means device available
BOOL OcrBarcode2D	: TRUE means device available

Valid in State : ***DeviceChangeParameters***

State Transition: None

Arguments: DWORD **DeviceID** – the Identification number of the device.

 char ****pDescription** – the address of the pointer where to allocate and store the device description received from the IDCard.

 BYTE ***pLen** – Receive the IDCard description's length.

Return Value: TRUE if successful.

Example

```
char *sDescription;
int  RealLen;
.....
if( GetIDCardDescription( m_DeviceID, &sDescription, &Len ) )
{
    .....
    VirtualFree( sDescription, 0, MEM_RELEASE );
}
.....
```

UpgradeIDCard

BOOL UpgradeIDCard(DWORD DeviceID, BYTE *pIDCardUpgrade, DWORD Len)

Use this function to upgrade a device using an IDCard Upgrade buffer. If the missing of a prerequisite is detected a `DEVICE_ERR_NO_PREREQUISITE_FOR_UPGRADE` error is retrieved.

Valid in State : *DeviceChangeParameters*

State Transition: None

Arguments: **DWORD DeviceID** – the Identification number of the device.
 BYTE *pIDCardUpgrade – it's the buffer that contain the IDCard upgrade data.
 DWORD Len – it's the length of the upgrade buffer (256 bytes).
 I-Deal requires 1024 bytes.

Return Value: TRUE if successful.

Example

```
FILE *pIDCardUpgradeFile;
BYTE pIDCardBuffer1[256] ;
.....
pIDCardUpgradeFile = fopen("MVXUpgrade.UpID","rb");
if( pIDCardUpgradeFile )
{
    fread(pIDCardBuffer1,256,pUpgradeFile);
    fclose(pIDCardUpgradeFile);
}
if( UpgradeIDCard( m_DeviceID, pIDCardBuffer1, 256 ) )
{
    .....
}
.....
```

GetDeviceFeature

BOOL *GetDeviceFeature*(**DWORD** DeviceID, **DWORD** FeatureID, **LPVOID** lpReturnedBuffer, **DWORD** BufferSize)

This function is used to obtain information, capabilities and licenses of the device. Most of the information returned by this function is equal to the ones returned by the *GetIDCardDescription* function.

The information are requested passing a feature ID number (*FeatureID*) to the function that returns the corresponding data in the user allocated buffer (*lpReturnedBuffer*). The buffer must be big enough to contain the data (*BufferSize*).

The data are defined as **DWORD** (32 bits) or char string. The features ID are defined in the header file.

The following table lists the features that can be requested:

FeatureID	Type	Size	Values	Description
DEVICE_FEATURE_DEVICE_TYPE	DWORD	4	DEVICE_MYVISIONX (0) DEVICE_VISIONX (1) DEVICE_VISIONS (2) DEVICE_IDEAL (3) DEVICE_WIDEAL (4) DEVICE_VXA4 (5) DEVICE_RESERVED1 (6) DEVICE_VN (7)	This value identifies the device version
DEVICE_FEATURE_SN	char*	11	Null terminated string	It's the device serial number
DEVICE_FEATURE_SN_IDCARD	char*	11	Null terminated string	It's the IDCard serial number
DEVICE_FEATURE_EXTERNAL_IDCARD	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	This value identifies if an external IDCard is connected
DEVICE_FEATURE_SN_EXTERNAL_IDCARD	char*		Null terminated string	It's the external IDCard serial number
DEVICE_FEATURE_DPM	DWORD	4	DPM number	It's the maximum DPM
DEVICE_FEATURE_FEEDER_LIMIT	DWORD	4	0 means unlimited. Others are the maximum feeder capacity	It's feeder capacity
DEVICE_FEATURE_FEEDER_SD	DWORD	4	DEVICE_FEATURE_NO (0) DEVICE_FEATURE_YES (1) DEVICE_FEATURE_V1 (2) DEVICE_FEATURE_Vx1B (3)	This value identifies an SD device (see VX appendix for more info)
DEVICE_FEATURE_MICR_E13B	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	MICR E13B dis/enable
DEVICE_FEATURE_MICR_CMC7	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	MICR CMC7 dis/enable
DEVICE_FEATURE_MICR_AUTO	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	MICR AUTO dis/enable
DEVICE_FEATURE_MICR_OCR_E13B	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	MICR+OCR E13B dis/enable
DEVICE_FEATURE_MICR_OCR_CMC7	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	MICR+OCR CMC7 dis/enable
DEVICE_FEATURE_PRINTER_TYPE	DWORD	4	DEVICE_PRINTER_DISABLED DEVICE_PRINTER_SINGLE_LINE	This value identifies

Panini Vision API 4.5.0 Reference Manual Revision 1

			DEVICE_PRINTER_AGP	the printer version
DEVICE_FEATURE_PRINTER_LINES	DWORD	4	0 means unlimited. Others are the maximum printer's lines	It's the printer's available lines
DEVICE_FEATURE_PRINTER_BITMAP	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	Printer bitmap dis/enable
DEVICE_FEATURE_PRINTER_BITMAP_HEIGHT	DWORD	4	Single line: 12 pixels AGP: 25, 50, 75 or 100	It's the height of the in pixels
DEVICE_FEATURE_IMAGE_FRONT	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	Front image dis/enable
DEVICE_FEATURE_IMAGE_FRONT_FAST_COLOR	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	Front image Fast color dis/enable
DEVICE_FEATURE_IMAGE_FRONT_TRUE_COLOR	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	Front image True Color dis/enable
DEVICE_FEATURE_IMAGE_FRONT_FAST_DROPOUT	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	Front image Fast Dropout dis/enable
DEVICE_FEATURE_IMAGE_FRONT_DPI	DWORD	4	200, 300	It's the front image max resolution (DPI)
DEVICE_FEATURE_IMAGE_FRONT_FAST_DROPOUT_IR	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	Front image Fast Infrared Dropout dis/enable
DEVICE_FEATURE_IMAGE_REAR	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	Rear image dis/enable
DEVICE_FEATURE_IMAGE_REAR_FAST_COLOR	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	Rear image Fast color dis/enable
DEVICE_FEATURE_IMAGE_REAR_TRUE_COLOR	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	Rear image True Color dis/enable
DEVICE_FEATURE_IMAGE_REAR_FAST_DROPOUT	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	Rear image Fast Dropout dis/enable
DEVICE_FEATURE_IMAGE_REAR_DPI	DWORD	4	200, 300	It's the Rear image max resolution (DPI)
DEVICE_FEATURE_IMAGE_REAR_FAST_DROPOUT_IR	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	Rear image Fast Infrared Dropout dis/enable
DEVICE_FEATURE_OCR_AB	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	OCR A/B fonts dis/enable
DEVICE_FEATURE_OCR_MICR	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	OCR MICR fonts dis/enable
DEVICE_FEATURE_OCR_BARCODE_1D	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	Barcode 1D fonts dis/enable
DEVICE_FEATURE_OCR_BARCODE_2D	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	Barcode 2D fonts dis/enable
DEVICE_FEATURE_IQA	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	IQA dis/enable
DEVICE_FEATURE_POCKETS	DWORD	4	1, 2	It's the maximum number of pockets available
DEVICE_FEATURE_ROHS_COMPLIANT	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	ROHS HW compliance

Panini Vision API 4.5.0

Reference Manual Revision 1

DEVICE_FEATURE_SMART_JET	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	Smart-Jet dis/enable
DEVICE_FEATURE_MAGCARD	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	Magcard dis/enable
DEVICE_FEATURE_OCR_ENGINE	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	OCR engine installed
DEVICE_FEATURE_BARCODE_1D_ENGINE	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	Barcode 1D engine installed
DEVICE_FEATURE_BARCODE_2D_ENGINE	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	Barcode 2D engine installed
DEVICE_FEATURE_MICR_OCR_ENGINE	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	MICR+OCR engine installed
DEVICE_FEATURE_ULTRASONIC_DFD	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	DFD ultrasound option available
DEVICE_FEATURE_CARDS	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	The device is able to process rigid documents
DEVICE_FEATURE_FRANKING	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	The franking roller stamp available
DEVICE_FEATURE_FRANKING_LENGTH	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	The franking roller length in mm
DEVICE_FEATURE_PAGES	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	The device is able to process page documents
DEVICE_FEATURE_OCR_PAGE	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	Page OCR is enabled.
DEVICE_FEATURE_OCR_PAGE	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	Page OCR is enabled.
DEVICE_FEATURE_PHYSICAL_POCKETS	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	The device mounts sorters.
DEVICE_FEATURE_PHYSICAL_POCKETS	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	The device mounts sorters.
DEVICE_FEATURE_IMAGE_FRONT_UV	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	The device mounts a Front UV Image sensor.
DEVICE_FEATURE_IMAGE_REAR_UV	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	The device mounts a Rear UV Image sensor.
DEVICE_FEATURE_ADF	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	The device has an automatic feeder (applicable to VX-A4Pg)
DEVICE_FEATURE_RESERVED1	DWORD	4	-	Reserved feature
DEVICE_FEATURE_CARDS_IMAGE_FRONT_DPI	DWORD	4	300, 600	It's the front image max resolution (DPI) for Rigids document (applicable to VN).
DEVICE_FEATURE_CARDS_IMAGE_REAR_DPI	DWORD	4	300, 600	It's the rear image max resolution

Panini Vision API 4.5.0 Reference Manual Revision 1

				(DPI) for Rigid document (applicable to VN).
DEVICE_FEATURE_PAGES_IMAGE_FRONT_DPI	DWORD	4	300, 600	It's the front image max resolution (DPI) for Pages document. (applicable to VN).
DEVICE_FEATURE_PAGES_IMAGE_REAR_DPI	DWORD	4	300, 600	It's the rear image max resolution (DPI) for Pages document. (applicable to VN).
DEVICE_FEATURE_PRINTER_MULTI_AREA	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	The device has a multi-area mobile AGP-14 printer (applicable to VN)
DEVICE_FEATURE_PRINTER_SERVICE	DWORD	4	DEVICE_FEATURE_NO DEVICE_FEATURE_YES	The device has the cartridge cleaning station (applicable to VN)

Valid in State : ***DeviceChangeParameters, DeviceOnLine***
State Transition: None

Arguments: **DWORD DeviceID** – the Identification number of the device.

DWORD FeatureID – It's the requested Feature ID number.

LPVOID lpReturnedBuffer – It's the user allocated buffer that receive the feature data.

DWORD BufferSize – It's the buffer size. It must be big enough to contain the data.

Return Value: TRUE if successful.

 If a device does not support at all a feature the function returns FALSE to application and the device error is set to `DEVICE_ERR_INVALID_PARAMETERS`.

Example

```
char SerialNumber[20];
DWORD FeatureValue;
.....
```

Panini Vision API 4.5.0 Reference Manual Revision 1

```
if( GetDeviceFeature( m_DeviceID, DEVICE_FEATURE_DEVICE_TYPE,
&FeatureValue, 4 ) )
{
    if( FeatureValue == DEVICE_VISIONX )
        MessageBox("VisionX device", "Device type", MB_OK );
}
if( GetDeviceFeature( m_DeviceID, DEVICE_FEATURE_SN, SerialNumber,
sizeof(SerialNumber) ) )
{
    MessageBox( SerialNumber, "Device serial number", MB_OK );
}
.....
```

Error Request Functions

GetUsbError

BOOL *GetUsbError*(DWORD DeviceID, DWORD * pdwError)

Use this function to retrieve information if a USB error occurs. pdwError will be the USB error code (see Error Handling Chapter for more information).

Valid in State : All

State Transition: None

Arguments: **DWORD DeviceID** – the Identification number of the device.

DWORD * pdwError – the pointer to a DWORD in which the Error Code will be stored.

Return Value: TRUE if successful.

Example

```
DWORD ErrorCode;
.....
if( GetUsbError ( m_DeviceID, &ErrorCode ) )
{
    .....
    // Report error
    MessageBox(...);
}
.....
```

GetUsbErrorString

BOOL *GetUsbErrorString*(DWORD DeviceID, char * pcErrorString, int MaxLen)

Use this function to retrieve information if a USB error occurs. The pointer pcErrorString will be the USB error description string.

Valid in State : All

State Transition: None

Arguments: **DWORD DeviceID** – the Identification number of the device.

 char * **pcErrorString** – the pointer to a string in which the Error description will be stored.

 int **MaxLen** – the max length of the string

Return Value: TRUE if successful.

Example

```
char ErrorString[200];
.....
if( GetUsbErrorString( m_DeviceID, ErrorString, 200 ) )
{
    .....
    // Report error
    MessageBox (...) ;
}
.....
```

GetDeviceError

BOOL *GetDeviceError*(DWORD DeviceID, DWORD * pdwError)

Use this function to retrieve information if a Device error occurs. pdwError will be the Device error code (see Error Handling Chapter for more information).

Valid in State : All

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

 DWORD * **pdwError** – the pointer to a DWORD in which the Error Code will be stored.

Return Value : TRUE if successful.

Example

```
DWORD DeviceErrorCode;  
DWORD UsbErrorCode;  
.....  
// A DEVICE ERROR has occurred, it could be due to an USB error  
GetDeviceError( m_DeviceID, & DeviceErrorCode );  
if( DeviceErrorCode == DEVICE_ERR_USB )  
{  
    // USB ERROR  
    GetUsbError ( m_DeviceID, & UsbErrorCode );  
    .....  
}
```

GetDeviceErrorString

BOOL *GetDeviceErrorString*(DWORD DeviceID, char * pcErrorString,
int MaxLen)

Use this function to retrieve information if a Device error occurs. pcErrorString will be the Device error description string.

Valid in State : All

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

char * pcErrorString – the pointer to a string in which the Error description will be stored.

int MaxLen – the max length of the string

Return Value : TRUE if successful.

Example

```
char ErrorString[200];  
.....  
if( GetDeviceErrorString( m_DeviceID, ErrorString, 200 ) )  
{  
    .....  
    // Report error  
    MessageBox (...);  
}  
.....
```

GetApiError

BOOL *GetApiError*(DWORD * pdwError)

Use this function to retrieve information if an API error occurs. pdwError will be the API error code (see Error Handling Chapter for more information).

This function is OBSOLETE, call **GetEngineApiError** instead. If the **VApiSelectDevice** is called the function will return FALSE.

Valid in State : All

State Transition : None

Arguments : DWORD * **pdwError** – the pointer to a DWORD in which the Error Code will be stored.

Return Value : TRUE if successful.

Example

```
DWORD ApiErrorCode;
DWORD UsbErrorCode;
DWORD DeviceErrorCode;

.....
// An API ERROR has occurred, it could be due to a USB error or
to a // device error
GetApiError( &ApiErrorCode );
if( ApiErrorCode == API_ERR_USB )
{
    // USB ERROR
    GetUsbError ( m_DeviceID, & UsbErrorCode );
    .....
}

else if( ApiErrorCode == API_ERR_DEVICE )
{
    // DEVICE ERROR
    GetDeviceError ( m_DeviceID, & DeviceErrorCode );
    .....
}
```

GetEngineApiError

BOOL *GetEngineApiError*(int DeviceType, DWORD * pdwError)

Use this function to retrieve information if an API error occurs for a certain device. pdwError will be the API error code (see Error Handling Chapter for more information).

Valid in State : All

State Transition : None

Arguments : int **DeviceType** – The type of the device.

DWORD * **pdwError** – the pointer to a DWORD in which the Error Code will be stored.

Return Value : TRUE if successful.

Example

```
DWORD ApiErrorCode;
DWORD UsbErrorCode;
DWORD DeviceErrorCode;
.....
// An API ERROR has occurred, it could be due to a USB error or
to a // device error
GetApiError( DeviceType, &ApiErrorCode );
if( ApiErrorCode == API_ERR_USB )
{
    // USB ERROR
    GetUsbError ( m_DeviceID, & UsbErrorCode );
    .....
}

else if( ApiErrorCode == API_ERR_DEVICE )
{
    // DEVICE ERROR
    GetDeviceError ( m_DeviceID, & DeviceErrorCode );
    .....
}
```

GetApiErrorString

BOOL *GetApiErrorString*(char * pcErrorString, int MaxLen)

Use this function to retrieve information if an API error occurs. pcErrorString will be the API error description string.

This function is OBSOLETE, call **GetEngineApiErrorString** instead. If the **VApiSelectDevice** is called the function will return FALSE.

Valid in State : All

State Transition : None

Arguments : char * **pcErrorString** – the pointer to a string in which the Error description will be stored.

int **MaxLen** – it's the max length of the string.

Return Value : TRUE if successful.

Example


```
char ErrorString[200];
.....
if( GetApiErrorString( ErrorString, 200 ) )
{
    .....
    // Report error
    MessageBox (...);
}
.....
```

GetEngineApiErrorString

BOOL *GetEngineApiErrorString*(int DeviceType, char * pcErrorString, int MaxLen)

Use this function to retrieve information if an API error occurs. pcErrorString will be the API error description string.

Valid in State : All

State Transition : None

Arguments : int **DeviceType** – The type of the device.

char * **pcErrorString** – the pointer to a string in which the Error description will be stored.

int **MaxLen** – it's the max length of the string.

Return Value : TRUE if successful.

Example

```
char ErrorString[200];
.....
if( GetEngineApiErrorString( DeviceType, ErrorString, 200 ) )
{
    .....
    // Report error
    MessageBox (...);
}
.....
```

GetSorterError

BOOL *GetSorterError*(DWORD DeviceID, DWORD * pdwError)

Use this function to retrieve information directly from the firmware, it is to be used only when more information about a problem is needed. pdwError will be the Sorter error code (see Error Handling Chapter for more information).

Valid in State : All

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

 DWORD * **pdwError** – the pointer to a DWORD in which the Error Code will be stored.

Return Value : TRUE if successful.

Example

```
DWORD SorterErrorCode;
BYTE  ErrorClass;
BYTE  Peripheral;
BYTE  ErrorCode;
BYTE  JamPoint;
.....
if(GetSorterError(m_DeviceID, &SorterErrorCode))
{
    ErrorClass = (SorterErrorCode & 0xFF000000) >> 24;
    Peripheral = (SorterErrorCode & 0x00FF0000) >> 16;
    ErrorCode = (SorterErrorCode & 0x0000FF00) >> 8;
    JamPoint = (SorterErrorCode & 0x000000FF);
}
```

GetSorterErrorString

BOOL *GetSorterErrorString*(DWORD DeviceID, char * pcErrorString, int MaxLen)

Use this function to retrieve information directly from the firmware, it is to be used only when more information about a problem is needed. pcErrorString will be the Sorter error description string.

Valid in State : All

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

char * **pcErrorString** – the pointer to a string in which the Error description will be stored.

BYTE **MaxLen** – the length of user allocated buffer.

Return Value : TRUE if successful.

Example

```
char ErrorString[200];
.....
if( GetSorterErrorString( m_DeviceID, ErrorString, 200 ) )
{
    .....
    // Report error
    MessageBox(...);
}
```

GetIQALastError

BOOL *GetUsbError*(DWORD DeviceID, int *Error)

Use this function to retrieve the IQA last error.

Valid in State : All

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

int ***Error** – the pointer to an int in which the Error Code will be returned.

Return Value : TRUE if successful.

Example

```
DWORD ErrorCode;
.....
if( GetIQALastError ( m_DeviceID, &ErrorCode ) )
{
    .....
    // Report error
    MessageBox(...);
}
.....
```

GetIQALastErrorString

BOOL *GetUsbErrorString*(DWORD DeviceID, char *ErrorString,
DWORD MaxErrorStringLen)

Use this function to retrieve the description of an IQA error code.

Valid in State : All

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

 char ***ErrorString** – the pointer to a user allocated string in which the Error description will be stored.

DWORD MaxErrorStringLen – the max length of the string

Return Value : TRUE if successful.

Example

```
char ErrorString[200];
.....
if( GetIQALastErrorString( m_DeviceID, ErrorString, 200 ) )
{
    .....
    // Report error
    MessageBox(...) ;
}
.....
```

Device States Control Functions

GetDeviceState

BOOL *GetDeviceState*(DWORD DeviceID, DWORD * pdwDeviceState)

Use this function to retrieve the current status code of the device.

Valid in State : All

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

DWORD * pdwDeviceState– the pointer to a DWORD in which the current state will be stored.

Return Value : TRUE if successful.

Example

```
DWORD DeviceState;  
.....  
if( GetDeviceState( m_DeviceID, &DeviceState ) )  
{  
    .....  
}
```

GetDeviceStateString

BOOL *GetDeviceStateString*(DWORD DeviceID, char * pcDeviceStateString, int MaxLen)

Use this function to retrieve the Device State string. pcDeviceStateString will be the Device State string.

Valid in State : All

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

char * pcDeviceStateString – the pointer to a string in which the Device State will be stored.

BYTE MaxLen – the length of the user allocated buffer.

Return Value : TRUE if successful.

Example

```
char DeviceStateString[200];
.....
if( GetDeviceStateString( m_DeviceID, DeviceStateString, 200) )
{
    .....
    // Report error
    MessageBox(...);
}
.....
```

StartUp

DWORD **StartUp**(HWND Handle, UINT SorterMessage)

This function call opens a communication channel between the device and the application, if the StartUp is successful the Device Identification Number is returned.

Valid in State : **DeviceShutDown**

State Transition : To **DeviceStartingUp** and then to **DeviceChangeParameters**

Arguments : HWND **Handle** – the handle to the application's messages destination window.

UINT **SorterMessage** – a safe user message identifier defined inside the application.

Return Value : DeviceID to be used for all further calls regarding this Reader. If 0 is retrieved an error occurred, use **GetApiError** or **GetApiErrorString** to get more information about it.

Example

```
#define WM_SORTER_API (WM_APP+10)    // Safe user defined message
                                      // for API
.....
DWORD m_DeviceID;
char ApiErrorString[200];
.....
m_DeviceID = StartUp( m_hWnd, WM_SORTER_API );
if( !m_DeviceID )
{
    .....
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox(...);
}
.....
```

ShutDown

BOOL *ShutDown*(DWORD DeviceID)

This function closes the communication channel between the sorter and the pc

Valid in State : *All*

State Transition : to *DeviceShutDown*

Arguments : DWORD **DeviceID** – the Identification number of the device.

Return Value : TRUE if the communication has been closed in the proper way, if an error occurred FALSE is returned, then call ***GetApiError*** or ***GetApiErrorString*** to get more information about it.

Example

```
char ApiErrorString[200];  
.....  
if( !ShutDown ( m_DeviceID ) )  
{  
    .....  
    // Report error  
    GetApiErrorString( ApiErrorString, 200 );  
    MessageBox(...);  
}  
.....
```

ChangeParameters

BOOL *ChangeParameters*(DWORD DeviceID)

This function forces the device in the *DeviceChangeParameters* state.

Valid in State : *DeviceOnLine, DeviceOffline*

State Transition : to *DeviceChangeParameters*

Arguments : DWORD **DeviceID** – the Identification number of the device.

Return Value : FALSE if the device is not configurable to the *DeviceChangeParameters* state, then call ***GetApiError*** or ***GetApiErrorString*** to get more information about it.

Example

```
char ApiErrorString[200];  
char pcDeviceStateString[100];  
.....
```

```
if( ChangeParameters ( m_DeviceID ) )
{
    // Get Device State String, it must be
    // "DeviceChangeParameters"
    GetDeviceStateString( DeviceID, pcDeviceStateString,
        100 );
}
else
{
    .....
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox(...);
}
.....
```

OnLine

BOOL **OnLine**(DWORD DeviceID)

This function forces the device into the **OnLine** state which is the operative state.
In Multi Document devices, this function carries out parameter compatibility analysis.

Valid in State : **DeviceChangeParameters**, **DeviceOffLine**

State Transition : to **DeviceOnLine**

Arguments : DWORD **DeviceID** – the Identification number of the device.

Return Value : FALSE if the device is not configurable to the **DeviceOnLine** state, then call **GetApiError** or **GetApiErrorString** to get more information about it.
In Multi Document devices, returns FALSE if the processing parameters were set erroneously.

Example

```
char ApiErrorString[200];
char pcDeviceStateString[100];
.....
if( OnLine ( m_DeviceID ) )
{
    // Get Device State String, it must be
    // "DeviceOnLine"

    GetDeviceStateString( DeviceID, pcDeviceStateString,
        100 );
}
else
{
    .....
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox(...);
}
```


OffLine

BOOL *OffLine*(DWORD DeviceID)

This function forces the device into the *OffLine* state, this is the diagnostic state.

Valid in State : *DeviceChangeParameters, DeviceOnLine*

State Transition : to *DeviceOffLine*

Arguments : DWORD **DeviceID** – the Identification number of the device.

Return Value : FALSE if the device is not configurable to the *DeviceOffLine* state, then call *GetApiError* or *GetApiErrorString* to get more information about it.

Example

```
char ApiErrorString[200];
char pcDeviceStateString[100];
.....
if( OffLine ( m_DeviceID ) )
{
    // Get Device State String, it must be
    // "DeviceOffLine"
    GetDeviceStateString( DeviceID, pcDeviceStateString,
        100 );
}
else
{
    .....
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox(...);
}
.....
```

GetDeviceStatus (VN only)

BOOL *GetDeviceStatus*(DWORD DeviceID, BYTE* pDeviceStatus)

It is a synchronous function available for Vision NeXt only, that gives back the current “physical” device state from a FW stand-point (not to be confused with the *GetDeviceState*() which returns the current device “logic” state from an Engine stand-point).

It can be called by the App in whatever Engine state to understand if the VN is able to perform scan operation (i.e. start the feeding) or some error/maintenance conditions are pending/running and therefore the App shall wait or execute a FreeTrack to recover.

Valid in State : All

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

BYTE * pDeviceStatus – the pointer to a BYTE in which the current status will be stored.

Return Value : TRUE if successful, with the following status value:

DEVICE_STATUS_READY (0) → VN is Ready to “accept commands”, i.e. go in Feeding

DEVICE_STATUS_NOT_READY (1) → VN is Not ready (there’s an error, a document jammed or an active free track or other lift movement operation running)

DEVICE_STATUS_COVER_OPEN (2) → VN has the top cover open

DEVICE_STATUS_MAINTENANCE (3) → VN is performing (either automatic or manual) cartridge maintenance operation: purging, cleaning or capping

DEVICE_STATUS_FULL_POCKET (4) → VN has a full pocket condition (either destination or exception pocket – use the *GetFullPocketStatus* call to understand which one)

Example

```
DWORD dwState = 0;
BYTE byStatus = 0;
if (GetDeviceState( m_DeviceID, &dwState ))
{
    if (GetDeviceStatus( m_DeviceID, &byStatus ))
    {
        sText.Format("Device State: %u - Status: %u", dwState, byStatus);
        WriteListBoxReport(sText);
    }
    else
    {
        // Report error
        MessageBox(...); () ;
    }
}
else
{
    // Report error
    MessageBox(...); () ;
}
```

Parameters Managing Functions

SetSorterParameter

BOOL *SetSorterParameter*(DWORD DeviceID, USHORT unParamId,
USHORT unParamValue)

Set a sorter parameter (see the chapter on FW Parameter Description).

Valid in State : ***DeviceChangeParameters, DeviceOffLine, DeviceOnLine***
State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

USHORT unParamId – the identification of the parameter that will be changed.

USHORT unParamValue – the value that will be set for the selected unParamId.

Return Value : FALSE if an error occurs.

Example

```
USHORT ParamID;  
USHORT ParamValue;  
char ApiErrorString[200];  
.....  
if( !SetSorterParameter( m_DeviceID, ParamID, ParamValue ) )  
{  
    // Report error  
    GetApiErrorString( ApiErrorString, 200 );  
    MessageBox(...);  
    .....  
}
```

GetSorterParameter

BOOL *GetSorterParameter*(DWORD DeviceID, USHORT unParamId,
AParameter * pParamStruct)

Retrieve from the FW a sorter parameter (see the chapter on FW Parameter Description).

AParameter is a structure defined in this way:

- **sDescription** : String describing the Parameter
- **sUnit** : String describing the usable unit for the parameter
- **usValue** : Actual value of the parameter
- **usDefault** : Default value of the parameter
- **usMin** : Minimum value available
- **usMax** : Maximum value available

Valid in State : ***DeviceChangeParameters, DeviceOffLine, DeviceOnLine***

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

USHORT unParamId – the identification of the parameter for which the value will be asked.

AParameter * pParamStruct – the pointer to the structure in which the Parameter will be stored.

Return Value : FALSE if an error occurred.

Example

```
AParameter ParamStruct;
USHORT ParamID;
USHORT RetrievedValue;
char ApiErrorString[200];
.....
if( !GetSorterParameter( m_DeviceID, ParamID, &ParamStruct ) )
{
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox(...);
    .....
}
else
{
    RetrievedValue = ParamStruct.usValue;
}
```

SetMicrOcrFontOverride

BOOL SetMicrOcrFontOverride (**DWORD DeviceID**, **BOOL bEnabled**)

To disable/enable the MICR+OCR enforcing (which by default is enabled on API 4.5+ for all Engines).

Valid in State : ***DeviceChangeParameters***

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

BOOL bEnabled - FALSE (disable) / TRUE (enable)

Return Value : TRUE if successful otherwise FALSE (see logs for errors)

NOTE: This function must be called before the *SetDeviceParameters()*

Example

```
DeviceParameters DevPar;
char ApiErrorString[200];

// to call DeviceChangeParameters method you must be in
// ChangeParameters state
if( ChangeParameters( m_DeviceID ) )
{
    // Set the MICR+OCR font override *before* calling the
    // SetDeviceParameters()
    if( m_DevPar.bMICREnable )
    {
        SetMicrOcrFontOverride( m_DeviceID, bMicrOcrFontOverride );
    }

    if( !SetDeviceParameters( m_DeviceID, DevPar ) )
    {
        // Report error
        GetApiErrorString( ApiErrorString, 200 );
        MessageBox(...);
        ...
    }
    ...
}
```

SetDeviceParameters

BOOL SetDeviceParameters(**DWORD DeviceID**, **DeviceParameters DeviceParam**)

Set the Device parameter structure containing the processing options setting in the API (see the chapter about the DeviceParameters structure).

In Multi Document devices, this function overwrites the parameters of all the documents' types.

Valid in State : **DeviceChangeParameters**

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

DeviceParameters **DeviceParam** – the structure containing all the processing settings.

Return Value : FALSE if an error occurred.

Example

```
DeviceParameters DevPar;  
char ApiErrorString[200];  
  
// to call DeviceChangeParameters method you must be in  
// ChangeParameters state  
if( ChangeParameters( m_DeviceID ) )  
{  
    if( !SetDeviceParameters( m_DeviceID, DevPar ) )  
    {  
        // Report error  
        GetApiErrorString( ApiErrorString, 200 );  
        MessageBox(...);  
        ...  
    }  
    ...  
}
```

GetDeviceParameters

BOOL *GetDeviceParameters*(DWORD DeviceID, DeviceParameters *DeviceParam)

Retrieve the actual Device parameter structure (see the chapter about the DeviceParameters structure).

Valid in State : *DeviceChangeParameters*

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

DeviceParameters ***DeviceParam** – the pointer to the structure in which all the actual processing settings will be stored.

Return Value : FALSE if an error occurred.

SetDocProcessingParam

BOOL *SetDocProcessingParam*(DWORD DeviceID, DWORD DocType,
DWORD Parameter, void* pValue, DWORD Length)

Set a specific processing parameter (front resolution, front color, rear resolution, rear color, etc.) for a specific type of document (check, page, rigid).

When calling this function only basic parameters compatibility is tested. The complete one is carried out in the **Online** function, which can fail due to an erroneous usage of the **SetDocProcessingParam** function.

Valid in State : **DeviceChangeParameters**

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

DWORD DocType – Type of document (check, page, rigid) for which to set the parameter

DWORD Parameter – Parameter to set (front resolution, front color, rear resolution, rear color, etc.)

void pValue – Value to set the related parameter with

DWORD Length – Length of the parameter

The DocType can be one of the following value:

DocType	Description
DOC_TYPE_CHECK	This value identifies Check documents
DOC_TYPE_CARD	This value identifies Rigid Card documents
DOC_TYPE_PAGE	This value identifies Page (A4/Letter/Legal/etc) documents

The Parameter that can be set are the following (refer to the description of the structures DeviceParameters, IMAGE_PROPERTIES and SNIPPET_PROPERTIES for more details):

Parameter	Applies to	pValue refers to	Length	Description
VAPI_DP_MICR_ENABLE	DocType	DeviceParameters.bMICREnable	4	Enables MICR reading
VAPI_DP_MICR_FONT	DocType	DeviceParameters.nMICRFont	4	Specifies MICR font type
VAPI_DP_MICR_SPACING	DocType	DeviceParameters.nMICRSpaces	4	Specifies the spacing for MICR codelines
VAPI_DP_MICR_SAVE_SAMPLES	DocType	DeviceParameters.bMICRSaveSamples	4	Enables saving the MICR samples
VAPI_DP_MICR_REJECT_SYMBOL	DocType	DeviceParameters.cRejectSymbol	4	Specifies the reject symbol for MICR codelines
VAPI_DP_MICR_RESERVED1	DocType	DeviceParameters.nReserved	4	Must be always 0
VAPI_DP_MICR_RESERVED2	DocType	DeviceParameters.bReserved	4	Must be always 0
VAPI_DP_FRONT_1_FMT	DocType	DeviceParameters.ImagePropertiesFront1.Format	4	Specifies the format for the first front image
VAPI_DP_FRONT_1_PAGING	DocType	DeviceParameters.ImagePropertiesFront1.Paging	4	Specifies the paging for the first front image

Panini Vision API 4.5.0 Reference Manual Revision 1

VAPI_DP_FRONT_1_RESOLUTION	DocType	DeviceParameters.ImagePropertiesFront1.Resolution	4	Specifies the resolution for the first front image
VAPI_DP_FRONT_1_CLR_DEPTH	DocType	DeviceParameters.ImagePropertiesFront1.ColorDepth	4	Specifies the color depth for the first front image
VAPI_DP_FRONT_1_QUALITY	DocType	DeviceParameters.ImagePropertiesFront1.Threshold	4	Specifies the quality for the first front image
VAPI_DP_FRONT_2_FMT	DocType	DeviceParameters.ImagePropertiesFront2.Format	4	Specifies the format for the second front image
VAPI_DP_FRONT_2_PAGING	DocType	DeviceParameters.ImagePropertiesFront2.Paging	4	Specifies the paging for the second front image
VAPI_DP_FRONT_2_RESOLUTION	DocType	DeviceParameters.ImagePropertiesFront2.Resolution	4	Specifies the resolution for the second front image
VAPI_DP_FRONT_2_CLR_DEPTH	DocType	DeviceParameters.ImagePropertiesFront2.ColorDepth	4	Specifies the color depth for the second front image
VAPI_DP_FRONT_2_QUALITY	DocType	DeviceParameters.ImagePropertiesFront2.Threshold	4	Specifies the quality for the second front image
VAPI_DP_REAR_1_FMT	DocType	DeviceParameters.ImagePropertiesRear1.Format	4	Specifies the format for the first rear image
VAPI_DP_REAR_1_PAGING	DocType	DeviceParameters.ImagePropertiesRear1.Paging	4	Specifies the paging for the first rear image
VAPI_DP_REAR_1_RESOLUTION	DocType	DeviceParameters.ImagePropertiesRear1.Resolution	4	Specifies the resolution for the first rear image
VAPI_DP_REAR_1_CLR_DEPTH	DocType	DeviceParameters.ImagePropertiesRear1.ColorDepth	4	Specifies the color depth for the first rear image
VAPI_DP_REAR_1_QUALITY	DocType	DeviceParameters.ImagePropertiesRear1.Threshold	4	Specifies the quality for the first rear image
VAPI_DP_REAR_2_FMT	DocType	DeviceParameters.ImagePropertiesRear2.Format	4	Specifies the format for the second rear image
VAPI_DP_REAR_2_PAGING	DocType	DeviceParameters.ImagePropertiesRear2.Paging	4	Specifies the paging for the second rear image
VAPI_DP_REAR_2_RESOLUTION	DocType	DeviceParameters.ImagePropertiesRear2.Resolution	4	Specifies the resolution for the second rear image
VAPI_DP_REAR_2_CLR_DEPTH	DocType	DeviceParameters.ImagePropertiesRear2.ColorDepth	4	Specifies the color depth for the second rear image
VAPI_DP_REAR_2_QUALITY	DocType	DeviceParameters.ImagePropertiesRear2.Threshold	4	Specifies the quality for the second rear image
VAPI_DP_PRINTER_ENABLE	DOC_TYPE_CHEQUE, DOC_TYPE_CARD, DOC_TYPE_PAGE	DeviceParameters.bPrintEnable	4	Enables printing
VAPI_DP_PRINTER_TRAILING_EDGE	DOC_TYPE_CHEQUE, DOC_TYPE_CARD, DOC_TYPE_PAGE	n/a	4	Enables printing based on the trailing edge of the document (it is by default based on the leading edge).
VAPI_DP_PRINTER_AGP_QUALITY	DOC_TYPE_CHEQUE, DOC_TYPE_CARD, DOC_TYPE_PAGE	n/a	4	Specifies the AGP quality

Panini Vision API 4.5.0

Reference Manual Revision 1

	PAGE			
VAPI_DP_PRINTER_SMART_JET	DOC_TYPE _CHEQUE, DOC_TYPE _CARD, DOC_TYPE _PAGE	n/a	4	Enables Smart Jet
VAPI_DP_PRINTER_VIRTUAL	DOC_TYPE _CHEQUE, DOC_TYPE _CARD, DOC_TYPE _PAGE	n/a	4	Enables Virtual Endorsement
VAPI_DP_ONE_DOC	DOC_TYPE _CHEQUE, DOC_TYPE _CARD, DOC_TYPE _PAGE	DeviceParameters.bOne Doc	4	Enables One Doc mode
VAPI_DP_SNP01_ENABLE	DocType	DeviceParameters.SnippetProperties[0].Enable	4	Enables Snippet № 1
VAPI_DP_SNP01_SIDE	DocType	DeviceParameters.SnippetProperties[0].Front	4	Specifies the side of Snippet № 1
VAPI_DP_SNP01_X	DocType	DeviceParameters.SnippetProperties[0].Properties.Xposition	4	Specifies the horizontal starting position of Snippet № 1
VAPI_DP_SNP01_Y	DocType	DeviceParameters.SnippetProperties[0].Properties.Yposition	4	Specifies the vertical starting position of Snippet № 1
VAPI_DP_SNP01_WIDTH	DocType	DeviceParameters.SnippetProperties[0].Properties.Width	4	Specifies the width of Snippet № 1
VAPI_DP_SNP01_HEIGHT	DocType	DeviceParameters.SnippetProperties[0].Properties.Height	4	Specifies the height of Snippet № 1
VAPI_DP_SNP01_ORIENTATION	DocType	DeviceParameters.SnippetProperties[0].Properties.Orientation	4	Specifies the orientation of Snippet № 1
VAPI_DP_SNP01_COLOR	DocType	DeviceParameters.SnippetProperties[0].Properties.Color	4	Specifies the color format of Snippet № 1
VAPI_DP_SNP01_COMPRESSION	DocType	DeviceParameters.SnippetProperties[0].Properties.Compression	4	Specifies the compression format of Snippet № 1
VAPI_DP_SNP01_MILLIMETERS	DocType	DeviceParameters.SnippetProperties[0].Properties.Millimeters	4	Specifies the unit of Snippet № 1's measures
VAPI_DP_SNP02_ENABLE	DocType	DeviceParameters.SnippetProperties[1].Enable	4	Enables Snippet № 2
VAPI_DP_SNP02_SIDE	DocType	DeviceParameters.SnippetProperties[1].Front	4	Specifies the side of Snippet № 2
VAPI_DP_SNP02_X	DocType	DeviceParameters.SnippetProperties[1].Properties.Xposition	4	Specifies the horizontal starting position of Snippet № 2
VAPI_DP_SNP02_Y	DocType	DeviceParameters.SnippetProperties[1].Properties.Yposition	4	Specifies the vertical starting position of Snippet № 2

Panini Vision API 4.5.0 Reference Manual Revision 1

VAPI_DP_SNP02_WIDTH	DocType	DeviceParameters.SnippetProperties[1].Properties.Width	4	Specifies the width of Snippet № 2
VAPI_DP_SNP02_HEIGHT	DocType	DeviceParameters.SnippetProperties[1].Properties.Height	4	Specifies the height of Snippet № 2
VAPI_DP_SNP02_ORIENTATION	DocType	DeviceParameters.SnippetProperties[1].Properties.Orientation	4	Specifies the orientation of Snippet № 2
VAPI_DP_SNP02_COLOR	DocType	DeviceParameters.SnippetProperties[1].Properties.Color	4	Specifies the color format of Snippet № 2
VAPI_DP_SNP02_COMPRESSION	DocType	DeviceParameters.SnippetProperties[1].Properties.Compression	4	Specifies the compression format of Snippet № 2
VAPI_DP_SNP02_MILLIMETERS	DocType	DeviceParameters.SnippetProperties[1].Properties.Millimeters	4	Specifies the unit of Snippet № 2's measures
VAPI_DP_SNP03_ENABLE	DocType	DeviceParameters.SnippetProperties[2].Enable	4	Enables Snippet № 3
VAPI_DP_SNP03_SIDE	DocType	DeviceParameters.SnippetProperties[2].Front	4	Specifies the side of Snippet № 3
VAPI_DP_SNP03_X	DocType	DeviceParameters.SnippetProperties[2].Properties.Xposition	4	Specifies the horizontal starting position of Snippet № 3
VAPI_DP_SNP03_Y	DocType	DeviceParameters.SnippetProperties[2].Properties.Yposition	4	Specifies the vertical starting position of Snippet № 3
VAPI_DP_SNP03_WIDTH	DocType	DeviceParameters.SnippetProperties[2].Properties.Width	4	Specifies the width of Snippet № 3
VAPI_DP_SNP03_HEIGHT	DocType	DeviceParameters.SnippetProperties[2].Properties.Height	4	Specifies the height of Snippet № 3
VAPI_DP_SNP03_ORIENTATION	DocType	DeviceParameters.SnippetProperties[2].Properties.Orientation	4	Specifies the orientation of Snippet № 3
VAPI_DP_SNP03_COLOR	DocType	DeviceParameters.SnippetProperties[2].Properties.Color	4	Specifies the color format of Snippet № 3
VAPI_DP_SNP03_COMPRESSION	DocType	DeviceParameters.SnippetProperties[2].Properties.Compression	4	Specifies the compression format of Snippet № 3
VAPI_DP_SNP03_MILLIMETERS	DocType	DeviceParameters.SnippetProperties[2].Properties.Millimeters	4	Specifies the unit of Snippet № 3's measures
VAPI_DP_SNP04_ENABLE	DocType	DeviceParameters.SnippetProperties[3].Enable	4	Enables Snippet № 4
VAPI_DP_SNP04_SIDE	DocType	DeviceParameters.SnippetProperties[3].Front	4	Specifies the side of Snippet № 4
VAPI_DP_SNP04_X	DocType	DeviceParameters.SnippetProperties[3].Properties	4	Specifies the horizontal starting position of Snippet № 4

Panini Vision API 4.5.0 Reference Manual Revision 1

		Xposition		
VAPI_DP_SNP04_Y	DocType	DeviceParameters.SnippetProperties[3].Properties.Yposition	4	Specifies the vertical starting position of Snippet № 4
VAPI_DP_SNP04_WIDTH	DocType	DeviceParameters.SnippetProperties[3].Properties.Width	4	Specifies the width of Snippet № 4
VAPI_DP_SNP04_HEIGHT	DocType	DeviceParameters.SnippetProperties[3].Properties.Height	4	Specifies the height of Snippet № 4
VAPI_DP_SNP04_ORIENTATION	DocType	DeviceParameters.SnippetProperties[3].Properties.Orientation	4	Specifies the orientation of Snippet № 4
VAPI_DP_SNP04_COLOR	DocType	DeviceParameters.SnippetProperties[3].Properties.Color	4	Specifies the color format of Snippet № 4
VAPI_DP_SNP04_COMPRESSION	DocType	DeviceParameters.SnippetProperties[3].Properties.Compression	4	Specifies the compression format of Snippet № 4
VAPI_DP_SNP04_MILLIMETERS	DocType	DeviceParameters.SnippetProperties[3].Properties.Millimeters	4	Specifies the unit of Snippet № 4's measures
VAPI_DP_SNP05_ENABLE	DocType	DeviceParameters.SnippetProperties[4].Enable	4	Enables Snippet № 5
VAPI_DP_SNP05_SIDE	DocType	DeviceParameters.SnippetProperties[4].Front	4	Specifies the side of Snippet № 5
VAPI_DP_SNP05_X	DocType	DeviceParameters.SnippetProperties[4].Properties.Xposition	4	Specifies the horizontal starting position of Snippet № 5
VAPI_DP_SNP05_Y	DocType	DeviceParameters.SnippetProperties[4].Properties.Yposition	4	Specifies the vertical starting position of Snippet № 5
VAPI_DP_SNP05_WIDTH	DocType	DeviceParameters.SnippetProperties[4].Properties.Width	4	Specifies the width of Snippet № 5
VAPI_DP_SNP05_HEIGHT	DocType	DeviceParameters.SnippetProperties[4].Properties.Height	4	Specifies the height of Snippet № 5
VAPI_DP_SNP05_ORIENTATION	DocType	DeviceParameters.SnippetProperties[4].Properties.Orientation	4	Specifies the orientation of Snippet № 5
VAPI_DP_SNP05_COLOR	DocType	DeviceParameters.SnippetProperties[4].Properties.Color	4	Specifies the color format of Snippet № 5
VAPI_DP_SNP05_COMPRESSION	DocType	DeviceParameters.SnippetProperties[4].Properties.Compression	4	Specifies the compression format of Snippet № 5
VAPI_DP_SNP05_MILLIMETERS	DocType	DeviceParameters.SnippetProperties[4].Properties.Millimeters	4	Specifies the unit of Snippet № 5's measures
VAPI_DP_SNP06_ENABLE	DocType	DeviceParameters.SnippetProperties[5].Enable	4	Enables Snippet № 6
VAPI_DP_SNP06_SIDE	DocType	DeviceParameters.Snippet	4	Specifies the side of

Panini Vision API 4.5.0 Reference Manual Revision 1

		petProperties[5].Front		Snippet № 6
VAPI_DP_SNP06_X	DocType	DeviceParameters.Snip petProperties[5].Prop erties. Xposition	4	Specifies the horizontal starting position of Snippet № 6
VAPI_DP_SNP06_Y	DocType	DeviceParameters.Snip petProperties[5].Prop erties. Yposition	4	Specifies the vertical starting position of Snippet № 6
VAPI_DP_SNP06_WIDTH	DocType	DeviceParameters.Snip petProperties[5].Prop erties. Width	4	Specifies the width of Snippet № 6
VAPI_DP_SNP06_HEIGHT	DocType	DeviceParameters.Snip petProperties[5].Prop erties. Height	4	Specifies the height of Snippet № 6
VAPI_DP_SNP06_ORIENTATION	DocType	DeviceParameters.Snip petProperties[5].Prop erties. Orientation	4	Specifies the orientation of Snippet № 6
VAPI_DP_SNP06_COLOR	DocType	DeviceParameters.Snip petProperties[5].Prop erties. Color	4	Specifies the color format of Snippet № 6
VAPI_DP_SNP06_COMPRESSION	DocType	DeviceParameters.Snip petProperties[5].Prop erties. Compression	4	Specifies the compression format of Snippet № 6
VAPI_DP_SNP06_MILLIMETERS	DocType	DeviceParameters.Snip petProperties[5].Prop erties. Millimeters	4	Specifies the unit of Snippet № 6's measures
VAPI_DP_SNP07_ENABLE	DocType	DeviceParameters.Snip petProperties[6].Enab le	4	Enables Snippet № 7
VAPI_DP_SNP07_SIDE	DocType	DeviceParameters.Snip petProperties[6].Fron t	4	Specifies the side of Snippet № 7
VAPI_DP_SNP07_X	DocType	DeviceParameters.Snip petProperties[6].Prop erties. Xposition	4	Specifies the horizontal starting position of Snippet № 7
VAPI_DP_SNP07_Y	DocType	DeviceParameters.Snip petProperties[6].Prop erties. Yposition	4	Specifies the vertical starting position of Snippet № 7
VAPI_DP_SNP07_WIDTH	DocType	DeviceParameters.Snip petProperties[6].Prop erties. Width	4	Specifies the width of Snippet № 7
VAPI_DP_SNP07_HEIGHT	DocType	DeviceParameters.Snip petProperties[6].Prop erties. Height	4	Specifies the height of Snippet № 7
VAPI_DP_SNP07_ORIENTATION	DocType	DeviceParameters.Snip petProperties[6].Prop erties. Orientation	4	Specifies the orientation of Snippet № 7
VAPI_DP_SNP07_COLOR	DocType	DeviceParameters.Snip petProperties[6].Prop erties. Color	4	Specifies the color format of Snippet № 7
VAPI_DP_SNP07_COMPRESSION	DocType	DeviceParameters.Snip petProperties[6].Prop erties. Compression	4	Specifies the compression format of Snippet № 7
VAPI_DP_SNP07_MILLIMETERS	DocType	DeviceParameters.Snip petProperties[6].Prop erties.	4	Specifies the unit of Snippet № 7's measures

Panini Vision API 4.5.0 Reference Manual Revision 1

		Millimeters		
VAPI_DP_SNP08_ENABLE	DocType	DeviceParameters.SnippetProperties[7].Enable	4	Enables Snippet № 8
VAPI_DP_SNP08_SIDE	DocType	DeviceParameters.SnippetProperties[7].Front	4	Specifies the side of Snippet № 8
VAPI_DP_SNP08_X	DocType	DeviceParameters.SnippetProperties[7].Properties.Xposition	4	Specifies the horizontal starting position of Snippet № 8
VAPI_DP_SNP08_Y	DocType	DeviceParameters.SnippetProperties[7].Properties.Yposition	4	Specifies the vertical starting position of Snippet № 8
VAPI_DP_SNP08_WIDTH	DocType	DeviceParameters.SnippetProperties[7].Properties.Width	4	Specifies the width of Snippet № 8
VAPI_DP_SNP08_HEIGHT	DocType	DeviceParameters.SnippetProperties[7].Properties.Height	4	Specifies the height of Snippet № 8
VAPI_DP_SNP08_ORIENTATION	DocType	DeviceParameters.SnippetProperties[7].Properties.Orientation	4	Specifies the orientation of Snippet № 8
VAPI_DP_SNP08_COLOR	DocType	DeviceParameters.SnippetProperties[7].Properties.Color	4	Specifies the color format of Snippet № 8
VAPI_DP_SNP08_COMPRESSION	DocType	DeviceParameters.SnippetProperties[7].Properties.Compression	4	Specifies the compression format of Snippet № 8
VAPI_DP_SNP08_MILLIMETERS	DocType	DeviceParameters.SnippetProperties[7].Properties.Millimeters	4	Specifies the unit of Snippet № 8's measures
VAPI_DP_SNP09_ENABLE	DocType	DeviceParameters.SnippetProperties[8].Enable	4	Enables Snippet № 9
VAPI_DP_SNP09_SIDE	DocType	DeviceParameters.SnippetProperties[8].Front	4	Specifies the side of Snippet № 9
VAPI_DP_SNP09_X	DocType	DeviceParameters.SnippetProperties[8].Properties.Xposition	4	Specifies the horizontal starting position of Snippet № 9
VAPI_DP_SNP09_Y	DocType	DeviceParameters.SnippetProperties[8].Properties.Yposition	4	Specifies the vertical starting position of Snippet № 9
VAPI_DP_SNP09_WIDTH	DocType	DeviceParameters.SnippetProperties[8].Properties.Width	4	Specifies the width of Snippet № 9
VAPI_DP_SNP09_HEIGHT	DocType	DeviceParameters.SnippetProperties[8].Properties.Height	4	Specifies the height of Snippet № 9
VAPI_DP_SNP09_ORIENTATION	DocType	DeviceParameters.SnippetProperties[8].Properties.Orientation	4	Specifies the orientation of Snippet № 9
VAPI_DP_SNP09_COLOR	DocType	DeviceParameters.SnippetProperties[8].Properties.Color	4	Specifies the color format of Snippet № 9
VAPI_DP_SNP09_COMPRESSION	DocType	DeviceParameters.SnippetProperties[8].Properties	4	Specifies the compression format of Snippet № 9

Panini Vision API 4.5.0 Reference Manual Revision 1

		erties. Compression		
VAPI_DP_SNP09_MILLIMETERS	DocType	DeviceParameters.SnippetProperties[8].Properties.Millimeters	4	Specifies the unit of Snippet № 9's measures
VAPI_DP_SNP10_ENABLE	DocType	DeviceParameters.SnippetProperties[9].Enable	4	Enables Snippet № 10
VAPI_DP_SNP10_SIDE	DocType	DeviceParameters.SnippetProperties[9].Front	4	Specifies the side of Snippet № 10
VAPI_DP_SNP10_X	DocType	DeviceParameters.SnippetProperties[9].Properties.Xposition	4	Specifies the horizontal starting position of Snippet № 10
VAPI_DP_SNP10_Y	DocType	DeviceParameters.SnippetProperties[9].Properties.Yposition	4	Specifies the vertical starting position of Snippet № 10
VAPI_DP_SNP10_WIDTH	DocType	DeviceParameters.SnippetProperties[9].Properties.Width	4	Specifies the width of Snippet № 10
VAPI_DP_SNP10_HEIGHT	DocType	DeviceParameters.SnippetProperties[9].Properties.Height	4	Specifies the height of Snippet № 10
VAPI_DP_SNP10_ORIENTATION	DocType	DeviceParameters.SnippetProperties[9].Properties.Orientation	4	Specifies the orientation of Snippet № 10
VAPI_DP_SNP10_COLOR	DocType	DeviceParameters.SnippetProperties[9].Properties.Color	4	Specifies the color format of Snippet № 10
VAPI_DP_SNP10_COMPRESSION	DocType	DeviceParameters.SnippetProperties[9].Properties.Compression	4	Specifies the compression format of Snippet № 10
VAPI_DP_SNP10_MILLIMETERS	DocType	DeviceParameters.SnippetProperties[9].Properties.Millimeters	4	Specifies the unit of Snippet № 10's measures
VAPI_DP_FEEDING_MODE	DOC_TYPE_CHEQUE, DOC_TYPE_CARD, DOC_TYPE_PAGE	DeviceParameters.nFeedingMode	4	Specifies the feeding mode

Return Value : FALSE if an error occurred.

Example

```
DeviceParameters PageDevPar:
char ApiErrorString[200];

.....
PageDevPar.bMICREnable = FALSE;
.....

// to call SetDocProcessingParam method you must be in
// the ChangeParameters state
if( ChangeParameters( m_DeviceID ) )
```

```
{
    if( !SetDocProcessingParam(m_DeviceID, DOC_TYPE_PAGE,
        VAPI_DP_MICR_ENABLE, &(PageDevPar.bMICREnable), 4))
    {
        // Report error
        GetApiErrorString( ApiErrorString, 200 );
        MessageBox(...);
        ...
    }
    ...
}
```

For an exhaustive examples see the Barebones' class `WiBareBonesDlg.cpp`.

SetImageAdjustment

BOOL SetImageAdjustment(DWORD DeviceID, int Contrast, int Brightness, BOOL Front)

Set the parameters for the image adjustment. These parameters affect the result of the image acquisition. Valid values for Contrast and Brightness are from -100 to 100. The API default is zero for both, which means there is no correction applied.

NOTICE: This function is for special image requirements. It should not be used in “normal” contest.

Valid in State : *DeviceChangeParameters, DeviceOnLine*

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

int Contrast – image contrast adjustment

int Brightness – image brightness adjustment

BOOL Front – if TRUE the adjustment is related to the front image, FALSE to the rear image.

Return Value : FALSE if an error occurred.

Example

```
// Set 10% of contrast correction on the front
if( !SetImageAdjustment( m_DeviceID, 10, 0, TRUE ) )
{
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox(...);
    .....
}
```

SetMaxDpm

BOOL *SetMaxDpm*(DWORD DeviceID, int Dpm)

This function is intended for demo purposes. Set the DPM value (according to the max available for that device, if there is an attempt to increase this number above the max possible an error is returned, the max available DPM is stored in the IDCard). Use *GetMaxDpm* function to retrieve the actual DPM setting. The available DPM values are the following:

DPM	Doc length independent	Doc length dependent Referred to 6" (152 mm) doc length
30	X	
60	X	
90	X	
50		X
75		X
100		X
125		X

The DPM dependent on doc length means that the documents throughput is declared using the reference document length. The reference document length is 6" (152 mm). This means that a shorter document results in a higher DPM. A longer one results in a lower DPM. The MyVisionX supports 30, 60 and 90. VisionX support 50, 75 and 100 DPM. The VisionS supports each single DPM value up to 200 (maximum).

Valid in State : *DeviceChangeParameters, DeviceOffLine, DeviceOnLine*
State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

int **Dpm** – the number of Dpm to be set.

Return Value : FALSE if an error occurred.

Example

```
if( !SetMaxDpm( m_DeviceID, 30 ) )
{
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox(...);
    .....
}
```


GetMaxDpm

BOOL *GetMaxDpm*(DWORD DeviceID, int *Dpm)

Get the actual Dpm value. Use *SetMaxDpm* function to change the actual Dpm setting.

Valid in State : *DeviceChangeParameters, DeviceOffLine, DeviceOnLine*

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

 int ***Dpm** – the pointer to the variable in which the number of DPM will be returned.

Return Value : FALSE if an error occurred.

Example

```
int Dpm;
// To use a 90 DPM device as a 60 DPM one.

.....
if( !GetMaxDpm( m_DeviceID, &Dpm ) )
{
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox(...);

    .....
}
if(Dpm == 90)
{
    SetMaxDpm(m_DeviceID, 60)
}
```

SetAGPLines

BOOL *SetAGPLines*(DWORD DeviceID, BYTE Lines)

This function sets the enabled lines for the AGP printer (according to the max available for that device, if there is an attempt to increase this number above the max possible an error is returned, the max available lines is stored in the IDCard). Use *GetIDCardDescription* function to retrieve the actual enabled lines. Valid values for AGP lines are defined in the header file. This function is intended for demo purposes.

Valid in State : *DeviceChangeParameters*

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

BYTE Lines – the number of lines to be set.

Return Value : FALSE if an error occurred.

Example

```
// Downgrade an AGP 4 lines to 2 lines
if( !SetAGPLines( m_DeviceID, AGP_2_LINES ) )
{
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox(...);
    .....
}
```

SetFeederLimit

BOOL SetFeederLimit(**DWORD DeviceID**, **BYTE Limit**)

This function sets the feeder limitation (according to the max available for that device, if there is an attempt to increase this number above the max possible an error is returned, the max available lines is stored in the IDCard). Use **GetIDCardDescription** function to retrieve the actual feeder limit. Valid values for Limit are defined in the header file.

This function is intended for demo purposes.

Valid in State : **DeviceChangeParameters**

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

BYTE Limit – the feeder limit.

Return Value : FALSE if an error occurred.

Example

```
// Downgrade a FULL feeder device to a SMALL feeder
if( !SetFeederLimit( m_DeviceID, FEEDER_SMALL ) )
{
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox(...);
    .....
}
```

SetHandDropDly

BOOL *SetHandDropDly*(DWORD DeviceID, DWORD Dly)

When the feeder is in Hand-Drop mode and the device is in Feeding state, when a document is detected in the feeder, the device waits a certain delay and then feeds the document in the track. This function can modify this delay.

The Delay range is from 100 to 60000 ms. The API default is 100 ms.

Valid in State : *DeviceChangeParameters*

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

 DWORD **Dly** – The delay expressed in ms.

Return Value : FALSE if an error occurred.

Example

```
// Set a delay of 1 second
if( !SetHandDropDly ( m_DeviceID, 1000 ) )
{
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox(...);
}
```

GetAvailablePockets

BOOL *GetAvailablePockets*(DWORD DeviceID, BYTE *pucPockets)

Get the pockets installed and enabled on the device.

Valid in State : *DeviceChangeParameters*

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

 BYTE ***pucPockets** – Destination buffer where the available pockets are returned

Return Value : FALSE if an error occurred.

Example

```
// Get the available pockets
```

```
BYTE Pockets = 0;
ChangeParameters( DeviceID );
if( !GetAvailablePockets( DeviceID, &Pockets ) )
{
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox(...);
}
```

SetMaxPocket

BOOL *SetMaxPocket*(DWORD DeviceID, BYTE Pocket)

This function permits to temporary downgrade a 2 pockets device to a 1 pocket device. It doesn't update the IDCard. It's intended for Demo purposes.

Valid in State : *DeviceChangeParameters*

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

BYTE Pocket – Pocket number (1 or 2)

Return Value : FALSE if an error occurred.

Example

```
// Get the available pockets
BYTE Pockets = 0;
ChangeParameters( DeviceID );
if( !GetAvailablePockets( DeviceID, &Pockets ) )
{
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox(...);
}
else
{
    if( Pockets == 2 )
    {
        if( !SetMaxPocket( DeviceID, 2 ) )
        {
            // Report error
            GetApiErrorString( ApiErrorString, 200 );
            MessageBox(...);
        }
    }
}
```

SetFeedingMode

BOOL *SetFeedingMode*(DWORD DeviceID, BYTE Mode)

This function sets the feeding mode of the device. It is supported by VisionS only.

There are two working modes:

- Start/Stop
The device has the capability to wait for the pocket setting, even if the application takes a lot of time to decide the pocket. In case of long delays the document is stopped in the View station and it will be restarted after the SetPocket call.
A document is fed only when the previous one has a destination pocket assigned. This means that the firmware feed a new doc when the application calls SetPocket to assign a destination to the previous one.
- Flow-mode
The machine never stop the document in the view station and if the application takes too much time to decide the pocket, the device stops the job signalling document without destination or, if a default pocket is defined, sends the document to it ignoring eventual delayed SetPocket call.
A document is fed following the DPM throughput. This means that firmware feed documents at a fixed throughput without waiting for the SetPocket call.

Valid in State : *DeviceChangeParameters*

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

 BYTE **Mode** – It's the feeding mode. The options are defined in the header file.

Return Value : FALSE if an error occurred.

Example

```
// Get the available pockets
BYTE Pockets = 0;
ChangeParameters( DeviceID );
if( !SetFeedingMode( DeviceID, FEEDING_STARTSTOP ) )
{
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox(...);
}
```

SetVirtualEndorsementProperty

BOOL *SetVirtualEndorsementProperty*(DWORD DeviceID, DWORD AreaID,
DWORD PropID, LPVOID pData)

This function gets the value of a property field of an endorsement area. The application is able to set all the area properties, even the predefined ones (Payee, BOFD and Transit).

Valid in State : *DeviceChangeParameters, DeviceOnline*

State Transition : None

Arguments: **DWORD DeviceID** – the Identification number of the device.
 DWORD AreaID – It's the area ID (see VApiInterface.h).
 DWORD PropID – It's the property ID (see VApiInterface.h).
 LPVOID pData – It's the user allocated buffer that contains the data.

Return Value: FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example:

```
ChangeParameters( DeviceID );  
  
// Rectangle  
RECT Area = {1000,4250,2000,625};  
SetVirtualEndorsementAreaProperty( DeviceID, ENDORSEMENT_AREA_BOFD,  
                                   ENDORSEMENT_PARAM_RECT, (LPVOID)&Area );
```

GetVirtualEndorsementProperty

BOOL *GetVirtualEndorsementProperty*(DWORD DeviceID, DWORD AreaID,
DWORD PropID, LPVOID pData)

This function gets the value of a property field of an endorsement area.

Valid in State : *DeviceChangeParameters, DeviceOnline*

State Transition : None

Arguments: **DWORD DeviceID** – the Identification number of the device.
 DWORD AreaID – It's the area ID (see VApiInterface.h).
 DWORD PropID – It's the property ID.
 LPVOID pData – It's the user allocated buffer that receives the data.

Return Value: FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example:

```
ChangeParameters( DeviceID );  
  
// Rectangle  
RECT Area;  
GetVirtualEndorsementAreaProperty( DeviceID, ENDORSEMENT_AREA_BOFD,
```

Panini Vision API 4.5.0 Reference Manual Revision 1

```
                                ENDORSEMENT_PARAM_RECT, (LPVOID)&Area );

// Font
LOGFONT font;
memset( &font, 0, sizeof(font) );

GetVirtualEndorsementAreaProperty( DeviceID, ENDORSEMENT_AREA_BOFD,
                                ENDORSEMENT_PARAM_FONT, (LPVOID)&font );

// Color
COLORREF Color;
GetVirtualEndorsementAreaProperty( DeviceID, ENDORSEMENT_AREA_BOFD,
                                ENDORSEMENT_PARAM_COLOR, (LPVOID)&Color
                                );

// Rotation
DWORD Rot;
GetVirtualEndorsementAreaProperty( DeviceID, ENDORSEMENT_AREA_BOFD,
                                ENDORSEMENT_PARAM_ROTATION, (LPVOID)&Rot
                                );

// Flags - Left horizontal alignment and word-wrap on
DWORD Flags;
GetVirtualEndorsementAreaProperty( DeviceID, ENDORSEMENT_AREA_BOFD,
                                ENDORSEMENT_PARAM_FLAGS, (LPVOID)&Flags
                                );
```

SetIQAParameter

BOOL *SetIQAParameter*(DWORD DeviceID, UINT ImageNumber,
DWORD Test, DWORD Parameter, int Value)

This function sets the IQA parameters. It can set one parameter per time.
This function has to be called during the ChangeParameters state.

The IQA parameters are organized according to the following schema:

- Image number: one of the four compressed image.

Image	Value
IQA_FR_1 (Front #1)	0
IQA_FR_2 (Front #2)	1
IQA_RR_1 (Rear #1)	2
IQA_RR_2 (Rear #2)	3

- Test: this select one the test for the specified image number.
The following table show all the available tests.

Test #	Test ID	Description
1	IQA_UNDERSIZE_IMAGE	Detect if the image dimensions are under a threshold
2	IQA_FOLDED_OR_TORN_DOCUMENT_CORNERS	Detect if there are folded or torn corners
3	IQA_FOLDED_OR_TORN_DOCUMENT_EDGES	Detect if there are folded or torn edges
4	IQA_DOCUMENT_FRAMING_ERROR	Detect if there is a cropping error (black lines along the document borders)
5	IQA_DOCUMENT_SKEW	Detect if the document presents a skew above a threshold
6	IQA_OVERSIZE_IMAGE	Detect if the image dimensions are above a threshold
7	IQA_PIGGY_BACK_DOCUMENT	Detect if there is a multiple document feed
8	IQA_IMAGE_TOO_LIGHT	Detect if an image is too light
9	IQA_IMAGE_TOO_DARK	Detect if an image is too dark
10	IQA_HORIZONTAL_STREAKS	Detect the presence of horizontal streaks

Panini Vision API 4.5.0
Reference Manual Revision 1

11	IQA_BELOW_MINIMUM_COMPRESSED_IMAGE_SIZE	Detect if the compressed size is under a threshold
12	N/A	-
13	IQA_ABOVE_MAXIMUM_COMPRESSED_IMAGE_SIZE	Detect if the compressed size is above a threshold
14	N/A	-
15	IQA_SPOT_NOISE	Detect if spot noise is present above a threshold (It's applicable on B&W image only)
16	IQA_FRONT_REAR_DIMENSIONS_MISMATCH	Detect different dimensions between front and rear images
17	IQA_CARBON_STRIP	Detect the presence of a carbon strip
18	IQA_OUT_OF_FOCUS	Detect out of focus images (It's applicable on gray levels image only)
19	IQA_DISABLE	Disable the IQA process
20	IQA_ENABLE	Enable the IQA process

- **Parameter:** this select one of the parameter related to the selected Test.
Each test has at list one parameter in order to enable or disable itself. This kind of parameters can assume two values : TRUE of FALSE
Most of the tests have more than one parameter in order to set the threshold for the detection of the image defects.
The following table shows all the available parameters for each test.

Test #	Parameters	Unit	Description
1	IQA_UNDERSIZE_IMAGE_ENABLE IQA_MINIMUM_IMAGE_HEIGHT_THRESHOLD IQA_MINIMUM_IMAGE_WIDTH_THRESHOLD	Tenths of inches	They are the minimum acceptable dimensions
2	IQA_FOLDED_OR_TORN_DOCUMENT_CORNERS_ENABLE IQA_MAXIMUM_BOTTOM_LEFT_CORNER_FOLD_TEAR_HEIGHT_THRESHOLD IQA_MAXIMUM_BOTTOM_LEFT_CORNER_FOLD_TEAR_WIDTH_THRESHOLD IQA_MAXIMUM_BOTTOM_RIGHT_CORNER_FOLD_TEAR_HEIGHT_THRESHOLD IQA_MAXIMUM_BOTTOM_RIGHT_CORNER_FOLD_TEAR_WIDTH_THRESHOLD IQA_MAXIMUM_TOP_LEFT_CORNER_FOLD_TEAR_HEIGHT_THRESHOLD IQA_MAXIMUM_TOP_LEFT_CORNER_FOLD_TEAR_WIDTH_THRESHOLD IQA_MAXIMUM_TOP_RIGHT_CORNER_FOLD_TEAR_HEIGHT_THRESHOLD IQA_MAXIMUM_TOP_RIGHT_CORNER_FOLD_TEAR_WIDTH_THRESHOLD	Tenths of inches	They set the maximum acceptable dimensions of a corner fold for each document corner
3	IQA_FOLDED_OR_TORN_DOCUMENT_EDGES_ENABLE IQA_MAXIMUM_BOTTOM_EDGE_HEIGHT IQA_MAXIMUM_BOTTOM_EDGE_WIDTH IQA_MAXIMUM_LEFT_EDGE_HEIGHT	Tenths of inches	They set the maximum acceptable dimensions of an edge fold

Panini Vision API 4.5.0
Reference Manual Revision 1

	IQA_MAXIMUM_LEFT_EDGE_WIDTH IQA_MAXIMUM_RIGHT_EDGE_HEIGHT IQA_MAXIMUM_RIGHT_EDGE_WIDTH IQA_MAXIMUM_TOP_EDGE_HEIGHT IQA_MAXIMUM_TOP_EDGE_WIDTH		for each document edge
4	IQA_DOCUMENT_FRAMING_ERROR_ENABLE IQA_MAXIMUM_ADDITIONAL_BOTTOM_SCAN_LINES_HEIGHT IQA_MAXIMUM_ADDITIONAL_LEFT_SCAN_LINES_WIDTH IQA_MAXIMUM_ADDITIONAL_RIGHT_SCAN_LINES_WIDTH IQA_MAXIMUM_ADDITIONAL_TOP_SCAN_LINES_HEIGHT	Tenths of inches	They set the maximum width or height of additional lines around the image
5	IQA_DOCUMENT_SKEW_ENABLE IQA_NEGATIVE_DOCUMENT_SKEW_ANGLE IQA_POSITIVE_DOCUMENT_SKEW_ANGLE	Tenths of degrees	They set the min/max acceptable skew. Ex: 300 is 30 degrees
6	IQA_OVERSIZE_IMAGE_ENABLE IQA_MAXIMUM_IMAGE_HEIGHT_THRESHOLD IQA_MAXIMUM_IMAGE_WIDTH_THRESHOLD	Tenths of inches	They are the maximum acceptable dimensions
7	IQA_PIGGY_BACK_DOCUMENT_ENABLE	-	-
8	IQA_IMAGE_TOO_LIGHT_ENABLE IQA_MINIMUM_PERCENT_BLACK_PIXELS IQA_MAXIMUM_PERCENT_BRIGHTNESS IQA_MINIMUM_PERCENT_CONTRAST	Units of 0.1 percent, e.g., 20 = 2%	The 2 nd parameter is applicable only to B&W images. The last two are applicable only to Gray Level images.
9	IQA_IMAGE_TOO_DARK_ENABLE IQA_MAXIMUM_PERCENT_BLACK_PIXELS IQA_MINIMUM_PERCENT_BRIGHTNESS	Units of 0.1 percent	The 2 nd parameter is applicable only to B&W images. The 3 rd is applicable only to Gray Level images.
10	IQA_HORIZONTAL_STREAKS_ENABLE IQA_MAX_GRAY_LEVEL_STREAK_COUNT IQA_MAX_GRAY_LEVEL_STREAK_MAX_HEIGHT IQA_STREAK_CONTRAST_THRESHOLD IQA_MAX_BLACK_STREAK_COUNT IQA_MAX_BLACK_STREAK_MAX_HEIGHT IQA_BLACK_STREAK_PERCENTAGE_THRESHOLD IQA_MAXIMUM_STREAK_HEIGHT_TO_BE_DETECTED_THRESHOLD	Height are expressed in pixels. Percentage are expressed in unit of 0.1 percent.	The 2 nd group of parameters are applicable to gray levels images. The 3 rd group to B&W images. The last parameters is the max streak height in pixel.
11	IQA_BELOW_MINIMUM_COMPRESSED_IMAGE_SIZE_ENABLE IQA_MINIMUM_BITONAL_COMPRESSED_IMAGE_SIZE IQA_MINIMUM_GRAY_LEVEL_COMPRESSED_IMAGE_SIZE	Bytes	They set the minimum compressed size for B&W

Panini Vision API 4.5.0
Reference Manual Revision 1

			and Gray level images
13	IQA_ABOVE_MAXIMUM_COMPRESSED_IMAGE_SIZE_ENABLE IQA_MAXIMUM_BITONAL_COMPRESSED_IMAGE_SIZE IQA_MAXIMUM_GRAY_LEVEL_COMPRESSED_IMAGE_SIZE	Bytes	They set the maximum compressed size for B&W and Gray level images
15	IQA_SPOT_NOISE_ENABLE IQA_MAXIMUM_AVERAGE_SPOT_NOISE_GROUPINGS_PER_SQUARE_INCH	Groups of “spot noise” per square inch	It sets the maximum acceptable spot noise density
16	IQA_FRONT_REAR_IMAGE_DIMENSION_MISMATCH_ENABLE IQA_FRONT_REAR_HEIGHT_DIFFERENCE IQA_FRONT_REAR_WIDTH_DIFFERENCE	Tenths of inches	It sets the maximum absolute value of the image Width or Height difference, between front and rear images
17	IQA_CARBON_STRIP_ENABLE IQA_MINIMUM_CARBON_STRIP_HEIGHT	Tenths of inches	It sets the minimum height of the detected carbon strip height
18	IQA_OUT_OF_FOCUS_ENABLE IQA_MINIMUM_IMAGE_FOCUS_SCORE	Score	It sets the minimum acceptable image focus score
19	-	Boolean	If TRUE, disable the whole IQA
20	-	Boolean	If TRUE enable the IQA

- Value: it sets the value of the selected Parameter.

The entire set of parameters shown in the above tables are defined in the VApiIQAInterface.h

Valid in State : **DeviceChangeParameters**

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

UINT **ImageNumber** – Image’s selector.

DWORD **Test** – Test’s selector.

DWORD **Parameter** – Parameter’s selector.

int **Value** – Parameter’s value.

Return Value : FALSE if an error occurs, call *GetIQALastError* or *GetIQALastErrorString* to get more information about it.

Example: Refer to VxBarebones source code.

SetExtendedParameter

BOOL *SetExtendedParameter*(DWORD DeviceID, UINT32 ParamID, void *pData, UINT32 Length)

This function sets the extended parameters. It can set one parameter per time. Please refer to the “Extended parameters” section for the details.

Valid in State : *DeviceChangeParameters*

State Transition : None

Arguments : DWORD **DeviceID** – It’s the Identification number of the device.

 UINT32 **ParamID** – It’s the parameter ID (VApiExtendedParameters.h).

 void ***pData** – It’s the parameter data value.

 UINT32 **Length** – It’s the parameter data length.

Return Value: FALSE if an error occurred.

Example

```
UINT32 IgnoreCropErr = EP_IP_IGNORE_CROPPING_ERROR_ENABLE;

// To call SetExtendedParameter method you must be in
// ChangeParameters state
if( ChangeParameters( DeviceID ) )
{
    BOOL bOK = SetExtendedParameter( DeviceID,
                                     EP_IP_IGNORE_CROPPING_ERROR,
                                     (void*)&IgnoreCropErr,
                                     sizeof(IgnoreCropErr) )

    if( !bOK )
    {
        // Report error
        ...
    }
    ...
}
```

GetExtendedParameter

BOOL *GetExtendedParameter*(DWORD DeviceID, UINT32 ParamID, UINT32 *pType,
void *pData, UINT32 *pLength)

This function gets the value of the extended parameters. It can set one parameter per time. Please refer to the “Extended parameters” section for the details.

Valid in State : *DeviceChangeParameters*

State Transition : None

Arguments : **DWORD DeviceID** – It’s the Identification number of the device.

UINT32 ParamID – It’s the parameter ID (VApiExtendedParameters.h).

UINT32 *pType – It’s the parameter type (VApiExtendedParameters.h).

void *pData – It’s the parameter data value.

UINT32 *pLength – It’s the parameter data length.

Return Value: FALSE if an error occurred.

Example

```
UINT32 IgnoreCropErr = 0;
UINT32 ParmType = EP_TYPE_NUM;

// To call SetExtendedParameter method you must be in
// ChangeParameters state
if( ChangeParameters( DeviceID ) )
{
    BOOL bOK = GetExtendedParameter( DeviceID,
                                     EP_IP_IGNORE_CROPPING_ERROR,
                                     &ParmType,
                                     (void*)&IgnoreCropErr,
                                     sizeof(IgnoreCropErr) )

    if( !bOK )
    {
        // Report error
        ...
    }
    ...
}
```

On Line Functions

IsFeederEmpty

BOOL *IsFeederEmpty*(DWORD DeviceID, BOOL *pFlag)

This function returns to the application the status of the feeder sensor to detect the presence of documents in the feeder. If the *pFlag* return value is TRUE, the feeder is empty. If FALSE the feeder contains one or more documents.

Valid in State : **DeviceOnLine**

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

 BOOL ***pFlag** – This parameter receive the Feeder status result.

Return Value : FALSE if an error occurs, call **GetApiError** or **GetApiErrorString** to get more information about it.

Example:

```
char ApiErrorString[200];
...
if( OnLine ( m_DeviceID ) )
{
    BOOL bFeederEmpty = FALSE;
    IsFeederEmpty(m_DeviceID, &bFeederEmpty );
    if( bFeederEmpty )
    {
        MessageBox( "...");
    }
    else if( !StartFeeding( m_DeviceID ) )
    {
        // Report error
        GetApiErrorString( ApiErrorString, 200 );
        MessageBox( "...");
    }
}
```

IsFeederEmpty (VN only)

BOOL *IsFeederEmpty*(DWORD DeviceID, DWORD FeederID, BOOL *pFlag)

This function returns to the application the status of the feeder sensor (cheque or rigid card based on selection) to detect the presence of documents in the feeder. If the *pFlag* return value is TRUE, the feeder is empty. If FALSE the feeder contains one or more documents.

This function is supported for the Vision NeXt only.

Valid in State : ***DeviceOnLine***

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

DWORD **FeederID** – the selected feeder. Possible values are:

FEEDER_CHEQUE (0)

FEEDER_RIGID (1)

BOOL ***pFlag** – this parameter receive the Feeder status result.

Return Value : FALSE if an error occurs, call ***GetApiError*** or ***GetApiErrorString*** to get more information about it.

StartFeeding

BOOL **StartFeeding**(DWORD DeviceID)

With this call the device begins to feed documents. The feeding mode is set in DeviceParameters structure (i.e. single or multiple document feeding, Main Hopper or Hand Drop mode).

Valid in State : ***DeviceOnLine***

State Transition : ***DeviceFeeding***

Arguments : DWORD **DeviceID** – the Identification number of the device.

Return Value : FALSE if an error occurs, call ***GetApiError*** or ***GetApiErrorString*** to get more information about it.

Example

```
char ApiErrorString[200];  
.....  
if( OnLine ( m_DeviceID ) )  
{  
    // Get Device State String, it must be  
    // "DeviceOnLine"  
    GetDeviceStateString( DeviceID, pcDeviceStateString,  
        100 );  
    if( !StartFeeding( m_DeviceID ) )  
    {  
        .....  
    }
```

```
        // Report error
        GetApiErrorString( ApiErrorString, 200 );
        MessageBox(...);
    }
    .....
}
```

StopFeeding

BOOL *StopFeeding*(DWORD DeviceID)

With this call the device stops feeding documents. If the Feeding mode is single document or if an exception occurs, *StopFeeding* call is not needed.

Using a 30/60 DPM device, it works as a Start/Stop device while a 90 Dpm works as a flow mode one. Calling *StopFeeding* function before setting the destination pocket, the device will stop on the last fed document for a 30 or a 60 Dpm device, while for a 90 Dpm, due to its feeding feature, the feeding will stop after the next document. For example with a 30/60 DPM device it will be possible to stop the device, if a reject has been detected on a certain document, when that document will reach the destination pocket. That won't be available on the 90 Dpm devices.

A 2 pocket machine, which is a Start/Stop device, is always capable to stop the machine on the last fed document.

Valid in State : *DeviceFeeding*

State Transition : *DeviceOnLine*

Arguments : DWORD **DeviceID** – the Identification number of the device.

Return Value : FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example

```
char ApiErrorString[200];
.....

if( !StopFeeding( m_DeviceID ) )
{
    .....
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox(...);
}
.....
```


FreeTrack

BOOL *FreeTrack*(DWORD DeviceID , UCHAR ucPocket)

With this call the device enables the transport motor (at a lower speed) to free the track from jammed documents sending them to the desired Pocket. For a 2 pocket machine it's possible to purge the track putting the documents in the second pocket. Using a 1 pocket machine the destination pocket must be 1.

Valid in State : *DeviceOnLine*

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

UCHAR ucPocket – the destination pocket in which the document will be sent.

Return Value : FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example

```
DWORD dwErrCode;
char ApiErrorString[200];
.....
// Automatic FreeTrack when a jam occurred
case WMPAR_SORTER_EXCEPTION:
    GetDeviceError( m_DeviceID, &dwErrCode );
    .....
    if( dwErrCode == DEVICE_ERR_SORTER_ERROR_PENDING)
    {
        // a jam has occurred
        if( !FreeTrack( m_DeviceID, 1 ) )
        {
            // Report error
            GetApiErrorString( ApiErrorString, 200 );
            MessageBox(...);
        }
        .....
    }
    break;
```

SetPocket

BOOL *SetPocket*(DWORD DeviceID, DWORD dwDocId, UCHAR ucPocket)

With this call the destination pocket for the current document is set, remember to call this function on WMPAR_SORTER_SET_ITEM_OUTPUT message. For a My Vision X with 1 pocket the application must set always 1. For a 2 pocket machine valid values are 1 and 2.

Valid in State : **DeviceFeeding** (SetItemOutput event)
State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

DWORD dwDocId – the Identification number of the document.

UCHAR ucPocket – it's the destination pocket in which the document will be sent.

Return Value : FALSE if an error occurs, call **GetApiError** or **GetApiErrorString** to get more information about it.

Example

```
DWORD DocNumber;  
char ApiErrorString[200];  
.....  
case WMPAR_SORTER_SET_ITEM_OUTPUT:  
    DocNumber = (DWORD) LPARAM;  
    .....  
    if( !SetPocket( m_DeviceID, DocNumber, 1 ) )  
    {  
        // Report error  
        GetApiErrorString( ApiErrorString, 200 );  
        MessageBox(...);  
    }  
    .....  
    break;
```

GetDocumentLength

BOOL GetDocumentLength(**DWORD DeviceID**, **DWORD dwDocId**,
 DWORD *pdwDocLength)

This function returns the length of the last processed document in millimeters. Due to firmware limits this has to be considered as an informative value, and the expected error tolerance is +-3mm.

Valid in State : **DeviceFeeding**
State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

DWORD dwDocId – the Identification number of the document.

DWORD * pdwDocLength – pointer to the variable where the length of the document will be written

Return Value : FALSE if an error occurs, call ***GetApiError*** or ***GetApiErrorString*** to get more information about it.

Example

```

DWORD DocNumber;
DWORD DocLen;
char ApiErrorString[200];
.....
case WMPAR_SORTER_MICR_AVAILABLE:
    DocNumber = (DWORD) LPARAM;
    .....
    if( !GetDocumentLength( m_DeviceID, DocNumber,
                           &DocLen ) )
    {
        // Report error
        GetApiErrorString( ApiErrorString, 200 );
        MessageBox (...);
    }

    break;










```

GetMicrCodeline

BOOL *GetMicrCodeline*(DWORD DeviceID, CHAR *pcDestination,
 DWORD dwMaxLength)

This function returns the last recognized MICR codeline. The character after the '\0' NULL terminator represents the font found for the recognized codeline. It will be 'E' for E13B or 'C' for CMC7 (this feature is useful only if MICR auto recognition is selected); 'V' means void string (no codeline).

The MICR field delimiters are translated as follows:

E13B			CMC7		
	transit	:		internal	:
	amount	;		terminator	;
	on-us	<		amount	<
	dash	=		routing	>
				-	=

The default reject symbols is ? if not set differently in *DeviceParameters.cRejectSymbol*.

Valid in State : *DeviceFeeding* (MicrAvailable event)

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

CHAR * **pcDestination** – the destination string for the MICR codeline.

DWORD **dwMaxLength** – Length of the user allocated buffer pointed to by pcDestination

Return Value : FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example

```
char MICRCodeline[80];
char ApiErrorString[200];
char MICRFont;
.....
case WMPAR_SORTER_MICR_AVAILABLE:
    // MICRCodeline string will contain the recognized
    codeline
    if( !GetMicrCodeline( m_DeviceID, MICRCodeline, 80 ) )
    {
        // Report error
        GetApiErrorString( ApiErrorString, 200 );
        MessageBox(...);
    }
    else
    {
        // In auto mode to understand the font type
        swith( MICRCodeline[strlen( MICRCodeline ) + 1] )
        {
            case 'C':
                // CMC7 codeline
                .....
            case 'E':
                // E13B codeline
                .....
            case 'V':
                // void codeline
                .....
            }
        }
        .....
        break;
```

GetOCRCodeline

BOOL *GetOCRCodeline* GetOCRCodeline(DWORD DeviceID, BYTE *pBmp,
char *pDestString, DWORD StringLen,
int Font, int Threshold)

This function returns the recognized OCR codeline and Barcode. It is called to decode a memory bitmap with the OCR engine (including Bitmap File Header and Bitmap Info Header).

The OCR and Barcode available fonts are:

OCR1_OCRA	0x0001	: OCR-A Euro limited
OCR1_OCRB	0x0002	: OCR-B Euro limited
OCR1_OCRAB	0x0003	: OCR-A and OCR-B Euro limited
OCR1_OCRBUK	0x0004	: OCR-B extended for UK banking
OCR1_E13BO	0x0005	: E13B optical
OCR1_E13BOXOCRB	0x0006	: E13B + OCRB for UK, switched on 'X'
OCR1_OCRAALNUM	0x0007	: OCR A alphanumeric
OCR1_OCRBALNUM	0x0008	: OCR B alphanumeric
OCR1_OCRB1403	0x0009	: OCRB 1403M
OCR1_OCRAN	0x000A	: OCRA numeric for use with OCRB1403
OCR1_BC128	0x0010	: Barcode 1D Code 128
OCR1_BC39	0x0011	: Barcode 1D Code 39
OCR1_BC2OF5	0x0012	: Barcode 1D Interleaved 2 of 5
OCR1_BCUPCA	0x0013	: Barcode 1D UPCA
OCR1_BCEAN13	0x0014	: Barcode 1D EAN 13
OCR1_BCUPCE	0x0015	: Barcode 1D UPCE
OCR1_BCEAN8	0x0016	: Barcode 1D EAN 8
OCR1_BCPDF417	0x0017	: Barcode 2D PDF417
OCR1_CMC7O	0x0018	: CMC7 Optical
OCR1_BCCODABAR	0x0019	: Barcode 1D CODABAR (aka Code 2 of 7, Codeabar, Ames Code, NW-7 and Monarch)
OCR1_BCUCCEAN128	0x001A	: Barcode 1D UCC/EAN-128
OCR1_BCAUTO	0x001B	: Barcode 1D Auto Recognition (VAPI 4.2.3+ only)
OCR1_DATAMATRIX	0x001C	: Barcode 2D DataMatrix
OCR1_QRCODE	0x001D	: QR-Code
OCR1_PATCH	0x001E	: Patch Code Symbols
OCR1_BC2OF5NI	0x001F	: Barcode 1D Code 2 of 5 Non-interleaved (Datalogic, Iata1, Iata2, Industrial, Matrix)
OCR1_BC93	0x0020	: Barcode 1D Code 93

OCR1_DATABAR	0x0021	: Barcode GS1-Databar (RSS-14, Truncated, Omnidirectional, Stacked Omnidirectional, Expanded, Expanded Stacked and Limited)
OCR1_BCAUTO2D	0x0022	: Barcode 2D Auto Recognition (VAPI 4.2.3+ only)

The valid range for the Reject threshold is 10-100, lower value generates more rejects.

Valid in State : All
State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

BYTE * pBmp – pointer to the image buffer.

char * pDestString – destination string pointer.

DWORD StringLen – Length of the string pointed to by pDestString.

int Font – OCR Font.

int Threshold – Reject Threshold.

Return Value : FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example

```
char OCRCodeline[80];
char ApiErrorString[200];
BYTE *pSnippetBuffer;
ImagesStruct *SnippetStruct;
.....
case WMPAR_SORTER_SNIPPET_READY:
    // Recognizing snippet 0 bitmap with OCR font
    SnippetStruct = (ImagesStruct *) LPARAM;
    pSnippetBuffer = SnippetStruct->Images[0].pBuffer;
    // OCRCodeline string will contain the recognized
    codeline
    if( !GetOCRCodeline( m_DeviceID, pSnippetBuffer,
                        OCRCodeline, 80, OCR1_OCRA, 55 ) )
    {
        // Report error
        GetApiErrorString( ApiErrorString, 200 );
        MessageBox(...);
    }
    break;
```

SendPrinterData

BOOL *SendPrinterData* (DWORD DeviceID, DWORD dwHead, LOGFONT lf,
char *sText, DWORD dwTextOffset, char *sImagePath,
DWORD dwImgOffset, DWORD dwImgSrcType)

This function is used to define the printer text and bitmap data, their position on the document and the font used to create the text.

The bitmap image can be loaded from a file, or passed directly by its memory address (as a DIB, Device Independent Bitmap).

All Windows fonts can be selected and used for the printed text. Printer default font has been chosen to optimize uppercase characters usage. Lowercase letters could lose some detail at the very upper or very lower end, and in this case a font size reduction could be needed.

Valid in State: Smart Printer disabled

This function has to be called in **DeviceOnline** state, before the **StartFeeding** call, for the first document. For the next documents, in **DeviceFeeding** state, has to be called during WMPAR_SORTER_NEW_DOCUMENT. For 30 and 60 DPM machine this function can be called during WMPAR_SORTER_SET_ITEM_OUTPUT, instead of WMPAR_SORTER_NEW_DOCUMENT.

When the Smart Printer is disabled, the printer is an up-stream device. This means that the printer information have to be defined before the document feeding. Thus, the call before **StartFeeding** defines the printer data for the first document. The following calls, during WMPAR_SORTER_NEW_DOCUMENT message, define the printer data for the next document.

Example of printer sequence:

1. SendPrinterData(...) for the 1st doc
2. StartFeeding(...)
3. During NEW_DOC message of the 1st doc call SendPrinterData(...) for the 2nd doc
4. During NEW_DOC message of the nth doc call SendPrinterData(...) for the n+1th doc

Smart Enabled

This function has to be called during the WMPAR_SORTER_SET_ITEM_OUTPUT message for all the machines. When Smart printer is enabled, the printer is a downstream device. This means that the printer information can be defined after MICR and/or OCR information are available.

Example of printer sequence:

1. Call StartFeeding(...)
2. During SET_ITEM_OUTPUT message of the 1st doc call SendPrinterData(...) for the 1st doc
3. During SET_ITEM_OUTPUT message of the nth doc call

SendPrinterData(...) for the nth doc

State Transition : None

Arguments:

- | | | |
|---------------------------|---|--|
| DWORD DeviceID | - | The Identification number of the device . |
| DWORD dwHead | - | Printing head selector. Must be always 0. |
| LOGFONT lf | - | Font descriptor. Setting this structure to all zero means default font.
My Vision X AGP default is Arial, 16, normal.
My Vision X no AGP (single line) default is Arial, 12, normal. |
| char * sText | - | User-allocated string containing the text to be printer. Capital letters are suggested.
A NULL value means that no text will be printed. |
| DWORD dwTextOffset | - | Horizontal position of the printing text on the document referred to the leading or to the trailing edge. The position is expressed in mm. |
| char * sImgPath | - | User-allocated string containing the path to the Bitmap image or user-allocated buffer pointing the DIB (Device Independent Bitmap) in memory.
A NULL value means that no image will be printed. |
| DWORD dwImgOffset | - | Horizontal Position of the image on document referred to leading or the trailing edge. . The position is expressed in mm.
Since API version 2.11.1.2 this value can express a vertical position in pixels. This option is available <u>only for the AGP with 2 lines</u> enabled and <u>when there's no text to print</u> (sText is NULL or empty). The position is expressed in pixels. The values range is form 0 to 49. The LSBytes of the DWORD must contain the horizontal position in mm. The MSBytes must contain the vertical position in pixels. |
| DWORD dwImgSrcType | - | Define sImgPath parameter type. Values are:
PRT_SRC_FILE: sImgPath is a path to a file
PRT_SRC_MEM_PTR: sImgPath is a pointer to a memory buffer |

Return Value : FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example

```
char ApiErrorString[200];
.....
// Send printer data for the first document
```


Panini Vision API 4.5.0 Reference Manual Revision 1

```
if( !SendPrinterData( m_DeviceID, 0, m_LogFont, StringToPrint,
    m_TextTab, pBmpToPrint, m_BmpTab, PRT_SRC_FILE ) )
{
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox(...);
}

else
{
    // Now the Device is ready to feed the first Document
    // the ID number of the Device is the only parameter of the
    // StartFeeding function
    StartFeeding( m_DeviceID );
}

.....
// Send printer data for the next documents
case WMPAR_SORTER_SET_ITEM_OUTPUT :
    if( !SendPrinterData( m_DeviceID, 0, m_LogFont,
        StringToPrint, m_TextTab, pBmpToPrint, m_BmpTab,
        PRT_SRC_FILE ) )
    {
        // Report error
        GetApiErrorString( ApiErrorString, 200 );
        MessageBox(...);
    }
    .....
break;

// Example for vertical position (for AGP 2 lines, no text)
DWORD dwBmpHorizPos = 10; // in mm
DWORD dwBmpVertPos = 20; // in pixels

SendPrinterData( m_DeviceID, 0, m_LogFont, "", 0,
    pBmpToPrint, (dwBmpVertPos<<16)| dwBmpHorizPos, PRT_SRC_FILE );
```

SetVirtualEndorsement

BOOL *SetVirtualEndorsement*(DWORD DeviceID, DWORD AreaID, char *sText,
char *sBmpPath, DWORD BmpSrcType)

This function is used to define the printer text and the bitmap data of a certain Virtual endorsement area during the document processing.

The text is a string that contains one or more lines separated by a '\n' character.

The bitmap image can be loaded from a file.

Valid in State: Smart Printer disabled

This function has to be called in **DeviceOnline** state, before the **StartFeeding** call, for the first document. For the next documents, in **DeviceFeeding** state, has to be called during WMPAR_SORTER_NEW_DOCUMENT. For 30 and 60 DPM machine this function can be called during

WMPAR_SORTER_SET_ITEM_OUTPUT, instead of
WMPAR_SORTER_NEW_DOCUMENT.

When the Smart Printer is disabled, the printer is an up-stream device. This means that the printer information have to be defined before the document feeding. Thus, the call before **StartFeeding** defines the printer data for the first document. The following calls, during
WMPAR_SORTER_NEW_DOCUMENT message, define the printer data for the next document.

Example of printer sequence:

5. SetVirtualEndorsement(...) for the 1st doc
6. StartFeeding(...)
7. During NEW_DOC message of the 1st doc call
SetVirtualEndorsement (...) for the 2nd doc
8. During NEW_DOC message of the nth doc call
SetVirtualEndorsement (...) for the n+1th doc

Smart Enabled

This function has to be called during the
WMPAR_SORTER_SET_ITEM_OUTPUT message for all the machines. When Smart printer is enabled, the printer is a downstream device. This means that the printer information can be defined after MICR and/or OCR information are available.

Example of printer sequence:

4. Call StartFeeding(...)
5. During SET_ITEM_OUTPUT message of the 1st doc call
SetVirtualEndorsement (...) for the 1st doc
6. During SET_ITEM_OUTPUT message of the nth doc call
SetVirtualEndorsement (...) for the nth doc

Valid in State : **DeviceOnline**, **DeviceFeeding** (SetItemOutput event)

State Transition : None

Arguments:

- | | | |
|------------------------|---|--|
| DWORD DeviceID | - | The Identification number of the device |
| DWORD AreaID | - | Sets the Area ID. Its values are defined in VApiInterface.h. |
| char * sText | - | The text to be printed on the image. It can contain multiple lines of text separated by a '\n' character. |
| char * sBmpPath | - | The path to a bitmap file. It can be a 1, 8 or 24 bpp image. The bitmap are automatically converted in the color space of the image (i.e. color converted in greyscale, color in black and white and greyscale in black and white). If the bitmap header contains the resolution of the image, it is automatically rescaled to the document image resolution. If the bitmap header doesn't define any resolution it is copied pixelwise on the document image. |

DWORD BmpSrcType - Sets the bitmap type. Only file path are admitted. Its value is defined in VApiInterface.h.

Return Value: FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example:

```
char ApiErrorString[200];
...
// Set the data for the first document. The application is able to
// call the function multiple times per document.
if( !SetVirtualEndorsement(m_DeviceID, ENDORSEMENT_AREA_BOFD, "\n\nBOFD
                                area", NULL, 0 ) )
{
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox(...);
}
else
{
    // Now the Device is ready to feed the first Document
    // the ID number of the Device is the only parameter of the
    // StartFeeding function
    StartFeeding( m_DeviceID );
}
...
// Send printer data for the next documents
case WMPAR_SORTER_SET_ITEM_OUTPUT :
    if( !SetVirtualEndorsement(m_DeviceID, ENDORSEMENT_AREA_BOFD, "\n\nBOFD
                                area", NULL, 0 ) )
    {
        // Report error
        GetApiErrorString( ApiErrorString, 200 );
        MessageBox(...);
    }
    ...
    break;
```

FreeImagesBuffer

BOOL FreeImagesBuffer(ImagesStruct *pImageStruct)

This function Frees the memory used for the Image buffers, the application must call this method when the buffers are no longer needed.

Valid in State : All

State Transition : None

Arguments : ImagesStruct * *pImageStruct* – the Image Structure to be released.

Return Value : FALSE if an error occurs, call ***GetApiError*** or ***GetApiErrorString*** to get more information about it.

Example

```
ImagesStruct * pImages;  
.....  
case WMPAR_IMAGES_READY:  
    pImages = ( ImagesStruct * ) LPARAM;  
    .....  
    FreeImagesBuffer(pImages);  
    pImages = NULL;  
    break;
```

FreeSnippetBuffer

BOOL ***FreeSnippetBuffer***(ImagesStruct *pImageStruct)

This function Frees the memory used for the Snippet buffers, the application must call this method when the buffers are no longer needed.

Valid in State : All

State Transition : None

Arguments : ImagesStruct * ***pImageStruct*** – the Snippet Structure to be released.

Return Value : FALSE if an error occurs, call ***GetApiError*** or ***GetApiErrorString*** to get more information about it.

Example

```
ImagesStruct * pSnippets;  
.....  
case WMPAR_IMAGES_READY:  
    pSnippets = ( ImagesStruct * ) LPARAM;  
    .....  
    FreeImagesBuffer(pSnippets);  
    pSnippets = NULL;  
    break;
```

DisableDownstreamImage

BOOL ***DisableDownStreamImage***(DWORD DeviceID, DWORD DocID)

This function disables on the fly the Rear image capture.

If the Rear image is not enabled the application hasn't to call it. If the Rear image requested the app can decide whether to call it or not.

If not called and the Rear is enabled, the image is captured and compressed and the application will receive a valid pointer in the ImagesStruct.

If called, the Rear capture is cancelled and the application will receive a NULL pointer in the ImagesStruct. The function must be called before the SetPocket.

It is available only for VisionS device.

Valid in State : ***DeviceFeeding***

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

DWORD DocID – the document ID.

Return Value : FALSE if an error occurs, call ***GetApiError*** or ***GetApiErrorString*** to get more information about it.

Example

```
DWORD DocID;
...
case WMPAR_SORTER_SET_ITEM_OUTPUT:
    DocID = (DWORD) LPARAM;
    ...
    // DisableDownStreamImage must be called before SetPocket
    if( DevParams.ImagePropertiesRear1.Format != FORMAT_NONE ||
        DevParams.ImagePropertiesRear2.Format != FORMAT_NONE )
    {
        // For some reason the application can decide to disable
        // the Rear image capture
        if( ... )
        {
            if( DisableDownStreamImage( m_DeviceID, DocID ) )
            {
                // Report error
                GetApiErrorString( ApiErrorString, 200 );
                MessageBox(...);
            }
        }
    }

    if( !SetPocket( m_DeviceID, DocNumber, 1 ) )
    {
        // Report error
        GetApiErrorString( ApiErrorString, 200 );
        MessageBox(...);
    }
    ...
    break;
```

GetDoubleFeedingResult

BOOL *GetDoubleFeedingResult*(DWORD DeviceID, DWORD dwDocID,
 BOOL *pbDetected)

This function returns the result of the double-feed detection. When a multiple feed has been detected pbDetected is set to TRUE. If the Double-feeding detection is disabled the function always return FALSE in the pbDetected parameter.

It is available only for VisionS device. VisionX doesn't support it.

Valid in State : *DeviceFeeding*

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

 DWORD **DocID** – the document ID.

 BOOL ***pbDetected** – receive the double-feeding result. TRUE means detected.

Return Value : FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example

```
DWORD DocID;  
...  
case WMPAR_SORTER_SET_ITEM_OUTPUT:  
    DocID = (DWORD) LPARAM;  
    ...  
    BOOL bDetected = FALSE;  
    GetDoubleFeedingResult( m_dwDeviceID, ID, &bDetected );  
    if( bDetected )  
    {  
        // Double-Feeding detected  
        StopFeeding( m_dwDeviceID );  
    }  
  
    if( !SetPocket( m_DeviceID, DocNumber, 1 ) )  
    {  
        // Report error  
        GetApiErrorString( ApiErrorString, 200 );  
        MessageBox(...);  
    }  
    ...  
    break;
```

GetFullPocketStatus

BOOL *GetFullPocketStatus*(DWORD DeviceID, DWORD *pStatus, BOOL bWarning)

This function returns the status of the Full Pockets detection.

This function is supported by the VisionS only.

There are two kind of full pocket messages:

1. Full pocket warning

It means that a pocket has become almost full. The VisionS device doesn't stop the process. Calling the *GetFullPocketStatus* the application get the actual status and it can exclude the almost full pocket from the sorting, allowing the operator to remove from the pocket. All this operations can be executed without stopping the process.

If after this warning the app continue to put documents in the "almost full" pockets the stop the job after N documents signalling a stop for Full Pocket condition.

N is a firmware parameter.

2. Full pocket

It means that a pocket is completely full. The VisionS device automatically stops the process completing the pending documents and the device state returns to OnLine.

The parameter pStatus must be an array of 2 DWORD. The LSb of the 1st DWORD is the pocket #1. MSb is the pocket #32. The LSb of the 2nd DWORD is the pocket #33. The MSb is the pocket #64. Each bit represent the Full Pocket status. A bit set to 1 means Full Pocket detected.

Valid in State : *DeviceOnline*, *DeviceFeeding*

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

DWORD ***pStatus** – It's a pointer to the destination buffer. It must be an array of two DWORDs.

BOOL **bWarning** – If it is set to TRUE the function returns the Full pocket warning status. If it is set to FALSE the function return the completely Full pocket status.

Return Value : FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example

```
DWORD DocID;
...
case WMPAR_SORTER_EXCEPTION:
    DWORD ErrorCode = (DWORD) (LParam & 0x0000FFFF);
    DocID = ((DWORD) LParam & 0xFFFF0000) >> 16;
    ...
    if( ErrorCode == DEVICE_ERR_FULL_POCKET )
```

```
{
    DWORD FullPktsStatus[2];
    // Get the Full Pockets warning status
    if( GetFullPocketStatus( DeviceID, FullPktsStatus, TRUE ) )
    {
        // Application has to manage the full pocket warning status
        ...
    }
    else
    {
        // Report error
        GetApiErrorString( ApiErrorString, 200 );
        MessageBox(...);
    }
}

if( !SetPocket( DeviceID, DocID, 1 ) )
{
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox(...);
}
...
break;
```

GetFullPocketStatus (VN only)

BOOL *GetFullPocketStatus*(DWORD DeviceID, DWORD *pStatus)

This function returns the status of the Full Pocket(s) sensor.
This function is supported for the Vision NeXt only.

The parameter pStatus must be a pointer to a DWORD.

If successful (i.e. the function returns TRUE), the value of pStatus DWORD will contain a value which gives the status of the two pockets:

- 0 : Pockets NOT full
- 1 : Pocket 1 full (destination pocket)
- 2 : Pocket 2 full (exception pocket)

Following defines can be used to easily query the return value:

```
#define STATUS_POCKET_NO_FULL      0
#define STATUS_POCKET_DEST_FULL   1
#define STATUS_POCKET_EXCP_FULL   2
```

Valid in State : ***DeviceOnline, DeviceFeeding***
State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

 DWORD ***pStatus** – it's a pointer to the result DWORD.

Return Value : TRUE if the function terminates correctly.
FALSE if an error occurs, call ***GetApiError*** or ***GetApiErrorString***
to get more information about it.

Example (refer also to the Barebones application source code – VNBarebonesDlg.cpp)

```
DWORD DocID;
...
case WMPAR_SORTER_EXCEPTION:
    DWORD ErrorCode = (DWORD)(LParam & 0x0000FFFF);
    DocID = ((DWORD)LParam & 0xFFFF0000)>>16;
    ...
    if( ErrorCode == DEVICE_ERR_FULL_POCKET )
    {
        DWORD pocketStatus = 0;
        if ( GetFullPocketStatus(m_DeviceID, &pocketStatus) )
        {
            if ( pocketStatus == STATUS_POCKET_EXCP_FULL )
            {
                // Application has to manage the exception full pocket status
            }
            ...
        }
        else
        {
            // Report error
            GetApiErrorString( ApiErrorString, 200 );
            MessageBox(...);
        }
    }
    ...
    break;
```

GetDocumentType

BOOL ***GetDocumentType***(DWORD DeviceID, DWORD DocID, DWORD* DocType)

This function has to be called during one of the following messages:

- NEW_DOC
- MICR_AVAILABLE
- SET_ITEM_OUTPUT
- IMAGES_READY

It returns, in the pointer DocType, a value that represents if the item is a check or a plastic card (rigid document) or a page (A4/Letter/Legal).

The available types are:

- DOC_TYPE_CHEQUE
- DOC_TYPE_CARD
- DOC_TYPE_PAGE

Valid in State : ***DeviceFeeding***

State Transition : None

Arguments : DWORD DeviceID – the Identification number of the device.

DWORD DocID – the Identification of the Document

DWORD * DocType – type of processed document (available values are defined in the header file).

Return Value: FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example:

```
DWORD DocID;
.....
case WMPAR_SORTER_NEW_DOCUMENT:
    DocID = (DWORD) LPARAM;
    if(!GetDocumentType(m_DeviceID, DocID, &Doctype))
    {
        // Report error
        GetApiErrorString( ApiErrorString, 200 );
        MessageBox(...);
    }
    break;
```

EnableFranking

BOOL ***EnableFranking***(DWORD DeviceID, DWORD TrailingEdgeAlignment,
WORD StartPosition)

This function is used to enable the franking on the check, it needs to be called during the SET_DOWNSTREAM_OPTIONS message.

Valid in State : Feeding.

State Transition : None

Arguments: DWORD DeviceID - The Identification number of the device.

DWORD TrailingEdgeAlignment – It defines the alignment of the printed text (LEADING or TRAILING).

DWORD StartPosition – It's the starting position in mm from the alignment edge of the document.

Return Value: FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example

```
char ApiErrorString[200];
.....
case WMPAR_SORTER_SET_ITEM_OUTPUT :
    if( !EnableFranking( m_DeviceID, TRUE, 30))
    {
        // Report error
        GetApiErrorString( ApiErrorString, 200 );
        MessageBox(...);
    }
    .....
break;
```

IsTrackFree

BOOL *IsTrackFree*(DWORD DeviceID, BOOL *pFlag)

This function returns the status of the track. If all the sensors are free, it means that the track is free and a new document/card can be processed. If pFlag is set to TRUE the track is free. Otherwise one or more sensors are busy due to a document/card stopped in the track

Valid in State : *DeviceOnline*

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

BOOL * **pFlag** – It's the destination flag variable.

Return Value : FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example

```
BOOL Free = FALSE;
if( !IsTrackFree( m_DeviceID, &Free ) )
{
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox(...);
}
else
{
    if( Free )
```

```
{
    StartFeeding(...);
}
else
{
    // Message to the operator
    MessageBox("Please remove the document in the track" );
}
}
```

GetIQAResults

BOOL *GetIQAResults*(DWORD DeviceID, DWORD DocID, UINT ImageNumber,
DWORD Test, int *Value)

This function returns the result of a specific IQA test.
It can be called during the following messages:

- WMPAR_IMAGES_READY;
- WMPAR_SET_DOWNSTREAM_OPTION.

The result's value can be one of the following:

- 0 : the condition has not been tested;
- 1 : the condition is tested, and the defect is present;
- 2 : the condition is tested, and the defect is not present.

Valid in State : *DeviceFeeding*

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

 DWORD **DocID** – The document ID.

 UINT **ImageNumber** – Image's selector.

 DWORD **Test** – Test's selector.

 int ***Value** – Result.

Return Value : FALSE if an error occurs, call *GetIQALastError* or *GetIQALastErrorString*
to get more information about it.

Example: Refer to VxBarebones source code.

GetIQAValues

BOOL *GetIQAValues*(DWORD DeviceID, DWORD DocID, UINT ImageNumber,
 DWORD Parameter, int *Value)

This function returns the value of a specific parameter after the IQA test.
It can be called during the following messages:

- WMPAR_IMAGES_READY;
- WMPAR_SET_DOWNSTREAM_OPTION.

The value is a number expressed using the same unit of the corresponding parameter.
The parameter selector are defined in the VApiQAInterface.h file.

Valid in State : *DeviceFeeding*

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

DWORD DocID – The document ID.

UINT ImageNumber – Image's selector.

DWORD Parameter – Parameter's selector.

int *Value – Result's value.

Return Value : FALSE if an error occurs, call *GetIQALastError* or *GetIQALastErrorString*
 to get more information about it.

Example: Refer to VxBarebones source code.

ProcessPage

BOOL *ProcessPage*(DWORD DeviceID, BYTE *pBmp, char *PDFFileName)

This function uses the ABBYY FineReader engine to run OCR analysis of page and creates a
PDF file with the result.

It can be called during the following messages:

- WMPAR_SNIPPETS_READY;

Valid in State : *DeviceFeeding*

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

BYTE *pBmp – The pointer to the snippet on which to run OCR.

char ***PDFFileName** – The filename of the file used to store the created PDF.

Return Value : FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example:

```
ImagesStruct *pSnippets = NULL;
...
switch( WParam )
{
    ...
    case WMPAR_SNIPPETS_READY:
        pSnippets = (ImagesStruct *)lParam;
        if (!ProcessPage( m_DeviceID, pSnippets->
>Images[0].pBuffer, "Output.pdf" ))
        {
            // Error recovering ...
        }
        ...
        break;
}
```

Serial Functions

The Vision|X device mounts a RS232 port.

The following functions manage the communication through this port with other “external” devices.

The application can set the baud rate from 300 to 57600 using *Rs232SetBaud*.

The firmware maintains a TX and a RX buffer both of 255 bytes.

When the Application sends data through the MVX serial port using *Rs232Write*, the data are stored in the TX buffer and are immediately transmitted at the right baud rate.

When an “external” device sends data to the MVX, the firmware temporarily stores them in the RX buffer. The application detects the received data using *Rs232GetLen* and transfers them in a local buffer using *Rs232Read*.

These functions are temporary disabled when a serial Panini dongle is connected with the device.

Rs232SetBaud

BOOL *Rs232SetBaud*(DWORD DeviceID, UINT BaudRate)

Set the serial port BaudRate. Valid values are from 300 to 57600.

Valid in State : *DeviceChangeParameters*

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

 DWORD **BaudRate** – the BaudRate value.

Return Value : FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example

```
// Set 9600 baud
Rs232SetBaud( m_dwDeviceID, 9600 );
```

Rs232Write

BOOL *Rs232Write*(DWORD DeviceID, BYTE *pBuffer, BYTE Len)

This function sends pBuffer data through the serial port.

Valid in State : *DeviceOnLine*, *DeviceFeeding*

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

BYTE ***pBuffer** – Buffer of the data to be sent.

BYTE **Len** – The length of the buffer. The range is from 1 to 255.

Return Value : FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example

```
// Data buffer
BYTE Buffer[] = {0x02, 0x30, 0x31, 0x32, 0x03};
// Send buffer through the serial port
Rs232Write( m_dwDeviceID, Buffer, sizeof(Buffer) );
```

Rs232GetLen

BOOL *Rs232GetLen*(DWORD DeviceID, BYTE *pLen)

This function reads the length of the data contained in the device reception buffer.
This length should be used to call Rs232Read.

Valid in State : *DeviceOnLine*, *DeviceFeeding*

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

BYTE ***pLen** – the destination buffer for the returned length. The returned values are from 0 to 255.

Return Value : FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example

```
// Data buffer
BYTE Buffer[] = {0x02, 0x30, 0x31, 0x32, 0x03};
// Send buffer through the serial port
Rs232Write( m_dwDeviceID, Buffer, sizeof(Buffer) );
```


Rs232Read

BOOL *Rs232Read*(DWORD DeviceID, BYTE *pBuffer, BYTE Len)

This function transfers the data contained in the device reception buffer. The length of the data to be transferred is indicated by a previous call to Rs232GetLen.

Valid in State : *DeviceOnLine*, *DeviceFeeding*

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

 BYTE ***pBuffer** – the destination buffer for the returned data.

 BYTE **Len** - The length of the data to be transferred in pBuffer. Valid values are from 1 to 255 and must be less than or equal to the value returned by Rs232GetLen.

Return Value : FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example

```
// Data buffer
BYTE Buffer[255];
BYTE Len;

// Receive buffer through the serial port
Rs232GetLen( m_dwDeviceID, &Len );
if( Len > 0 )
    Rs232Read( m_dwDeviceID, Buffer, Len );
```

Maintenance Functions

ReadPrinterDropsCounter

BOOL *ReadPrinterDropsCounter*(DWORD DeviceID, DWORD *pdwDropsCounter)

The device is able to count the number of ink drops fired by the printer cartridge enabling the user application to monitor the cartridge consumption status. This function reads the printer cartridge's drops counter. It's available for every kind of printer device (Single line and AGP multi-line).

Valid in State : *DeviceOnLine*

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

DWORD *pdwDropsCounter – the destination buffer for the returned data.

Return Value : FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

ResetPrinterDropsCounter

BOOL ReadPrinterDropsCounter(DWORD DeviceID)

This function resets the drops counter for the printer cartridge. This function should be used to reset the drops counter after the ink-jet cartridge replacement. The user application is responsible for deciding when the ink cartridge needs to be replaced, refer to the relative technical bulletin for the respective ink quantity in each type of ink cartridge.

Valid in State : *DeviceOnLine*

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.

Return Value : FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example

```
// Data buffer
DWORD DropsCnt;

// Read the actual drops counter and control if the
// cartridge needs to be replaced
ReadPrinterDropsCounter( m_dwDeviceID, &DropsCnt );
if( DropsCnt > MAX_DROPS_COUNTER )
{
    MessageBox( "Replace the cartridge" );
    ResetPrinterDropsCounter( m_dwDeviceID );
}
```

GetPrinterCartridgeInfo

**BOOL GetPrinterCartridgeInfo(DWORD DeviceID, BOOL *bInserted,
DWORD *pCartridgeID)**

This function has the capability to detect the presence of the ink-jet cartridge in the device nest (Single line and AGP both) and can return the cartridge ID number for the AGP version.

This ID can be used to detect when a cartridge is replaced. The presence flag can be used, when the printer is enabled, to check if the printer cartridge is correctly mounted before starting a printing job.

It can be called before starting a printing job to verify if the cartridge is correctly mounted. This feature is not available for the previous version of the MyVisionX. For the previous version of the device the cartridge is considered always inserted and the ID is always 0.

Valid in State : ***DeviceOnLine***

State Transition : None

Arguments : **DWORD DeviceID** - the ID number of the device returned by the StartUp
BOOL *pInserted - receive the presence flag. If TRUE means cartridge inserted. If FALSE means cartridge not mounted.
DWORD *pCartridgeID - receive the AGP cartridge ID. For a single line printer this value is always 0. For an AGP printer this value is not 0 only when the presence flag is TRUE.

Example

```
// Data buffer
BOOL Inserted = FALSE;
DWORD CartridgeID = 0;

// get the cartridge info when printer is enabled
GetPrinterCartridgeInfo( DeviceID, &Inserted, &CartridgeID );
if( Inserted == FALSE )
{
    // The cartridge is not present
    // EXAMPLE: Ask to user to insert the cartridge
}
```

GetHistoricalCounter

BOOL *GetHistoricalCounter*(DWORD DeviceID, DWORD CounterID, DWORD *pValue)

This function returns the values of some historical counters.

The available counters are:

- **COUNTER_DOCS**
Return how many documents have been processed during the product life.
- **COUNTER_CARDS**
Return how many plastic cards have been processed during the product life.
- **COUNTER_FRANKING**
Return how many documents have been franked. This can be used in order to detect when the franking roller has to be replaced.

Valid in State : ***DeviceOnline***

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.
DWORD CounterID – It selects the counter.
DWORD *pValue – It's the destination counter variable.

Return Value : FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example

```
DWORD DocsCounter = 0;
if( !GetHistoricalCounter( DeviceID, COUNTER_DOCS, &DocsCounter ) )
{
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox (...);
}
```

ResetHistoricalCounter

BOOL ResetHistoricalCounter(DWORD DeviceID, DWORD CounterID)

This function reset some specific historical counters of consumable parts.

The available counters are:

- **COUNTER_FRANKING**

Return how many documents have been franked. This can be used in order to reset the counter when the franking roller has to be replaced.

Valid in State : *DeviceOnline*

State Transition : None

Arguments : **DWORD DeviceID** – the Identification number of the device.
DWORD CounterID – It selects the counter.

Return Value : FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Example

```
DWORD DocsCounter = 0;
if( !ResetHistoricalCounter( DeviceID, COUNTER_FRANKING ) )
{
    // Report error
    GetApiErrorString( ApiErrorString, 200 );
    MessageBox (...);
}
```

ServicePrinterHead (VN only)

BOOL *ServicePrinterHead*(DWORD DeviceID, DWORD ServiceID)

This function allows to perform a cartridge maintenance procedure. It is synchronous, so the calling thread is blocked until the maintenance of the cartridge is completed. If the operation requested doesn't complete successfully, the function returns FALSE and a proper exception is raised.

This function is supported for the Vision NeXt only.

Valid in State : *DeviceOnline*, *ChangeParameters*

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

 DWORD **ServiceID** – It selects the requested service type:

- CLEANING = 1
- CAPPING = 2
- PURGING = 3

Return Value : FALSE if an error occurs, call *GetDeviceError* to get more information about it.

NOTE: The function GetDeviceStatus() allows to get current status of "readiness" of the device and can be called to verify if the device is performing a maintenance procedure in multi-threaded applications.

Magnetic card reader

The Vision|X device makes available a Magnetic card reader. This device is compliant with the standard ISO 7810, 7811 and 7813.

This feature introduces a new message for the application (defined in VApiInterface.h): **WMPAR_MAGCARD**.

The message is sent to the application when a card is passed in front of the reading head. The device state has to be set to **OnLine**. In any other state the magcard reader is disabled.

The LPARAM field of the message contains an error code. Here is a list of them:

Magcard error code	Description
MAGCARD_ERROR_NONE (0)	Magcard decoding successful. The application can call GetMagCardResult in order to obtain the raw bit stream or GetMagCardResultString to get the decoded result (ASCII string).
MAGCARD_ERROR_PARITY (1)	A parity error has been detected. Ask for a new read. No data (raw or decoded) returned.
MAGCARD_ERROR_STANDARD (2)	The standard of the magcard is not recognized / supported. The application can call GetMagCardResult in order to obtain the raw bit-stream of the card. The API is not able to decode the card, but the application has the raw data and could proceed to decode the card by itself.
MAGCARD_ERROR_DIRECTION (3)	The card has been passed in the wrong direction. No data returned.

GetMagCardResult, GetMagCardResultString

BOOL *GetMagCardResult*(DWORD DeviceID, PCHAR sResult, DWORD BufferLen)

BOOL *GetMagCardResultString*(DWORD DeviceID, PCHAR sResult, DWORD BufferLen)

These functions have to be called when the application receives the message WMPAR_MAGCARD. The destination buffer has to be large enough to contain the result.

If the buffer length is not enough the function fails and doesn't return any data. We suggest to allocate a buffer of MAGCARD_MIN_BUFFER_LEN bytes.

The GetMagCardResult() will return the raw bit stream – inverted if card swapped in opposite direction (i.e. forward swipe) –in case of AAMVA cards, but decoded string in case of CREDIT/DEBIT cards (unless there's a MAGCARD_ERROR_PARITY error in which case the returned bit stream is EMPTY).

The GetMagCardResultString() will return the string containing decoded data read using both the ISO 4909 (financial) and the AAMVA (US driver licenses) standards. If both standards

cannot decode the acquired bit stream, a `MAGCARD_ERROR_STANDARD` error is returned in the `WMPAR_MAGCARD` event).

This function is not available for the previous version of the MyVisionX.

Valid in State : ***DeviceOnLine***

State Transition : None

Arguments: **DWORD DeviceID** - the ID number of the device returned by the StartUp
 PCHAR sResult - receive the magcard result (this buffer has to be allocated by the application)
 DWORD BufferLen – it's the sResult buffer length. We suggest allocating at least `MAGCARD_MIN_BUFFER_LEN` bytes.

Example

```
case WMPAR_MAGCARD:
{
    DWORD ErrCode = (DWORD)lParam;
    char sMagCardResult[MAGCARD_MIN_BUFFER_LEN];
    memset( sMagCardResult, 0, sizeof(sMagCardResult) );

    switch( ErrCode )
    {
        case MAGCARD_ERROR_NONE:
            if( GetMagCardResultString( m_DeviceID, sMagCardResult,
                                        sizeof(sMagCardResult) ) )
            {
                cstrDisplay.Format( "%s", sMagCardResult );
                WriteListBoxReport(cstrDisplay);
            }
            else
            {
                // API ERROR
            }

            MessageBox( sMagCardResult, "MagCard reader result",
                        MB_OK|MB_ICONINFORMATION|MB_TOPMOST );

            break;

        case MAGCARD_ERROR_PARITY:
            MessageBox( "Data corrupted!\n\nPlease, pass again the card.",
                        "MagCard reader result",
                        MB_OK|MB_ICONWARNING|MB_TOPMOST );

            break;

        case MAGCARD_ERROR_STANDARD:
            if( GetMagCardResult( m_DeviceID, sMagCardResult,
                                sizeof(sMagCardResult) ) )
            {
                sprintf(FileToSave,"%s\\MagCardBitStream.bin", m_DeviceDir);
                f = fopen( FileToSave, "wb+" );
                if( f )
                {

```

Panini Vision API 4.5.0

Reference Manual Revision 1

```
        fwrite( sMagCardResult, 1, sizeof(sMagCardResult), f );
        fclose(f);
    }
}
else
{
    // API ERROR
}

MessageBox( "Card not supported!\n\nBitstream saved on the disk.",
            "MagCard reader result", MB_OK|MB_ICONSTOP|MB_TOPMOST );
break;

case MAGCARD_ERROR_DIRECTION:
    MessageBox( "Wrong reading direction!\n\nPlease, pass again the
card in the opposite direction.",
            "MagCard reader result", MB_OK|MB_ICONSTOP|MB_TOPMOST );
    break;
}}
break;
```


Appendix A: Devices comparison

Interface functions

Each device engine implements a subset of the entire set of exported functions of the VisionAPI.

All the interface functions are defined in the C header file *VApiInterface.h*.

The following table compares the functions list implemented by the specific engine.

Function	VX	VN	ID	WI	VxA4	VS
GetApiRelease	X	X	X	X	X	X
GetDriverRelease	X	X	X	X	X	X
GetFwVersion	X	X	X	X	X	X
GetApiError	X	X	X	X	X	X
GetApiErrorString	X	X	X	X	X	X
GetUsbError	X	X	X	X	X	X
GetUsbErrorString	X	X	X	X	X	X
GetDeviceError	X	X	X	X	X	X
GetDeviceErrorString	X	X	X	X	X	X
GetSorterError	X	X	X	X	X	X
GetSorterErrorString	X	X	X	X	X	X
GetSerialNumber	X	X	X	X	X	X
ReadCryptedIDCard	X	X	X	X	X	X
UpgradeIDCard	X	X	X	X	X	X
GetIDCardDescription	X	X	X	X	X	X
SetMaxDPM	X	X				X
GetMaxDPM	X	X				X
GetAvailablePockets	X	X				X
SetMaxPocket	X	X				
SetHandDropDly	X	X	X	X	X	X
SetAGPLines	X	X				X
SetFeederLimit	X	X				
SetSorterParameter	X	X				X
GetSorterParameter	X	X				X
SetMicrOcrFontOverride	X	X	X	X	X	X
SetDeviceParameters	X	X	X	X	X	X
GetDeviceParameters	X	X	X	X	X	
SetDocProcessingParam		X		X		
GetDeviceState	X	X	X	X	X	X
GetDeviceStateString	X	X	X	X	X	X
GetDeviceStatus		X				
StartUp	X	X	X	X	X	X
ShutDown	X	X	X	X	X	X
OnLine	X	X	X	X	X	X
ChangeParameters	X	X	X	X	X	X
IsFeederEmpty	X	X	X	X	X	
StartFeeding	X	X	X	X	X	X
StopFeeding	X	X	X	X	X	X
FreeTrack	X	X				X
SetPocket	X	X	X	X	X	X

Panini Vision API 4.5.0
Reference Manual Revision 1

GetDocumentLength	X	X	X	X	X	X
GetMicrCodeline	X	X	X	X		X
GetOCRCodeline	X	X	X	X	X	X
SendPrinterData	X	X		X		X
FreeImagesBuffer	X	X	X	X	X	X
FreeSnippetBuffer	X	X	X	X	X	
SetImageAdjustment	X	X	X	X	X	X
Rs232SetBaud	X					
Rs232Write	X					
Rs232GetLen	X					
Rs232Read	X					
ReadPrinterDropsCounter	X	X		X		X
ResetPrinterDropsCounter	X	X		X		X
GetPrinterCartridgeInfo	X			X		
GetMagcardResult	X					
GetDeviceFeature	X	X	X	X	X	X
DisableDownstreamImage						X
GetDoubleFeedingResult						X
GetFullPocketStatus						X
SetFeedingMode						X
GetVirtualEndorsementAreaProperty	X	X	X	X	X	
SetVirtualEndorsementAreaProperty	X	X	X	X	X	
SetVirtualEndorsement	X	X	X	X	X	
SetIQAParameter	X	X		X	X	X
GetIQAResults	X	X		X	X	X
GetIQAValues	X	X		X	X	X
GetIQALastError	X	X		X	X	X
GetIQALastErrorString	X	X		X	X	X
GetDocumentType		X	X	X	X	
EnableFranking	X		X	X		
IsTrackFree		X	X	X	X	
GetHistoricalCounter			X	X	X	
ResetHistoricalCounter			X	X	X	
ProcessPage	X		X	X	X	
ServicePrinterHead		X				

Implemented messages list

Message	VX	VN	ID	WI	VxA4	VS
WMPAR_SORTER_CONNECTED	X	X	X	X	X	X
WMPAR_SORTER_DISCONNECTED	X	X	X	X	X	X
WMPAR_SORTER_NEW_DOCUMENT	X	X	X	X	X	X
WMPAR_SORTER_MICR_AVAILABLE	X	X	X	X	X	
WMPAR_SORTER_SET_ITEM_OUTPUT	X	X	X	X	X	X
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET	X	X	X	X	X	X
WMPAR_SORTER_EXCEPTION	X	X	X	X	X	X
WMPAR_SORTER_IMAGES_READY	X	X	X	X	X	
WMPAR_SORTER_SNIPPETS_READY	X	X	X	X	X	X
WMPAR_SORTER_DOC_COMPLETED	X	X	X	X	X	X
WMPAR_SORTER_STANDBY	X					
WMPAR_SORTER_MAGCARD	X					
WMPAR_SET_DOWNSTREAM_OPTIONS	X		X	X	X	
WMPAR_COMPRESSED_IMG_FOR_DECISION			X	X	X	

Device parameters

MICR Font	VX	VN	ID	WI	VxA4	VS
MICR_FONT_CMC7	X	X	X	X	X	X
MICR_FONT_E13B	X	X	X	X	X	X
MICR_FONT_AUTO	X	X	X	X	X	X
MICR_FONT_E13B_OCR	X	X	X	X	X	X
MICR_FONT_CMC7_OCR	X	X	X	X	X	X
MICR_FONT_AUTO_OCR	X	X	X	X	X	

Image DPI	VX	VN	ID	WI	VxA4	VS
100	X	X	X	X	X	X
200	X	X	X	X	X	X
300	X	X	X	X	X	
600		X		X	X	

Printer quality	VX	VN	ID	WI	VxA4	VS
PRINTER_AGP_QUALITY_NORMAL	X	X				X
PRINTER_AGP_QUALITY_HIGH	X	X				X
PRINTER_AGP_QUALITY_DRAFT	X	X				X

Printer alignment	VX	VN	ID	WI	VxA4	VS
PRINTER_ENABLE_LEADING_EDGE	X	X		X		X
PRINTER_ENABLE_TRAILING_EDGE	X	X		X		X

Snippet Color	VX	VN	ID	WI	VxA4	VS
SNIPPET_COLOR_GREY_LEVEL	X	X	X	X	X	X
SNIPPET_COLOR_BLACK_WHITE	X	X	X	X	X	X
SNIPPET_COLOR_COLOR	X	X		X	X	X

Panini Vision API 4.5.0
Reference Manual Revision 1

OCR font	VX	VN	ID	WI	VxA4	VS
OCR1_OCRA	X	X	X	X	X	X
OCR1_OCRB	X	X	X	X	X	X
OCR1_OCRAB	X	X	X	X	X	X
OCR1_OCRBUK	X	X	X	X	X	X
OCR1_E13BO	X	X	X	X	X	X
OCR1_E13BOXOCRBUK	X	X	X	X	X	X
OCR1_OCRAALNUM	X	X	X	X	X	X
OCR1_OCRBALNUM	X	X	X	X	X	X
OCR1_OCRB1403	X	X	X	X	X	X
OCR1_OCRAN	X	X	X	X	X	X
OCR1_BC128	X	X	X	X	X	X
OCR1_BC39	X	X	X	X	X	X
OCR1_BC2OF5	X	X	X	X	X	X
OCR1_BCUPCA	X	X	X	X	X	X
OCR1_BCEAN13	X	X	X	X	X	X
OCR1_BCUPCE	X	X	X	X	X	X
OCR1_BCEAN8	X	X	X	X	X	X
OCR1_BCPDF417	X	X	X	X	X	X
OCR1_CMC7O	X	X	X	X	X	X
OCR1_BCCODABAR						X
OCR1_BCUCCEAN128						X
OCR1_BCAUTO						X

Feeding Mode	VX	VN	ID	WI	VxA4	VS
HOPPER_FEED	X	X				X
DROP_FEED	X	X	X	X	X	X

Appendix B: Vision S

This appendix is an integration of the VisionAPI reference manual. It explains how to use VisionAPI interface with a VisionS device.

The Panini VisionS software interface is *VisionAPI.dll*. The Device engine layer is *VSEngine.dll* file.

The entire set of file needed to install VisionAPI is composed of the following list of files:

- *VisionAPI.dll*;
- *VsEngine.dll*;
- *Baroc.dll* (OCR engine, optional);
- *AxBar32.dll* (Barcode engine, optional);
- *PaniniOCR.dll* (OCR engine for MICR+OCR, optional);
- *PaniniIQALibrary.dll* (IQA engine, optional)
- PaniniOCRParms folder (data for the PaniniOCR engine, optional);
- PaniniOCR.tbl and Panini.tbl (PaniniOCR configurations, optional);
- PaniniOCR depends on 3 Microsoft libraries: *mfc71.dll*, *msvcp71.dll* and *msvcr71.dll*;
- Microsoft USB Driver.

USB connection

The VisionS requires two USB 2.0 ports.

One port is for the Control main-board. The other is for the image control board.

The 1st port connection (main-board) is required in order to connect to the device with the PC. This means that if this port is not connected to the PC the application will never receive the message WMPAR_SORTER_CONNECTED after a **StartUp** call.

The 2nd port connection is required in order to be able to capture images. If the port is not connected the application receive the connected message, but an error is returned when an image is requested in the **DeviceParameters** structure.

The following table explains the connection cases:

Main-board USB cable	Image-board USB cable	WMPAR_SORTER_CONNECTED message toward the app	Image capture available
N.C.	N.C.	Not sent	No
C.	N.C.	Sent	No
N.C.	C.	Not sent	No
C.	C.	Sent	Yes

N.C. = Not connected

C. = Connected

USB driver

USB driver certification	VxEngine	VsEngine
WHQL	X	

Vision S Messages

As for MyVisionX/VisionX applications, the VisionS applications have create a message handler to receive the messages from the API.

VisionS uses a subset of the VisionAPI messages. These are the following:

- WMPAR_SORTER_CONNECTED;
- WMPAR_SORTER_DISCONNECTED;
- WMPAR_SORTER_NEW_DOCUMENT;
- WMPAR_SORTER_SET_ITEM_OUTPUT;
- WMPAR_SORTER_DOCUMENT_INSIDE_POCKET;
- WMPAR_SNIPPETS_READY;
- WMPAR_DOC_COMPLETED;
- WMPAR_SORTER_EXCEPTION.

It doesn't support the following messages:

- WMPAR_MICR_AVAILABLE;
- WMPAR_IMAGES_READY;
- WMPAR_STANDBY;
- WMPAR_MAGCARD.

The first two messages signal the connection and the disconnection of the VisionS with the PC as usual (refer to VisionAPI reference manual).

The **WMPAR_SORTER_NEW_DOCUMENT** message signals that a new item is entered in the track. The *LParam* contains the document ID.

During this event the application can call the function: ***GetDocumentLength***. It returns the length of document in mm. This value can be used as criteria of the sorting.

The **WMPAR_SORTER_SET_ITEM_OUTPUT** message signals that the MICR codeline is available (is MICR is enabled) and the machine need information about the downstream devices (printer, rear image and destination pocket). The *LParam* contains the document ID.

During this event the application can call the following functions:

- ***GetDoubleFeedingResult***
- ***GetMicrCodeline***
- ***SendPrinterData***
- ***EnableDownStreamImage***
- ***StopFeeding***
- ***SetPocket***

The function ***GetDoubleFeedingResult*** returns the result of the double-feed detection. When a multiple feed has been detected the function returns TRUE. If the Double-feeding detection is disabled the function always return FALSE in the pbDetected parameter. The VisionS never stops the process for a double-feed, it's always the application that decides to stop or continue the process.

The prototype of function is declared in *VApiInterface.h*. It is available only for VisionS device. VisionX doesn't support it.

The function ***GetMicrCodeline*** returns the MICR results (if MICR is enabled).

The function ***SendPrinterData*** set the printing data for the document. If the Printer is disabled the app must not call this function. If enabled the application can decide whether call or not the function using the document information (MICR, length, double feeding, OCR ...). If you don't call it the VisionS will not print anything. VisionS AGP printer is able to print up to 4 text lines and a black and white bitmap. The bitmap height must be 100 pixels.

The function ***DisableDownStreamImage*** disables the Rear image capture.

If the Rear image is not enabled the application hasn't to call it. If the Rear image requested the app can decide whether to call or not.

If not called and the Rear is enabled, the image is captured and compressed and the application will receive a valid pointer in the *ImagesStruct*.

If called, the Rear capture is cancelled and the application will receive a NULL pointer in the *ImagesStruct*.

The function ***SetPocket*** set the destination pocket of the document. The application must call it during the SetItemOutput event and after all the previous function. It is the last function to be called during the **SET_ITEM_OUTPUT** event.

The VisionS can work as synchronous or asynchronous machine.

The synchronous mode is called Start/stop and works like the Panini S1Vision. The device has the capability to wait for the pocket setting, even if the application takes a lot of time to decide the pocket. In case of delay the document is stopped in the View station and it will be restarted after the SetPocket call.

The asynchronous mode is called Flow-mode. The machine never stop the document in the view station and if the application takes too much time to decide the pocket, the device stops the job signalling document without destination or, if a default pocket is defined, sends the document to it ignoring eventual delayed SetPocket call.

To know how many pockets are available on the machine the application has to call the function ***GetAvailablePockets***.

To set the *Start/Stop* or the *Flow-Mode* working mode the application has to call the function ***SetFeedingMode***.

The **WMPAR_SORTER_DOCUMENT_INSIDE_POCKET** message signals that the document has gone in the destination pocket without problem. LParam contains the document ID. This message means "the paper transport process has been completed on this document ID".

The **WMPAR_SNIPPETS_READY** message signals that one or more snippets are ready. LParam contains the pointer to an ImagesStruct data. The application can define up to 10 snippets, Front or Rear. If you define N snippets you will receive N times this message.

The image data are transferred from the image board to the PC in packets. Packet by packet a partial image is processed. As soon as a snippet can be extracted from the partial image, it is sent to the application. This is done especially for Front snippet used for OCR recognition in order to sort the document based on the OCR result.

The Front snippets have two kind of enabling: one is the normal enable, the other is “Enable for Decision”. In the last case the snippet is always sent before the SetItemOutput event to let the application decide the pocket based on the OCR result.

Example 1 – Snippet for decision:

Snippet 0: Front, enabled for decision

The message sequence will be:

1. WMPAR_NEW_DOCUMENT
2. WMPAR_SNIPPETS_READY(Snippet0)
3. WMPAR_SET_ITEM_OUTPUT (where the app calls SetPocket)
4. ...

If a snippet is enabled for decision its message is always sent before SET_ITEM_OUPUT_MESSAGE.

Example 1 – Snippet not for decision:

Snippet 0: Front, enabled

The message sequence could be:

1. WMPAR_NEW_DOCUMENT
2. WMPAR_SNIPPETS_READY(Snippet0)
3. WMPAR_SET_ITEM_OUTPUT (where the app calls SetPocket)
4. ...

Or it could be:

1. WMPAR_NEW_DOCUMENT
2. WMPAR_SET_ITEM_OUTPUT (where the app calls SetPocket)
3. WMPAR_SNIPPETS_READY(Snippet0)
4. ...

If a snippet is not enabled for decision, the sequences order it's not assured. It depends on internal timings. It works as for a VisionX device.

The VisionS includes an OCR engine (used for VisionX too), but the developer can decide to use his own OCR using the snippets bitmap data.

The snippets for sorting are available only for the Front image side and when the VisionS device works as a Start/Stop machine.

The **WMPAR_DOC_COMPLETED** message signals that document processing is complete. If an image is requested LParam contains the pointer to an ImagesStruct data including up to 5 images (2 fronts, 2 rears and the multipage TIFF).

If there aren't enabled images LParam contains only the document ID.

Example:

```
ImagesStruct *pImages = NULL;
DWORD DocID = 0;
...
switch( WParam )
{
    ...
    case WMPAR_DOC_COMPLETED:
```



```
if( DevParams.ImagePropertiesFront1.Format != FORMAT_NONE ||  
    DevParams.ImagePropertiesFront2.Format != FORMAT_NONE ||  
    DevParams.ImagePropertiesRear1.Format != FORMAT_NONE ||  
    DevParams.ImagePropertiesRear2.Format != FORMAT_NONE )  
{  
    ImagesStruct *pImages = (ImagesStruct *)LParam;  
    DocID = pImages->DocNumber;  
}  
else  
{  
    DocID = (DWORD)LParam;  
}  
...  
break;  
}
```

The **WMPAR_SORTER_EXCEPTION** message signals an exception in the API. *LParam* is composed of:

Bit15-Bit0: Exception code

Bit31-Bit16: Document ID (if the exception is related to a document, otherwise is zero).

Full pocket handling

The VisionS has the capability to detect is the full pocket status for each pocket and introduces the Full pocket warning message.

The full pocket detection is signalled to the application by a **WMPAR_SORTER_EXCEPTION** message with the exception code equal to **DEVICE_ERR_FULL_POCKET**.

During this message the application have to call the function **GetFullPocketStatus** to retrieve the full pocket sensors status of the device. Refer to the reference manual for a description of the function.

There are two kind of full pocket messages:

3. Full pocket warning

It means that a pocket has become almost full. The VisionS device doesn't stop the process. Calling the **GetFullPocketStatus** the application get the actual status and it can exclude the almost full pocket from the sorting, allowing the operator to remove from the pocket. All this operations can be executed without stopping the process.

If after this warning the app continue to put documents in the "almost full" pockets the stop the job after N documents signalling a stop for Full Pocket condition.

N is a firmware parameter.

4. Full pocket

It means that a pocket is completely full. The VisionS device automatically stops the process completing the pending documents and the device state returns to OnLine.

Flow-Mode remarks

The Flow-Mode setting is different from the Start/Stop mode, because it implies the following features:

- Fixed DPM (application independent)
The device throughput (DPM) is managed by the firmware and is fixed. It means that the throughput is application independent. This permits to reach throughput greater than 200 DPM.
- Stop in View Station is not allowed
If the application keeps a long time to decide the destination pocket, the document in the track reach the last sensor at the beginning of sorters without a destination and the device rise an exception “Document without destination”, because the firmware never stop the document to wait the pocket decision.

The Flow-Mode is a critical process because the device has to manage a lot of documents in the track and the application must keep the device feeding pace. This implies that the PC has to commit most of the CPU resources to the VisionS process (as MICR and Image processing are made by the CPU) and the application has to call the SetPocket function as soon as possible when it receives the **SET_ITEM_OUPUT** message.

The document messages sent to the application (NewDoc, SetItemOutput...) are exactly the same of Start/Stop mode.

The Flow-mode introduces a list of limitations. The followings options are not allowed:

- Hand-drop feeder;
- MICR+OCR;
- OCR.

Firmware parameters

Like in MyVisionX, VisionS has a set of firmware parameters managed by the functions:

- *SetSorterParameter*
- *GetSorterParameter*

Here are the parameters:

0. Enable double-feeding detection
1. Power for double-feeding
2. Delay for double-feeding detection
3. Confidence for double-feeding
4. Hole filter
5. Delay to start Endorser
6. Max symbols in MICR codeline
7. Min document length
8. Max document length
9. Enable full-pocket detection
10. Full pocket stop
11. Full pocket polling
12. Transport Error

- 13. Move Separator(Dis,Every N Docs)
- 14. Move Separator period (Docs)
- 15. Free track destination pocket
- 16. Default dest pocket (0=Dis)
- 17. Stop with Default pocket
- 18. Delay for Stop on View-Station
- 19. Delay Stop motor doc on View-Station
- 20. Enable Button

Appendix C: I:Deal

This appendix is an integration of the VisionAPI reference manual. It explains how to use VisionAPI interface with an I:Deal device.

The structure of the software remains the same as the one for Vision X or Vision S, the upper interface layer will be the Panini Vision API library (VisionApi.dll) and the lower layer of the driver will be the engine of the device itself (IDEngine.dll). The engine will interface with the optional libraries for the OCR and other additional features.

The interface will use the majority of the Vision X function calls given that the device state structure will remain the same.

The main difference from the Vision X is that I:Deal is a single document device that has upstream and downstream devices.

The upstream devices are:

- MICR
- Image capture.

The downstream devices are:

- Franking: on the front of the check;
- Destination “pocket”: the doc can be sent to the pocket or returned in the feeder.

The application can decide is enable or not the franking operation based on the MICR and the Front image snippets.

The document can be sent to the pocket or returned to the feeder, it depends on the applications.

The front or rear image can be used to decide the destination of the document: send it back to the feeder can be useful in case of errors (ex: MICR rejects) during the processing, but it's up to the application to define it.

The device can acquire the Front image after the franking of the document to let the application stores the image of the franked document.

Document life cycle

The document life cycle has been maintained as similar as possible to the one of the Vision X.

Call *StartFeeding* function to begin the processing of the documents, *StopFeeding* to end it.

If the device is working in One Document mode or if an exception occurred, it is not necessary to call *StopFeeding*.

During the document transport, if a jam occurs, the operator cannot use the *FreeTrack* function to eject the document from the track. The document must be manually removed by the operator.

The progress of the document in the track is followed by the messages received from the application as described below:

WMPAR_SORTER_NEW_DOCUMENT is sent when the document enters the track, the LPARAM of this message reports the document number.

WMPAR_SORTER_MICR_AVAILABLE is sent when the MICR Codeline has been fully recognized (if it had been requested), and it is ready for the application to use; to obtain it call the *GetMicrCodeline* function. The LPARAM of this message reports the document number.

WMPAR_SNIPPET_READY is sent when the snippet is ready for the application. This message will be received twice, the first time for the front snippets, and the second time for the rear ones; in this way, the front snippets are available very quickly for OCR recognition without waiting for the rear ones. LPARAM field of this message is a structure similar to the one used for the images.

WMPAR_SET_DOWNSTREAM_OPTIONS is sent after the MICR and/or the front snippets message. During this message the application is able to call the *EnableFranking()* function in order to activate the franking process. This is a specific message of the I:Deal device.

WMPAR_SORTER_SET_ITEM_OUTPUT is sent when the document is ready to be pocketed. This message can be sent after the MICR codeline and/or the front snippet (if decision is required) and/or the rear snippet (if decision is requested). This because in some condition it will be necessary to do an action downstream respect the front image and other downstream respect the rear image.

Call the *SetPocket* function to set the destination pocket (because there will be the possibility of send the document back to the feeder setting the pocket to 2, 1 is the physical pocket). The LPARAM of this message reports the document number.

WMPAR_SORTER_DOCUMENT_INSIDE_POCKET is sent when the document has been pocketed. The LPARAM of this message reports the document number.

WMPAR_IMAGE_READY is sent when the images from the scanners are available for the application. LPARAM returns a structure containing the requested images (see the description of Image and Snippet Structure).

WMPAR_DOC_COMPLETED is sent when all the processing phases have been completed. LPARAM is the document number.

The sequence of the message can vary in the following way.

When a snippet is requested with ENABLE only, the sequence of the document messages is:

- NEW_DOC;
- **SNIPPET_READY (FRONT);**
- MICR;
- **SET_DOWNSTREAM_OPTIONS;**
- **SET_ITEM_OUTPUT;**

- IN_POCKET;
- **SNIPPET_READY (REAR);**
- IMAGE_READY;
- DOC_COMPLETE.

The position of the SNIPPET_READY(FRONT) message is unpredictable. It depends on the system where the software is running. It could be sent before or after the SET_DOWNSTREAM_OPTIONS and SET_ITEM_OUPUT message.

Using the ENABLE_FOR_DECISION flag for the front snippet the messages arrive in the following order:

- NEW_DOC;
- **SNIPPET_READY (FRONT);**
- MICR;
- **SET_DOWNSTREAM_OPTIONS;**
- **SET_ITEM_OUTPUT;**
- IN_POCKET;
- IMAGE_READY;
- DOC_COMPLETE.

In this case the SNIPPET_READY(FRONT) message is sent surely before the SET_DOWNSTREAM_OPTIONS message.

Using the ENABLE_FOR_DECISION for the front and rear snippets the sequence will be:

- NEW_DOC;
- **SNIPPET_READY (FRONT);**
- MICR;
- **SET_DOWNSTREAM_OPTIONS;**
- **SNIPPET_READY (REAR);**
- **SET_ITEM_OUTPUT;**
- IN_POCKET;
- IMAGE_READY;
- DOC_COMPLETE.

As you can see the SNIPPET_READY(REAR) message is sent after the SET_DOWNSTREAM_OPTIONS and before the SET_ITEM_OUTPUT, in order to decide the destination of the doc based on the rear info, too.

DeviceParameters

The structure DeviceParameters has remained unchanged.

The I:Deal introduces the possibility to acquire the image after a frontal franking has been added to the document. To get those images the FRANKED_IMAGE flag need to be set.

For example setting the Format of the image to *ImagePropertiesFront1.Format* = FORMAT_JPEG | FRANKED_IMAGE, it will return the image for the front after the franking compressed in Jpeg format.

With this option enabled, the application must call the EnableFranking function; otherwise it will receive an exception.

The application is able to request Front1 without the FRANKED_IMAGE and Front2 with and vice versa.

SetPocket

This function must be called during the SET_ITEM_OUTPUT message. It's the last API function to be called during that message. The must call this function in order to set the destination of the processed document.

There are two possible options:

- Pocket #1
The doc is sent in the pocket
- Pocket #2
The doc is returned in the feeder.

Compressed Images for Decision

The I:Deal introduces the capability of make downstream decision based on the compressed image data.

This option could be used to decide whether toenable the franking option on a document after the result of the IQA analysis on the compressed images.

In order to enable this behaviour the user application has to add to the image format a new flag. It's defined in the Vision API header file and its definition is the following:

```
#define COMP_IMAGE_FOR_DECISION 0x00020000
```

The application can decide to use only the front image or both the front and rear image. The following examples explain how to do that.

Example: Use front image

```
DEVICE_PARAMETERS DevParms;  
DevParms.ImagePropertiesFront1.Format = FORMAT_GIV;  
DevParms.ImagePropertiesFront1.Format |= COMP_IMAGE_FOR_DECISION;  
DevParms.ImagePropertiesRear1.Format = FORMAT_GIV;
```

Example: Use both images

```
DEVICE_PARAMETERS DevParms;  
DevParms.ImagePropertiesFront1.Format = FORMAT_GIV;  
DevParms.ImagePropertiesFront1.Format |= COMP_IMAGE_FOR_DECISION;  
DevParms.ImagePropertiesRear1.Format = FORMAT_GIV;  
DevParms.ImagePropertiesRear1.Format |= COMP_IMAGE_FOR_DECISION;
```

This feature introduces a new message. It's defined as follow:

```
#define WMPAR_COMPRESSED_IMG_FOR_DECISION 13
```

This message can be sent optionally based on the application settings. It is sent when the compressed images for decision are less than the total compressed images.

The LPARAM of the message represents a pointer to an ***ImagesStruct*** structure. It is the same structure used by the WMPAR_IMAGES_READY message. The application shall release the image structure by calling `FreeImagesBuffer` on the received pointer when finished to use the images.

This option implies also a change of the device behaviour. If the application uses only the front image data to make decisions, the franking operation is done during the 2nd paper pass (from the pocket towards the feeder).

If it uses also the rear data, the franking is done during the 3rd pass (from the feeder towards the pocket).

The following examples explain all the possible scenarios and the resulting sequence of the messages sent to the application:

Example #1			The Front 1 image is to be Evaluated Rear is disabled		
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)					
		Front 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION		
		Rear 1	-		
		Front 2	FORMAT_JPEG		
		Rear 2	-		
WMPAR CallBack Messages Sequence			Images Returned For The Decision Process		Images Returned Outside of the Decision Process
WMPAR_SORTER_NEW_DOCUMENT					
WMPAR_SORTER_MICR_AVAILABLE					
WMPAR_COMPRESSED_IMG_FOR_DECISION			Front 1 Image Front 2 Image		
WMPAR_IMAGES_READY					Front 1 Image Front 2 Image
WMPAR_SET_DOWNSTREAM_OPTIONS					
WMPAR_SORTER_SET_ITEM_OUTPUT					
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET					
WMPAR_DOC_COMPLETED					

Example #2		The Front 1 image is to be Evaluated The Front 2 image is to be Franked Rear is disabled	
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)			
	Front 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION	
	Rear 1	-	
	Front 2	FORMAT_JPEG + FRANKED_IMAGE	
	Rear 2	-	
WMPAR CallBack Messages Sequence		Images Returned For The Decision Process	Images Returned Outside of the Decision Process
WMPAR_SORTER_NEW_DOCUMENT			
WMPAR_SORTER_MICR_AVAILABLE			
WMPAR_COMPRESSED_IMG_FOR_DECISION		Front 1 Image	
WMPAR_SET_DOWNSTREAM_OPTIONS			
WMPAR_SORTER_SET_ITEM_OUTPUT			
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET			
WMPAR_IMAGES_READY			Front 1 Image Front 2 Image (Franked)
WMPAR_DOC_COMPLETED			

Example #3		The Front 1 image is to be Evaluated Rear is enabled	
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)			
	Front 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION	
	Rear 1	FORMAT_GIV	
	Front 2	FORMAT_JPEG	
	Rear 2	FORMAT_JPEG	
WMPAR CallBack Messages Sequence		Images Returned For The Decision Process	Images Returned Outside of the Decision Process
WMPAR_SORTER_NEW_DOCUMENT			
WMPAR_SORTER_MICR_AVAILABLE			
WMPAR_COMPRESSED_IMG_FOR_DECISION		Front 1 Image Front 2 Image	

Panini Vision API 4.5.0
Reference Manual Revision 1

WMPAR_SET_DOWNSTREAM_OPTIONS		
WMPAR_SORTER_SET_ITEM_OUTPUT		
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET		
WMPAR_IMAGES_READY		Front 1 Image Rear 1 Image Front 2 Image Rear 2 Image
WMPAR_DOC_COMPLETED		

Example #4 <div> The Front 1 image is to be Evaluated The Rear 1 image is to be Evaluated </div>		
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)		
	Front 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION
	Rear 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION
	Front 2	FORMAT_JPEG
	Rear 2	FORMAT_JPEG
WMPAR CallBack Messages Sequence	Images Returned For The Decision Process	Images Returned Outside of the Decision Process
WMPAR_SORTER_NEW_DOCUMENT		
WMPAR_SORTER_MICR_AVAILABLE		
WMPAR_IMAGES_READY		Front 1 Image Rear 1 Image Front 2 Image Rear 2 Image
WMPAR_SET_DOWNSTREAM_OPTIONS		
WMPAR_SORTER_SET_ITEM_OUTPUT		
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET		
WMPAR_DOC_COMPLETED		

NOTE: No WMPAR_COMPRESSED_IMG_FOR_DECISION message will be received since all images (Front and Rear) have been selected for evaluation, therefore the WMPAR_IMAGES_READY is sent before decision takes place.

Example #5		The Front 1 image is to be Evaluated The Rear 1 image is to be Evaluated The Front 2 image is to be Franked	
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)			
	Front 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION	
	Rear 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION	
	Front 2	FORMAT_JPEG + FRANKED_IMAGE	
	Rear 2	FORMAT_JPEG	
WMPAR CallBack Messages Sequence		Images Returned For The Decision Process	Images Returned Outside of the Decision Process
WMPAR_SORTER_NEW_DOCUMENT			
WMPAR_SORTER_MICR_AVAILABLE			
WMPAR_COMPRESSED_IMG_FOR_DECISION		Front 1 Image Rear 1 Image Rear 2 Image	
WMPAR_SET_DOWNSTREAM_OPTIONS			
WMPAR_SORTER_SET_ITEM_OUTPUT			
WMPAR_IMAGES_READY			Front 1 Image Rear 1 Image Front 2 Image (Franked) Rear 2 Image
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET			
WMPAR_DOC_COMPLETED			

Example #6		The Rear 1 image is to be Evaluated The Front 1 image is to be Franked	
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)			
	Front 1	FORMAT_GIV + FRANKED_IMAGE	
	Rear 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION	
	Front 2	FORMAT_JPEG	
	Rear 2	FORMAT_JPEG	
WMPAR CallBack Messages Sequence		Images Returned For The Decision Process	Images Returned Outside of the Decision Process
WMPAR_SORTER_NEW_DOCUMENT			

Panini Vision API 4.5.0
Reference Manual Revision 1

WMPAR_SORTER_MICR_AVAILABLE		
WMPAR_COMPRESSED_IMG_FOR_DECISION	Rear 1 Image Front 2 Image Rear 2 Image	
WMPAR_SET_DOWNSTREAM_OPTIONS		
WMPAR_SORTER_SET_ITEM_OUTPUT		
WMPAR_IMAGES_READY		Front 1 Image (Franked) Rear 1 Image Front 2 Image Rear 2 Image
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET		
WMPAR_DOC_COMPLETED		

Example #7		The Front 2 image is to be Evaluated The Front 1 image is to be Franked	
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)			
	Front 1	FORMAT_GIV + FRANKED_IMAGE	
	Rear 1	FORMAT_GIV	
	Front 2	FORMAT_JPEG + COMP_IMAGE_FOR_DECISION	
	Rear 2	FORMAT_JPEG	
WMPAR CallBack Messages Sequence		Images Returned For The Decision Process	Images Returned Outside of the Decision Process
WMPAR_SORTER_NEW_DOCUMENT			
WMPAR_SORTER_MICR_AVAILABLE			
WMPAR_COMPRESSED_IMG_FOR_DECISION		Front 2 Image	
WMPAR_SET_DOWNSTREAM_OPTIONS			
WMPAR_SORTER_SET_ITEM_OUTPUT			
WMPAR_IMAGES_READY			Front 1 Image (Franked) Rear 1 Image Front 2 Image Rear 2 Image
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET			
WMPAR_DOC_COMPLETED			

Sorter errors

The I-Deal has its own errors codes:

The error word (4 bytes long) is composed of the following fields:

Byte 3 (MSB) - Motor stall condition

Byte 2 - Error code

Byte 1 - Pass index

Byte 0 (LSB) - Sensor index

Motor stall condition

This field indicates that the error is due to a stall condition of the motor.

1 means stall condition detected

Error Codes

ERROR LABEL	ERROR CODE	DESCRIPTION
ERR_FEED_FAILURE	1	Feeding operations failed
ERR_DOC_LOST	2	A doc has not reached a certain position of the paper track
ERR_DOC_TOO_LONG	3	The fw detected a too long document in front of a certain sensor
ERR_DOC_TOO_SHORT	4	The fw detected a short document
ERR_BUSY_TRACK	5	The device cannot feed documents because the track is busy.
ERR_SCANNER_FLIP	6	CIS flip process failed
ERR_MULTIPLE_FEED	7	The fw detected a multiple feed attempt.

Pass index

This field indicates in which pass the error occurred.

PASS LABEL	PASS CODE
First	1
Second	2
Third	3

Sensor index

This field indicates on which sensor the error occurred.

SENSOR LABEL	SENSOR CODE
Feeder	0
Track #1	1
Track #2	2
Alignment	3

Connection attempt timeout

The I:Deal has a connection timeout of 1 second.

Exception errors

When the I:Deal is processing documents (Feeding state) and an exception is triggered, the API automatically stops the processing returning to the OnLine state.

There are some exceptions that do not stop the feeding.

They are included in the following table:

EXCEPTION ERROR LABEL	DESCRIPTION
DEVICE_ERR_COMPRESSION_ERR	Triggered when there is a problem processing the image data. The IMAGES_READY message will be sent the same and the ImagesStruct structure will contain NULL pointers instead of the image data. Example: Front1 and Rear1 enabled. Front1 compression failed. The Front1 field will be NULL. The Rear1 field will contain a valid data.
DEVICE_ERR_MISSING_FRANKING	Triggered when a franked image is requested and the <i>EnableFranking</i> call is skipped by the application
DEVICE_ERR_FRANKING_FAILED	Triggered when the firmware detects a problem during the franking operations.

Appendix D: VisionX

Compressed Images for Decision

The VisionX is able to make downstream decision based on the compressed image data. This option could be used to decide whether to enable the smart printing on a document or select the destination pocket (on a 2-pocket device) after the result of the IQA analysis on the compressed images.

In order to enable this behaviour the user application has to add to the image format a new flag. It's defined in the Vision API header file and its definition is the following:

```
#define COMP_IMAGE_FOR_DECISION    0x00020000
```

The following examples explain how to enable the compressed images for decision.

Example:

```
DEVICE_PARAMETERS DevParms;  
DevParms.ImagePropertiesFront1.Format = FORMAT_GIV;  
DevParms.ImagePropertiesFront1.Format |= COMP_IMAGE_FOR_DECISION;  
DevParms.ImagePropertiesRear1.Format = FORMAT_GIV;
```

NOTE: just adding the flag to a single image format will enable all the images for decision.

This option implies also a change of the device behaviour.

The following example explains the resulting sequence of the messages sent to application.

Example #1		The Front 1 image is to be Evaluated Rear is disabled	
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)			
	Front 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION	
	Rear 1	-	
	Front 2	FORMAT_JPEG	
	Rear 2	-	
WMPAR CallBack Messages Sequence		Images Returned For The Decision Process	Images Returned Outside of the Decision Process
WMPAR_SORTER_NEW_DOCUMENT			
WMPAR_SORTER_MICR_AVAILABLE			
WMPAR_IMAGES_READY		Front 1 Image Front 2 Image	-

Panini Vision API 4.5.0
Reference Manual Revision 1

WMPAR_SORTER_SET_ITEM_OUTPUT		
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET		
WMPAR_DOC_COMPLETED		

Example #3 The Front 1 image is to be Evaluated Rear is enabled		
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)		
	Front 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION
	Rear 1	FORMAT_GIV
	Front 2	FORMAT_JPEG
	Rear 2	FORMAT_JPEG
WMPAR Callback Messages Sequence	Images Returned For The Decision Process	Images Returned Outside of the Decision Process
WMPAR_SORTER_NEW_DOCUMENT		
WMPAR_SORTER_MICR_AVAILABLE		
WMPAR_IMAGES_READY	Front 1 Image Rear 1 Image Front 2 Image Rear 2 Image	-
WMPAR_SORTER_SET_ITEM_OUTPUT		
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET		
WMPAR_DOC_COMPLETED		

NOTE: No WMPAR_COMPRESSED_IMG_FOR_DECISION message will be received since all images (Front and Rear) have been selected for evaluation, therefore the WMPAR_IMAGES_READY is sent before decision takes place.

Example #4 The Front 1 image is to be Evaluated The Rear 1 image is to be Evaluated		
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)		
	Front 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION
	Rear 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION
	Front 2	FORMAT_JPEG
	Rear 2	FORMAT_JPEG
WMPAR Callback Messages Sequence	Images Returned For The Decision	Images Returned Outside of the Decision

	Process	Process
WMPAR_SORTER_NEW_DOCUMENT		
WMPAR_SORTER_MICR_AVAILABLE		
WMPAR_IMAGES_READY	Front 1 Image Rear 1 Image Front 2 Image Rear 2 Image	-
WMPAR_SORTER_SET_ITEM_OUTPUT		
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET		
WMPAR_DOC_COMPLETED		

NOTE: No WMPAR_COMPRESSED_IMG_FOR_DECISION message will be received since all images (Front and Rear) have been selected for evaluation, therefore the WMPAR_IMAGES_READY is sent before decision takes place.

Devices with maximum DPM greater than 100

VisionX 125 DPM has the capability to run up to 125 DPM.

In order to achieve the maximum throughput the application has to set the parameters as the following list suggests:

- One of these image formats:
FORMAT_JPEG
FORMAT_GIV
- The maximum image resolution is 200 dpi.
A greater resolution leads to a throughput lower than 125 DPM.

Recapture image after printing

The VisionX is able to recapture the rear images (Rear1, Rear2 or both) after the endorsement printing. This can be done only if the SmartJet mode has been selected and least one snippet *for decision* has been enabled.

In order to enable this behaviour the user application has to add to the image format a new flag. It's defined in the Vision API header file and its definition is the following:

```
#define PRINTED_IMAGE          0x00040000
```

The following examples explain how to enable the printed image recapture.

Example:

```
DEVICE_PARAMETERS DevParms;
DevParms.ImagePropertiesFront1.Format = FORMAT_GIV;
DevParms.ImagePropertiesRear1.Format = FORMAT_GIV;
DevParms.ImagePropertiesRear2.Format = FORMAT_GIV;
DevParms.ImagePropertiesRear2.Format |= PRINTED_IMAGE;
```

The first rear image (Rear1) is captured before the printing (as usual), while the second one (Rear2) is recaptured and contains the printed text.

GetDeviceFeature

BOOL *GetDeviceFeature*(DWORD DeviceID, DWORD FeatureID, LPVOID
lpReturnedBuffer, DWORD BufferSize)

This function, when called with DEVICE_FEATURE_FEEDER_SD feature ID value, is used to obtain information on which exact VisionX device version is connected.

Possible return values are:

DEVICE_FEATURE_NO	(0):	Vision X
DEVICE_FEATURE_YES	(1):	Vision SD
DEVICE_FEATURE_V1	(2):	Vision 1
DEVICE_FEATURE_Vx1B	(3):	Vision 1B

Appendix E: wI:Deal

This appendix is an integration of the VisionAPI reference manual. It explains how to use VisionAPI interface with a wI:Deal device.

The structure of the software remains the same as the one for Vision X, the upper interface layer will be the Panini Vision API library (VisionApi.dll) and the lower layer of the driver will be the engine of the device itself (WiEngine.dll). The engine will interface with the optional libraries for the OCR and other additional features.

The interface will use the majority of the Vision X function calls given that the device state structure will remain the same.

The wI:Deal is a single document device (as the I:Deal) that has the capability to handle automatically 3 different types of documents:

1. Cheques
2. Pages (A4/Letter/Legal formats)
3. Rigid documents (Image capture only)

Here is a table that shows which the processing capabilities for each kind of document:

Processing	Cheque	Page	Rigid
MICR	X	n/a	n/a
Image capture	X	X	X
Franking	X	X	n/a
Printer	X	X	n/a
Pocket	X	X	n/a
Feeder Auto alignment	X	X	n/a

Most of the device parameters for each document type can be set independently.

EX: Set a different image capture format for each type of document:

- Cheque: G4, 200 dpi
- Page: JPEG True color 200 dpi
- Rigid: JPEG Gray Level 200 dpi

The behaviour of the device is essentially the same of the I:Deal. It is a single device that has upstream and devices.

The upstream devices are:

- MICR
- Image capture.

The downstream devices are:

- Franking: on the front of the check
It's available for cheques and pages.
- Printer: on the rear of the document
It's available for cheques and pages.

- Destination “pocket”: the doc can be sent to the pocket or returned in the feeder.
The rigid document type must always be returned in the feeder.

In order to process the rigid documents the user must open the door located on the rear side of the device. When the door is open a status LED turns on to show to the user that the device has detected the rigid document mode condition.

Here is a table that explains how the position of the rear door affects the device processing capabilities:

Door Status	Cheque	Page	Rigid
Close	Allowed	Allowed	Not Allowed
Open	Allowed	Not allowed	Allowed

The device is always capable of processing cheques.

When a Page document is detected the device check the door status that must closed. If not the device stops the processing to avoid to damage the document.

When a Rigid document is detected the device check the door status that must open. If not the device stops the processing to avoid to damage the document and the device.

The document can be sent to the pocket or returned to the feeder, it depends on the applications.

The front or rear image can be used to decide the destination of the document: send it back to the feeder can be useful in case of errors (ex: MICR rejects) during the processing, but it's up to the application to define it.

The device can acquire the Front image after the franking of the document to let the application stores the image of the franked document.

Document life cycle

The document life cycle has been maintained as similar as possible to the one of the Vision X.

Call *StartFeeding* function to begin the processing of the documents, *StopFeeding* to end it.

If the device is working in One Document mode or if an exception occurred, it is not necessary to call *StopFeeding*.

During the document transport, if a jam occurs, the operator cannot use the *FreeTrack* function to eject the document from the track. The document must be manually removed by the operator.

The document messages cycle the behaviour of the application will be different based on the document type under processing. Using the *GetDocumentType()* function the app is always aware of the kind of document under processing.

The progress of the document in the track is followed by the messages received from the application as described below:

WMPAR_SORTER_NEW_DOCUMENT is sent when the document enters the track, the LPARAM of this message reports the document number.

During this message is possible to call the function *GetDocumentType()* to understand if the document type (cheque, page or rigid).

WMPAR_SORTER_MICR_AVAILABLE is sent when the MICR Codeline has been fully recognized (if it had been requested), and it is ready for the application to use; to obtain it call the *GetMicrCodeline* function. The LPARAM of this message reports the document number.

WMPAR_SNIPPET_READY is sent when the snippet is ready for the application. This message will be received twice, the first time for the front snippets, and the second time for the rear ones; in this way, the front snippets are available very quickly for OCR recognition without waiting for the rear ones. LPARAM field of this message is a structure similar to the one used for the images.

WMPAR_SET_DOWNSTREAM_OPTIONS is sent after the MICR and/or the front snippets message. During this message the application is able to call the *EnableFranking()* function in order to activate the franking process. This is a specific message of the wI:Deal device. The application is able, thanks to the *GetDocumentType()* function, to decide if enable or not the franking.

EX:

The document type is a check, it will be franked. If not, the franking won't be activated.

The document type is a page or a check, it will be printed. If it is a card an empty string will be passed to the *SendPrinterData* function.

WMPAR_SORTER_SET_ITEM_OUTPUT is sent when the document is ready to be pocketed. This message can be sent after the MICR codeline and/or the front snippet (if decision is required) and/or the rear snippet (if decision is requested). This because in some condition it will be necessary to do an action downstream respect the front image and other downstream respect the rear image.

Call the *SetPocket* function to set the destination pocket (because there will be the possibility of send the document back to the feeder setting the pocket to 2, 1 is the physical pocket). The LPARAM of this message reports the document number. For rigid documents the application must always set the destination pocket to 2. No other settings are allowed.

WMPAR_SORTER_DOCUMENT_INSIDE_POCKET is sent when the document has been pocketed. The LPARAM of this message reports the document number.

WMPAR_IMAGE_READY is sent when the images from the scanners are available for the application. LPARAM returns a structure containing the requested images (see the description of Image and Snippet Structure).

WMPAR_DOC_COMPLETED is sent when all the processing phases have been completed. LPARAM is the document number.

The sequence of the message can vary in the following way.

When a snippet is requested with ENABLE only, the sequence of the document messages is:

- NEW_DOC;
- SNIPPET_READY (FRONT);
- MICR;

- **SET_DOWNSTREAM_OPTIONS;**
- **SET_ITEM_OUTPUT;**
- IN_POCKET;
- **SNIPPET_READY (REAR);**
- IMAGE_READY;
- DOC_COMPLETE.

The position of the SNIPPET_READY(FRONT) message is unpredictable. It depends on the system where the software is running. It could be sent before or after the SET_DOWNSTREAM_OPTIONS and SET_ITEM_OUTPUT message.

Using the ENABLE_FOR_DECISION flag for the front snippet the messages arrive in the following order:

- NEW_DOC;
- **SNIPPET_READY (FRONT);**
- MICR;
- **SET_DOWNSTREAM_OPTIONS;**
- **SET_ITEM_OUTPUT;**
- IN_POCKET;
- IMAGE_READY;
- DOC_COMPLETE.

In this case the SNIPPET_READY(FRONT) message is sent surely before the SET_DOWNSTREAM_OPTIONS message.

Using the ENABLE_FOR_DECISION for the front and rear snippets the sequence will be:

- NEW_DOC;
- **SNIPPET_READY (FRONT);**
- MICR;
- **SET_DOWNSTREAM_OPTIONS;**
- **SNIPPET_READY (REAR);**
- **SET_ITEM_OUTPUT;**
- IN_POCKET;
- IMAGE_READY;
- DOC_COMPLETE.

As you can see the SNIPPET_READY(REAR) message is sent after the SET_DOWNSTREAM_OPTIONS and before the SET_ITEM_OUTPUT, in order to decide the destination of the doc based on the rear info, too.

DeviceParameters

The structure DeviceParameters has remained unchanged.

The wI:Deal device, being a multi-document device, introduces the capability of settings different device parameters for each document type.

The *SetDocProcessingParam()* function has been created for this purpose.

The application could use two different approach setting the parameters:

- Use *SetDeviceParameters()*
This function sets the same parameters values for the entire set of document types.

- Use *SetDocProcessingParam()*

This function is able to set a different parameter value for each document.

To speed up the development the developer can call the *SetDeviceParameter()* to create a common settings among the documents types, then he can call the *SetDocProcessingParam()* to specialize the processing based on the document type.

There are some parameters that cannot specialize. Here they are:

- OneDoc flag
- Feeding mode
- Printer settings

When an application sets one of them for a certain document type, it is internally copied to the other document settings.

The suggested setting for the printer is to always set the Smart-Jet options. This gives the maximum flexibility because the application is able to decide the printing data based on the document type.

When the application receives the SET_ITEM_OUTPUT message, it can use the document type info, along with the others (MICR and OCR), to set the right data to print on the document.

If the printer is enabled, the application must call *SendPrinterData()* for each type of document and can skip the printing passing an empty string.

EX:

If the app has to print on cheques only and a page is processed by the user, it can call *SendPrinterData()* with an empty string.

The wI:Deal has the capability to acquire the image after a frontal franking has been added to the document. To get those images the FRANKED_IMAGE flag need to be set.

For example setting the Format of the image to *ImagePropertiesFront1.Format = FORMAT_JPEG | FRANKED_IMAGE*, it will return the image for the front after the franking, compressed in Jpeg format.

With this option enabled, the application must call the EnableFranking function; otherwise it will receive an exception.

The application is able to request Front1 without the FRANKED_IMAGE and Front2 with and vice versa.

There is one limitation for the device settings. The application cannot set at the same time MICR and True colour acquisition for cheque.

SetPocket

This function must be called during the SET_ITEM_OUTPUT message. It's the last API function to be called during that message. The must call this function in order to set the destination of the processed document.

There are two possible options:

- Pocket #1
the doc is sent in the pocket.
It is allowed for cheque and page.

- Pocket #2
the doc is returned in the feeder.
It is allowed for all the documents types.

If a plastic card is under process (refer to the *GetDocumentType()* function), the application must set the pocket number 2.

Compressed Images for Decision

The wI:Deal introduces the capability to take downstream decision based on the compressed image data.

This option could be used to decide whether to enable the franking option on a document after the result of the IQA analysis on the compressed images.

In order to enable this behaviour the user application has to add to the image format a new flag. It's defined in the Vision API header file and its definition is the following:

```
#define COMP_IMAGE_FOR_DECISION    0x00020000
```

The application can decide to use only the front image or both the front and rear image. The following examples explain how to do that.

Example: Use front image

```
DEVICE_PARAMETERS DevParms;  
DevParms.ImagePropertiesFront1.Format = FORMAT_GIV;  
DevParms.ImagePropertiesFront1.Format |= COMP_IMAGE_FOR_DECISION;  
DevParms.ImagePropertiesRear1.Format = FORMAT_GIV;
```

Example: Use both images

```
DEVICE_PARAMETERS DevParms;  
DevParms.ImagePropertiesFront1.Format = FORMAT_GIV;  
DevParms.ImagePropertiesFront1.Format |= COMP_IMAGE_FOR_DECISION;  
DevParms.ImagePropertiesRear1.Format = FORMAT_GIV;  
DevParms.ImagePropertiesRear1.Format |= COMP_IMAGE_FOR_DECISION;
```

This feature introduces a new message. It's defined as follow:

```
#define WMPAR_COMPRESSED_IMG_FOR_DECISION    13
```

This message can be sent optionally based on the application settings. It is sent when the compressed images for decision are less than the total compressed images.

The LPARAM of the message represents a pointer to an **ImagesStruct** structure. It is the same structure used by the WMPAR_IMAGES_READY message. The application shall release the image structure by calling *FreeImagesBuffer* on the received pointer when finished to use the images.

This option implies also a change of the device behaviour. If the application uses only the front image data to make decisions, the franking operation is done during the 2nd paper pass (from the pocket towards the feeder).

If it uses also the rear data, the franking is done during the 3rd pass (from the feeder towards the pocket).

The following examples explain all the possible scenarios and the resulting sequence of the messages sent to the application:

Example #1		The Front 1 image is to be Evaluated Rear is disabled	
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)			
	Front 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION	
	Rear 1	-	
	Front 2	FORMAT_JPEG	
	Rear 2	-	
WMPAR CallBack Messages Sequence		Images Returned For The Decision Process	Images Returned Outside of the Decision Process
WMPAR_SORTER_NEW_DOCUMENT			
WMPAR_SORTER_MICR_AVAILABLE			
WMPAR_COMPRESSED_IMG_FOR_DECISION		Front 1 Image Front 2 Image	
WMPAR_IMAGES_READY			Front 1 Image Front 2 Image
WMPAR_SET_DOWNSTREAM_OPTIONS			
WMPAR_SORTER_SET_ITEM_OUTPUT			
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET			
WMPAR_DOC_COMPLETED			

Example #2		The Front 1 image is to be Evaluated The Front 2 image is to be Franked Rear is disabled	
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)			
	Front 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION	
	Rear 1	-	
	Front 2	FORMAT_JPEG + FRANKED_IMAGE	
	Rear 2	-	

Panini Vision API 4.5.0
Reference Manual Revision 1

WMPAR CallBack Messages Sequence	Images Returned For The Decision Process	Images Returned Outside of the Decision Process
WMPAR_SORTER_NEW_DOCUMENT		
WMPAR_SORTER_MICR_AVAILABLE		
WMPAR_COMPRESSED_IMG_FOR_DECISION	Front 1 Image	
WMPAR_SET_DOWNSTREAM_OPTIONS		
WMPAR_SORTER_SET_ITEM_OUTPUT		
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET		
WMPAR_IMAGES_READY		Front 1 Image Front 2 Image (Franked)
WMPAR_DOC_COMPLETED		

Example #3		The Front 1 image is to be Evaluated Rear is enabled	
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)			
	Front 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION	
	Rear 1	FORMAT_GIV	
	Front 2	FORMAT_JPEG	
	Rear 2	FORMAT_JPEG	
WMPAR CallBack Messages Sequence		Images Returned For The Decision Process	Images Returned Outside of the Decision Process
WMPAR_SORTER_NEW_DOCUMENT			
WMPAR_SORTER_MICR_AVAILABLE			
WMPAR_COMPRESSED_IMG_FOR_DECISION		Front 1 Image Front 2 Image	
WMPAR_SET_DOWNSTREAM_OPTIONS			
WMPAR_SORTER_SET_ITEM_OUTPUT			
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET			
WMPAR_IMAGES_READY			Front 1 Image Rear 1 Image Front 2 Image Rear 2 Image
WMPAR_DOC_COMPLETED			

Example #4		The Front 1 image is to be Evaluated The Rear 1 image is to be Evaluated	
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)			
	Front 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION	
	Rear 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION	
	Front 2	FORMAT_JPEG	
	Rear 2	FORMAT_JPEG	
WMPAR CallBack Messages Sequence		Images Returned For The Decision Process	Images Returned Outside of the Decision Process
WMPAR_SORTER_NEW_DOCUMENT			
WMPAR_SORTER_MICR_AVAILABLE			
WMPAR_IMAGES_READY			Front 1 Image Rear 1 Image Front 2 Image Rear 2 Image
WMPAR_SET_DOWNSTREAM_OPTIONS			
WMPAR_SORTER_SET_ITEM_OUTPUT			
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET			
WMPAR_DOC_COMPLETED			

NOTE: No WMPAR_COMPRESSED_IMG_FOR_DECISION message will be received since all images (Front and Rear) have been selected for evaluation, therefore the WMPAR_IMAGES_READY is sent before decision takes place.

Example #5		The Front 1 image is to be Evaluated The Rear 1 image is to be Evaluated The Front 2 image is to be Franked	
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)			
	Front 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION	
	Rear 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION	
	Front 2	FORMAT_JPEG + FRANKED_IMAGE	
	Rear 2	FORMAT_JPEG	
WMPAR CallBack Messages Sequence		Images Returned For The Decision Process	Images Returned Outside of the Decision Process
WMPAR_SORTER_NEW_DOCUMENT			

WMPAR_SORTER_MICR_AVAILABLE		
WMPAR_COMPRESSED_IMG_FOR_DECISION	Front 1 Image Rear 1 Image Rear 2 Image	
WMPAR_SET_DOWNSTREAM_OPTIONS		
WMPAR_SORTER_SET_ITEM_OUTPUT		
WMPAR_IMAGES_READY		Front 1 Image Rear 1 Image Front 2 Image (Franked) Rear 2 Image
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET		
WMPAR_DOC_COMPLETED		

Example #6		
The Rear 1 image is to be Evaluated The Front 1 image is to be Franked		
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)		
	Front 1	FORMAT_GIV + FRANKED_IMAGE
	Rear 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION
	Front 2	FORMAT_JPEG
	Rear 2	FORMAT_JPEG
WMPAR CallBack Messages Sequence	Images Returned For The Decision Process	Images Returned Outside of the Decision Process
WMPAR_SORTER_NEW_DOCUMENT		
WMPAR_SORTER_MICR_AVAILABLE		
WMPAR_COMPRESSED_IMG_FOR_DECISION	Rear 1 Image Front 2 Image Rear 2 Image	
WMPAR_SET_DOWNSTREAM_OPTIONS		
WMPAR_SORTER_SET_ITEM_OUTPUT		
WMPAR_IMAGES_READY		Front 1 Image (Franked) Rear 1 Image Front 2 Image Rear 2 Image
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET		
WMPAR_DOC_COMPLETED		

Example #7		The Front 2 image is to be Evaluated The Front 1 image is to be Franked	
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)			
	Front 1	FORMAT_GIV + FRANKED_IMAGE	
	Rear 1	FORMAT_GIV	
	Front 2	FORMAT_JPEG + COMP_IMAGE_FOR_DECISION	
	Rear 2	FORMAT_JPEG	
WMPAR CallBack Messages Sequence		Images Returned For The Decision Process	Images Returned Outside of the Decision Process
WMPAR_SORTER_NEW_DOCUMENT			
WMPAR_SORTER_MICR_AVAILABLE			
WMPAR_COMPRESSED_IMG_FOR_DECISION		Front 2 Image	
WMPAR_SET_DOWNSTREAM_OPTIONS			
WMPAR_SORTER_SET_ITEM_OUTPUT			
WMPAR_IMAGES_READY			Front 1 Image (Franked) Rear 1 Image Front 2 Image Rear 2 Image
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET			
WMPAR_DOC_COMPLETED			

Sorter errors

The wI:Deal has its own errors codes:

The error word (4 bytes long) is composed of the following fields:

Byte 3 (MSB) - Motor stall condition

Byte 2 - Error code

Byte 1 - Pass index

Byte 0 (LSB) - Sensor index

Motor stall condition

This field indicates that the error is due to a stall condition of the motor.

1 means stall condition detected

Error Codes

ERROR LABEL	ERROR CODE	DESCRIPTION
ERR_FEED_FAILURE	1	Feeding operations failed
ERR_DOC_LOST	2	A doc has not reached a certain position of the paper track
ERR_DOC_TOO_LONG	3	The fw detected a too long document in front of a certain sensor
ERR_DOC_TOO_SHORT	4	The fw detected a short document
ERR_BUSY_TRACK	5	The device cannot feed documents because the track is busy.
ERR_SCANNER_FLIP	6	CIS flip process failed
ERR_MULTIPLE_FEED	7	The fw detected a multiple feed attempt.
ERR_DOC_FORMAT	8	The fw detected a that a page is under processing and the door for rigid is open, of that a rigid doc is under processing and the rigid door is closed
ERR_PRINTER	9	The FW detected that the printer cannot be printed on the document under processing, because the it cannot be contained on the paper.

Pass index

This field indicates in which pass the error occurred.

PASS LABEL	PASS CODE
First	1
Second	2
Third	3

Sensor index

This field indicates on which sensor the error occurred.

SENSOR LABEL	SENSOR CODE
Alignment	0
Feeder	1
Rigid document	2
Track #1	3
Track #2	4
Page	5
Rigid track	6

Connection attempt timeout

The wI:Deal has a connection timeout of 1 second.

Exception errors

When the wI:Deal is processing documents (Feeding state) and an exception is triggered, the API automatically stops the processing returning to the OnLine state.

There are some exceptions that do not stop the feeding.
They are included in the following table:

EXCEPTION ERROR LABEL	DESCRIPTION
DEVICE_ERR_COMPRESSION_ERR	Triggered when there is a problem processing the image data. The IMAGES_READY message will be sent the same and the ImagesStruct structure will contain NULL pointers instead of the image data. Example: Front1 and Rear1 enabled. Front1 compression failed. The Front1 field will be NULL. The Rear1 field will contain a valid data.
DEVICE_ERR_MISSING_FRANKING	Triggered when a franked image is requested and the <i>EnableFranking</i> call is skipped by the application
DEVICE_ERR_FRANKING_FAILED	Triggered when the firmware detects a problem during the franking operations.

Appendix F: VisionX Page Scanner Extension

This appendix is an integration of the VisionAPI reference manual. It explains how to use VisionAPI interface with a VisionX Page Scanner Extension device.

The structure of the software remains the same as the one for Vision X, the upper interface layer will be the Panini Vision API library (VisionApi.dll) and the lower layer of the driver will be the engine of the device itself (VxA4Engine.dll). The engine will interface with the optional libraries for the OCR and other additional features.

The interface will use the majority of the Vision X function calls given that the device state structure will remain the same.

The VxA4 is a single document device (as the I:Deal) that has the capability to handle page documents.

It is capable of acquiring images. The other options are not available.
The processed document is always considered as a page.

Here is a table that shows which the processing capabilities for each kind of document:

Processing	Page
MICR	n/a
Image capture	X
Franking	n/a
Printer	n/a
Pocket	n/a
Feeder Auto alignment	X

The behaviour of the device is essentially the same of the wI:Deal. It is a single device that has upstream and devices.

The upstream devices are:

- Image capture.

The downstream devices are:

- Destination “pocket”: the doc can be sent to the pocket or returned in the feeder.

The document can be sent to the pocket or returned to the feeder, it depends on the applications.

The front or rear image can be used to decide the destination of the document.

Document life cycle

The document life cycle has been maintained as similar as possible to the one of the Vision X.

Call *StartFeeding* function to begin the processing of the documents, *StopFeeding* to end it.

If the device is working in One Document mode or if an exception occurred, it is not necessary to call *StopFeeding*.

During the document transport, if a jam occurs, the operator cannot use the *FreeTrack* function to eject the document from the track. The document must be manually removed by the operator.

The document messages cycle the behaviour of the application will be different based on the document type under processing.

The progress of the document in the track is followed by the messages received from the application as described below:

WMPAR_SORTER_NEW_DOCUMENT is sent when the document enters the track; the LPARAM of this message reports the document number.

WMPAR_SNIPPET_READY is sent when the snippet is ready for the application. This message will be received twice, the first time for the front snippets, and the second time for the rear ones; in this way, the front snippets are available very quickly for OCR recognition without waiting for the rear ones. LPARAM field of this message is a structure similar to the one used for the images.

WMPAR_SORTER_SET_ITEM_OUTPUT is sent when the document is ready to be pocketed. This message can be sent after the front snippet (if decision is required) and/or the rear snippet (if decision is requested). This because in some condition it will be necessary to do an action downstream respect the front image and other downstream respect the rear image.

Call the *SetPocket* function to set the destination pocket.

The LPARAM of this message reports the document number.

WMPAR_SORTER_DOCUMENT_INSIDE_POCKET is sent when the document has been pocketed. The LPARAM of this message reports the document number.

WMPAR_IMAGE_READY is sent when the images from the scanners are available for the application. LPARAM returns a structure containing the requested images (see the description of Image and Snippet Structure).

WMPAR_DOC_COMPLETED is sent when all the processing phases have been completed. LPARAM is the document number.

The sequence of the message can vary in the following way.

When a snippet is requested with ENABLE only, the sequence of the document messages is:

- NEW_DOC;
- **SNIPPET_READY (FRONT);**
- **SET_DOWNSTREAM_OPTIONS;**
- **SET_ITEM_OUTPUT;**
- IN_POCKET;
- **SNIPPET_READY (REAR);**
- IMAGE_READY;

- **DOC_COMPLETE.**

The position of the **SNIPPET_READY (FRONT)** message is unpredictable. It depends on the system where the software is running. It could be sent before or after the **SET_DOWNSTREAM_OPTIONS** and **SET_ITEM_OUPUT** message.

Using the **ENABLE_FOR_DECISION** flag for the front snippet the messages arrive in the following order:

- **NEW_DOC;**
- **SNIPPET_READY (FRONT);**
- **SET_DOWNSTREAM_OPTIONS;**
- **SET_ITEM_OUTPUT;**
- **IN_POCKET;**
- **IMAGE_READY;**
- **DOC_COMPLETE.**

In this case the **SNIPPET_READY (FRONT)** message is sent surely before the **SET_DOWNSTREAM_OPTIONS** message.

Using the **ENABLE_FOR_DECISION** for the front and rear snippets the sequence will be:

- **NEW_DOC;**
- **SNIPPET_READY (FRONT);**
- **SET_DOWNSTREAM_OPTIONS;**
- **SNIPPET_READY (REAR);**
- **SET_ITEM_OUTPUT;**
- **IN_POCKET;**
- **IMAGE_READY;**
- **DOC_COMPLETE.**

As you can see the **SNIPPET_READY (REAR)** message is sent after the **SET_DOWNSTREAM_OPTIONS** and before the **SET_ITEM_OUTPUT**, in order to decide the destination of the doc based on the rear info, too.

DeviceParameters

The structure **DeviceParameters** has remained unchanged.

SetPocket

This function must be called during the **SET_ITEM_OUTPUT** message. It's the last API function to be called during that message. The must call this function in order to set the destination of the processed document.

There are two possible options:

- **Pocket #1**
the doc is sent in the pocket.
- **Pocket #2**
the doc is returned in the feeder.

Compressed Images for Decision

The VxA4 introduces the capability of make downstream decision based on the compressed image data.

This option could be used to decide the pocketing of a document after the result of the IQA analysis on the compressed images.

In order to enable this behaviour the user application has to add to the image format a new flag. It's defined in the Vision API header file and its definition is the following:

```
#define COMP_IMAGE_FOR_DECISION    0x00020000
```

The application can decide to use only the front image or both the front and rear image. The following examples explain how to do that.

Example: Use front image

```
DEVICE_PARAMETERS DevParms;  
DevParms.ImagePropertiesFront1.Format = FORMAT_GIV;  
DevParms.ImagePropertiesFront1.Format |= COMP_IMAGE_FOR_DECISION;  
DevParms.ImagePropertiesRear1.Format = FORMAT_GIV;
```

Example: Use both images

```
DEVICE_PARAMETERS DevParms;  
DevParms.ImagePropertiesFront1.Format = FORMAT_GIV;  
DevParms.ImagePropertiesFront1.Format |= COMP_IMAGE_FOR_DECISION;  
DevParms.ImagePropertiesRear1.Format = FORMAT_GIV;  
DevParms.ImagePropertiesRear1.Format |= COMP_IMAGE_FOR_DECISION;
```

This feature introduces a new message. It's defined as follow:

```
#define WMPAR_COMPRESSED_IMG_FOR_DECISION    13
```

This message can be sent optionally based on the application settings. It is sent when the compressed images for decision are less than the total compressed images.

The LPARAM of the message represents a pointer to an *ImagesStruct* structure. It is the same structure used by the WMPAR_IMAGES_READY message. The application shall release the image structure by calling `FreeImagesBuffer` on the received pointer when finished to use the images.

This option implies also a change of the device behaviour

The following examples explain all the possible scenarios and the resulting sequence of the messages sent to the application:

Example #1	The Front 1 image is to be Evaluated Rear is disabled
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)	

Panini Vision API 4.5.0
Reference Manual Revision 1

Front 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION	
Rear 1	-	
Front 2	FORMAT_JPEG	
Rear 2	-	
WMPAR CallBack Messages Sequence	Images Returned For The Decision Process	Images Returned Outside of the Decision Process
WMPAR_SORTER_NEW_DOCUMENT		
WMPAR_SORTER_MICR_AVAILABLE		
WMPAR_COMPRESSED_IMG_FOR_DECISION	Front 1 Image Front 2 Image	
WMPAR_IMAGES_READY		Front 1 Image Front 2 Image
WMPAR_SET_DOWNSTREAM_OPTIONS		
WMPAR_SORTER_SET_ITEM_OUTPUT		
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET		
WMPAR_DOC_COMPLETED		

Example #3		The Front 1 image is to be Evaluated Rear is enabled	
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)			
	Front 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION	
	Rear 1	FORMAT_GIV	
	Front 2	FORMAT_JPEG	
	Rear 2	FORMAT_JPEG	
WMPAR CallBack Messages Sequence		Images Returned For The Decision Process	Images Returned Outside of the Decision Process
WMPAR_SORTER_NEW_DOCUMENT			
WMPAR_SORTER_MICR_AVAILABLE			
WMPAR_COMPRESSED_IMG_FOR_DECISION		Front 1 Image Front 2 Image	
WMPAR_SET_DOWNSTREAM_OPTIONS			
WMPAR_SORTER_SET_ITEM_OUTPUT			
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET			

WMPAR_IMAGES_READY		Front 1 Image Rear 1 Image Front 2 Image Rear 2 Image
WMPAR_DOC_COMPLETED		

Example #4		The Front 1 image is to be Evaluated The Rear 1 image is to be Evaluated	
Device Parameters for Image Format Settings: (Any Valid Image Format Can Be Specified, GIV and JPEG Are Used For Illustration Purposes only)			
	Front 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION	
	Rear 1	FORMAT_GIV + COMP_IMAGE_FOR_DECISION	
	Front 2	FORMAT_JPEG	
	Rear 2	FORMAT_JPEG	
WMPAR CallBack Messages Sequence		Images Returned For The Decision Process	Images Returned Outside of the Decision Process
WMPAR_SORTER_NEW_DOCUMENT			
WMPAR_SORTER_MICR_AVAILABLE			
WMPAR_IMAGES_READY			Front 1 Image Rear 1 Image Front 2 Image Rear 2 Image
WMPAR_SET_DOWNSTREAM_OPTIONS			
WMPAR_SORTER_SET_ITEM_OUTPUT			
WMPAR_SORTER_DOCUMENT_INSIDE_POCKET			
WMPAR_DOC_COMPLETED			

NOTE: No WMPAR_COMPRESSED_IMG_FOR_DECISION message will be received since all images (Front and Rear) have been selected for evaluation, therefore the WMPAR_IMAGES_READY is sent before decision takes place.

Sorter errors

The VxA4 has its own errors codes:

The error word (4 bytes long) is composed of the following fields:

Byte 3 (MSB) - Motor stall condition

Byte 2 - Error code

Byte 1 - Pass index

Byte 0 (LSB) - Sensor index

Motor stall condition

This field indicates that the error is due to a stall condition of the motor.

1 means stall condition detected

Error Codes

ERROR LABEL	ERROR CODE	DESCRIPTION
ERR_FEED_FAILURE	1	Feeding operations failed
ERR_DOC_LOST	2	A doc has not reached a certain position of the paper track
ERR_DOC_TOO_LONG	3	The fw detected a too long document in front of a certain sensor
ERR_DOC_TOO_SHORT	4	The fw detected a short document
ERR_BUSY_TRACK	5	The device cannot feed documents because the track is busy.
ERR_SCANNER_FLIP	6	CIS flip process failed
ERR_MULTIPLE_FEED	7	The fw detected a multiple feed attempt.

Pass index

This field indicates in which pass the error occurred.

PASS LABEL	PASS CODE
First	1
Second	2
Third	3

Sensor index

This field indicates on which sensor the error occurred.

SENSOR LABEL	SENSOR CODE
Alignment	0
Feeder	1
Rigid document	2
Track #1	3
Track #2	4

Connection attempt timeout

The VxA4 has a connection timeout of 1 second.

Exception errors

When the VxA4 is processing documents (Feeding state) and an exception is triggered, the API automatically stops the processing returning to the OnLine state.

There are some exceptions that do not stop the feeding.

They are included in the following table:

EXCEPTION ERROR LABEL	DESCRIPTION
DEVICE_ERR_COMPRESSION_ERR	Triggered when there is a problem processing the image data. The IMAGES_READY message will be sent the same and the ImagesStruct structure

	will contain NULL pointers instead of the image data. Example: Front1 and Rear1 enabled. Front1 compression failed. The Front1 field will be NULL. The Rear1 field will contain a valid data.
--	--

Gamma Correction

When scanning A4 pages that are not properly aligned during insertion or that are creased, the VxA4 is creating different shadows in the final output image due to "paper bubbles" happening during paper transport. This is more visible especially when white background documents are used (e.g. invoice or letters).

To minimize this bad behavior a Gamma Correction adjustment has been applied before compression in case of Grey Level or Black/White images only (no color).

To enable this Gamma Correction a new Image Correction bit shall be added to the Image Format selected:

```
#define IMG_CORRECTION_DOCTYPE_TXT 0x01000000
```

Appendix G: VisionNext

This appendix explains how to manage a VisionNext device with the VisionAPI interface.

The software structure remains the same as for the Vision X device. The upper interface layer is the Panini Vision API library (VisionApi.dll) and the lower layer is the device engine itself (X2Engine.dll). The X2 engine interfaces with the optional libraries for OCR and other additional features.

The device state structure remains the same.

The VisionNext has the capability to handle 2 different types of documents:

1. Cheques;
2. Rigid (ID card) documents.

The table below shows the processing capabilities for both types of documents:

Processing	Cheque	Rigid
MICR	X	n/a
Image capture	X	X
Printer	X	n/a
Pocket	X	n/a
Feeder Auto alignment	X	n/a

Device parameters for both document types are set independently using the SetDocProcessingParam function.

For example: Different image formats can be set for both types of documents:

- Cheque: G4, 200 dpi
- Rigid: JPEG Gray Level 300 dpi

The behaviour of the device is the same of the VisionX regarding the Checks processing. Rigid documents (ID cards) is an added capability.

Documents can be sent to the exit pocket or to the exception pocket, under the control of the application. The front or rear images, MICR information such as Routing/Transit numbers or reject characters and OCR information from image snippets can be used to make the pocketing decision.

Document life cycle

Select *StartFeeding* function to begin the processing of the documents, *StopFeeding* to end it. (If the device is working in One Document mode or an exception occurs, it is not necessary to select *StopFeeding*.)

If a jam occurs while the document is in the track, , the application may select the *FreeTrack* function to eject the document from the track.

The behaviour of the application will be different based on the document type being processed. Using the *GetDocumentType()* function the app is always aware of the type of document being processed.

The progress of the document in the track is followed by the messages received by the application as described below:

WMPAR_SORTER_NEW_DOCUMENT is sent when the document enters the track, the LPARAM of this message reports the document number.

During this message it is possible to select the function *GetDocumentType()* to determine the document type (cheque or rigid).

WMPAR_SORTER_MICR_AVAILABLE is sent when the MICR Codeline is available to the application; to obtain the MICR codeline select the *GetMicrCodeline* function. The LPARAM of this message reports the document number.

WMPAR_SNIPPET_READY is sent when the snippets are available to the application. This message will be received twice, the first time for the front snippets, and the second time for the rear snippets. LPARAM returns a structure containing the requested snippets (see the description of Image and Snippet Structure).

WMPAR_SORTER_SET_ITEM_OUTPUT is sent when the document is ready to be pocketed. This message can be sent after the MICR codeline and/or the front snippets (if decision is required) and/or the rear snippets (if decision is requested).

Select the *SetPocket* function to set the destination pocket. Available options for Cheques:

- 1 for the exit pocket
- 2 for the exception pocket

The LPARAM of this message reports the document number. For rigid documents the application must always use the exit pocket.

WMPAR_SORTER_DOCUMENT_INSIDE_POCKET is sent when the document has been pocketed. The LPARAM of this message reports the document number.

WMPAR_IMAGE_READY is sent when the images from the scanners are available to the application. LPARAM returns a structure containing the requested images (see the description of Image and Snippet Structure).

WMPAR_DOC_COMPLETED is sent when all the processing phases have been completed. LPARAM is the document number.

The sequence of the message can vary as follows.

When a snippet is requested with ENABLE only, the sequence of the document message is:

- NEW_DOC;
- MICR;
- SET_ITEM_OUTPUT;
- IN_POCKET;
- SNIPPET_READY (FRONT);

- **SNIPPET_READY (REAR);**
- IMAGE_READY;
- DOC_COMPLETE.

The position of the SNIPPET_READY (FRONT) message is unpredictable. It depends on the system where the software is running. It could be sent before or after the IN_POCKET message.

Using the ENABLE_FOR_DECISION flag for the front snippet the messages arrive in the following order:

- NEW_DOC;
- MICR;
- IN_POCKET;
- **SNIPPET_READY (FRONT);**
- **SET_ITEM_OUTPUT;**
- IMAGE_READY;
- DOC_COMPLETE.

In this case the SNIPPET_READY (FRONT) message is sent before the SET_ITEM_OUTPUT message.

Using the ENABLE_FOR_DECISION for the front and rear snippets the sequence will be:

- NEW_DOC;
- MICR;
- **SNIPPET_READY (FRONT);**
- IN_POCKET;
- **SNIPPET_READY (REAR);**
- **SET_ITEM_OUTPUT;**
- IMAGE_READY;
- DOC_COMPLETE.

The SNIPPET_READY (REAR) message is sent before the SET_ITEM_OUTPUT, in order to use the rear image for the pocketing decision.

DeviceParameters

The structure DeviceParameters has remained unchanged.

The application could use two different approaches to set parameters:

- Use *SetDeviceParameters()*
This function sets the same parameters values for the entire set of document types.
- Use *SetDocProcessingParam()*
This function is able to set a different parameter value for each document.
To speed up the development the developer can call the *SetDeviceParameter()* to create common settings among the two documents types, then call the *SetDocProcessingParam()* to specialize the processing based on the document type.

The same OneDoc flag, Feeding mode and Printer settings are used for all document types. When an application sets one of these for a certain document type, the settings are copied to the other document type settings.

If the printer is enabled, the application must call *SendPrinterData()* for each type of document and can bypass printing by sending an empty string.

SetMaxPocket

In order to use the 2nd pocket (exception pocket), the two pocket option must be enabled on the IDCard.

To enable use of the exception pocket, use the *SetMaxPocket()* function to change the maximum number of pockets from 1 to 2:

- If a VisionNext is a 2 pocket device, the default settings for MaxPocket is 1. (NOTE: for a 2 pocket VisionX scanner the default setting is 2.)

Refer to the Barebones application source code for an example.

SetPocket

This function must be called during the *WMPAR_SORTER_SET_ITEM_OUTPUT* message.

SetPocket is the last API function to be called during that message. This function must be called in order to set the destination pocket for the processed document.

There are two possible options:

- Pocket #1
the doc is sent in the exit pocket.
- Pocket #2
the doc is returned in the exception pocket.
(Cheques only).

When a document is sent to the exception pocket the device immediately stops and returns a “Full pocket” exception to the application. In order to restart the device, the operator must remove the document from the exception pocket before scanning can resume.

Sorter errors

The return values for sorter errors codes have been expanded for the Vision next (please find them listed at the bottom of VApiInterface.h include file).

The error word (4 bytes long) is composed of the following fields:

Byte 3 (MSB) - Error Class/Type
 Byte 2 - Error Reason
 Byte 1 - Error happened in Pass # / Module #
 Byte 0 (LSB) - Sensor Index (Photocell #) / Other info

Error Class/Type

ERROR LABEL	ERROR CODE	DESCRIPTION
ERR_CLASS_GENERAL	1	General Purpose error
ERR_CLASS_COMUNICATION	2	Communication Error (Computer <-> Device)
ERR_CLASS_INITIALIZATION	3	Hardware Initialization Error
ERR_CLASS_PERIPHERAL	4	Error Generated by a Peripheral
ERR_CLASS_TRANSPORT	5	Paper Transport Error

Error Reasons for ERR_CLASS_COMUNICATION

ERROR LABEL	ERROR CODE	DESCRIPTION
ERR_DUE_TO_CHECKSUM	1	Checksum verification failed
ERR_DUE_TO_UNKNOWN_CMD	2	Unknown Command
ERR_DUE_TO_INVALID_CMD	3	Invalid Command
ERR_DUE_TO_DOC_ID	4	Document ID already exists
ERR_DUE_TO_POCKET	5	Invalid Pocket ID
ERR_DUE_TO_SOURCE	6	Invalid Document Source
ERR_DUE_TO_PREV_POCKET	7	Previous Document Pocket not set
ERR_DUE_TO_PARAM	8	Error while reading/writing parameter

Error Reasons for ERR_CLASS_INITIALIZATION

ERROR LABEL	ERROR CODE	DESCRIPTION
ERR_DUE_TO_INIT_PHOTOS	1	Photocells calibration failed

Error Reasons for ERR_CLASS_PERIPHERAL

ERROR LABEL	ERROR CODE	DESCRIPTION
ERR_DUE_TO_OPENED_COVER	1	The top cover has been opened during feeding (AGP-14)
ERR_DUE_TO_CLOSED_COVER	2	RESERVED FOR FUTURE USE
ERR_DUE_TO_BATTERY_LOW	3	RESERVED FOR FUTURE USE
ERR_DUE_TO_PRINTER_LIFT	4	Mobile AGP printer lift error

Error Reasons for ERR_CLASS_TRANSPORT

ERROR LABEL	ERROR CODE	DESCRIPTION
ERR_DUE_TO_DOC_LOST	1	Document never reached the next photo sensor
ERR_DUE_TO_DOC_TOO_LONG	2	The document is too long
ERR_DUE_TO_DOC_TOO_SHORT	3	The document is too short
ERR_DUE_TO_DOC_FEED	4	Feed failure
ERR_DUE_TO_FREE_TRACK	5	Free track unsuccessful
ERR_DUE_TO_DOC_DEST	6	Document's pocket is not assigned
ERR_DUE_TO_FULL_POCKET	7	Full pocket detected
ERR_DUE_TO_OPEN_POCKET	8	Problem opening the pocket
ERR_DUE_TO_DOUBLE_FEEDING	9	Double-feeding detected
ERR_DUE_TO_DEFAULT_POCKET	10	Document is in the default pocket
ERR_DUE_TO_BUSY_PHOTO	11	Track photos are busy (there is something inside the track)
ERR_DUE_TO_PRINTER	12	Printer error detected
ERR_DUE_TO_IMAGE	13	Image error detected
ERR_DUE_TO_BUFFER_BUSY	14	Internal FW error (buffer overwriting)
ERR_DUE_TO_FRANKING	15	RESERVED FOR FUTURE USE
ERR_DUE_TO_FEED_FAILURE	16	Feeding procedure failed: either the alignment sensor is not covered correctly, or the document has been removed
ERR_DUE_TO_TRACK	17	When a document is detected by the feed, one or more of the other sensors are busy
ERR_DUE_TO_SCANNER_FLIP	18	RESERVED FOR FUTURE USE
ERR_DUE_TO_MULTIPLE_FEED	19	A document feed was detected while another document was already being processed
ERR_DUE_TO_MOTOR_STUCK	20	RESERVED FOR FUTURE USE
ERR_DUE_TO_HOST_DISCON	21	Internal FW error
ERR_DUE_TO_FULL_FEEDER	22	The feeder is full-up of documents and cannot start.
ERR_DUE_TO_PREFEED_FAILURE	23	Document is in a wrong (unrecoverable) position during the pre-feeding phase

Error Reasons for ERR_DUE_TO_PRINTER

ERROR LABEL	ERROR CODE	DESCRIPTION
PRT_ERR_LENGTH	1	The selected horizontal position is not compatible with actual document length and current text/bitmap size.
PRT_ERR_LENGTH_SMART	2	Same as PRT_ERR_LENGTH error but happening when SmartJet is enabled.

Error Reasons for ERR_DUE_TO_PRINTER_LIFT

ERROR LABEL	ERR	DESCRIPTION
LIFT_ERR_END_OF_STROKE_STUCK_CLOSED	1	The lift switch cannot be released during the zeroing procedure
LIFT_ERR_END_OF_STROKE_NOT_FOUND	2	The lift switch cannot be found during the zeroing procedure
LIFT_ERR_END_OF_STROKE_FOUND	3	The lift switch is unexpectedly closed
LIFT_ERR_UNKNOWN_COMMAND	4	The lift command is unknown or unmanaged
LIFT_ERR_COMMAND_TIMEOUT	5	A new lift command is requested before that the previous command is completed, and a timeout expires
LIFT_ERR_CART_MAINT_ERR_ZERO	6	Zeroing request cannot be executed during the cleaning procedure
LIFT_ERR_CART_MAINT_ERR_CLEAN	7	Lift cannot be moved during the cleaning procedure
LIFT_ERR_CART_MAINT_ERR_CAP	8	Capping cannot be executed
LIFT_ERR_CART_MAINT_ERR_PURGE	9	Lift cannot be moved during the purge procedure
LIFT_ERR_CART_MAINT_ERR_TRANSPORT	10	Track is not free during the execution of the maintenance procedure
LIFT_ERR_CART_MAINT_ERR_OVERLOAD	11	A new maintenance procedure is requested before the previous is one completed
LIFT_ERR_CART_MAINT_ERR_LIFT	12	Lift is in error during the execution of the maintenance procedure
LIFT_ERR_CART_MAINT_ERR_COVER	13	Cover is open during the execution of the maintenance procedure

Pass index

This field indicates in which pass the error occurred.

PASS LABEL	PASS CODE
First	1
Second	2
Third	3

Sensor index

This field indicates on which sensor the error occurred.

SENSOR LABEL	SENSOR CODE
Alignment system	1
Pressure Plate	2
Cheques' Feeder	3
Track #1 (pre-Feeder)	4
Track #2 (MICR-DFD)	5
Track #3 (Printer)	6
Track #4 (CIS)	7
Rigids' (Card) Feeder	8
Exception pocket	9

Image resolutions

The Vision neXt supports image acquisition at 100, 200, 300 and 600 DPI.

The 600 DPI is available for RIGID (ID cards) documents only and it is permitted with below image formats. See the following tables for details.

Image DPI	Cheques	Rigids
100	X	X
200	X	X
300	X	X
600		X

Image Formats	Available up to 300 DPI	Available @300 DPI (cards only)	Available @600 DPI (cards only)
FORMAT_GIV	X	X	X
FORMAT_JPEG	X	X	X
FORMAT_BMP	X	X	X
FORMAT_NATIVE_BMP	X	X	X
FORMAT_JPEG_COLOR	X	X	X
FORMAT_BMP_COLOR	X	X	X
FORMAT_GIV_DROP_OUT_RED	X	X	X
FORMAT_GIV_DROP_OUT_GREEN	X	X	X
FORMAT_GIV_DROP_OUT_BLUE	X	X	X
FORMAT_BMP_TRUE_COLOR		X	X
FORMAT_JPEG_TRUE_COLOR		X	X
FORMAT_GIV_DOR_TRUE_COLOR		X	X
FORMAT_GIV_DOG_TRUE_COLOR		X	X
FORMAT_GIV_DOB_TRUE_COLOR		X	X
FORMAT_GIV_DROP_OUT_IR			
FORMAT_JPEG_UV			
FORMAT_JPEG_UV_NEGATIVE			
FORMAT_PNG	X	X	X
FORMAT_PNG_COLOR	X	X	X
FORMAT_PNG_TRUE_COLOR		X	X

GetFullPocketStatus (VN only)

BOOL *GetFullPocketStatus*(DWORD DeviceID, DWORD *pStatus)

This function returns the status of the Full Pocket(s) sensor.

This function is supported for the Vision NeXt only.

The parameter pStatus must be a pointer to a DWORD.

If successful (i.e. the function returns TRUE), the value of pStatus DWORD will contain a value which gives the status of the two pockets:

- 0 : Pockets NOT full
- 1 : Pocket 1 full (destination pocket)
- 2 : Pocket 2 full (exception pocket)

Following defines can be used to easily query the return value:

```
#define STATUS_POCKET_NO_FULL      0
#define STATUS_POCKET_DEST_FULL    1
#define STATUS_POCKET_EXCP_FULL    2
```

Valid in State : ***DeviceOnline, DeviceFeeding***

State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

DWORD ***pStatus** – it's a pointer to the result DWORD.

Return Value : TRUE if the function terminates correctly.

FALSE if an error occurs, call ***GetApiError*** or ***GetApiErrorString*** to get more information about it.

Example (refer also to the Barebones application source code – VNBarebonesDlg.cpp)

```
DWORD DocID;
...
case WMPAR_SORTER_EXCEPTION:
    DWORD ErrorCode = (DWORD)(LParam & 0x0000FFFF);
    DocID = ((DWORD)LParam & 0xFFFF0000)>>16;
    ...
    if( ErrorCode == DEVICE_ERR_FULL_POCKET )
    {
        DWORD pocketStatus = 0;
        if ( GetFullPocketStatus(m_DeviceID, &pocketStatus) )
        {
            if ( pocketStatus == STATUS_POCKET_EXCP_FULL )
            {
                // Application has to manage the exception full pocket status
            }
            ...
        }
        else
        {
            // Report error
            GetApiErrorString( ApiErrorString, 200 );
            MessageBox(...);
        }
    }
    ...
    break;
```


IsFeederEmpty (VN only)

BOOL *IsFeederEmpty*(DWORD DeviceID, DWORD FeederID, BOOL *pFlag)

This function returns to the application the status of the feeder sensor (cheque or rigid card based on selection) to detect the presence of documents in the feeder. If the *pFlag* return value is TRUE, the feeder is empty. If FALSE the feeder contains one or more documents. This function is supported for the Vision NeXt only.

Valid in State : *DeviceOnLine*
State Transition : None

Arguments : DWORD **DeviceID** – the Identification number of the device.

 DWORD **FeederID** – the selected feeder. Possible values are:

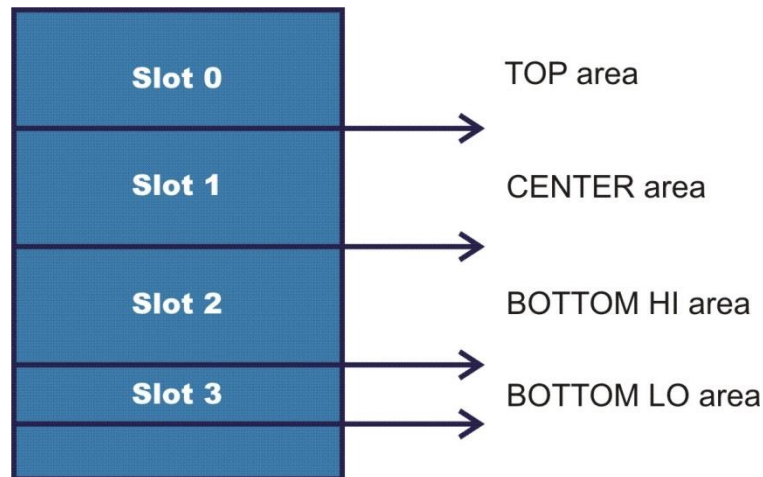
FEEDER_CHEQUE	(0)
FEEDER_RIGID	(1)

 BOOL ***pFlag** – this parameter receive the Feeder status result.

Return Value : FALSE if an error occurs, call *GetApiError* or *GetApiErrorString* to get more information about it.

Mobile AGP-14 printer

The mobile AGP printer prints up to 14 lines of text at 200 DPI. The printer head is positioned to use four areas (slots) for printing with a lifting mechanism (lift).



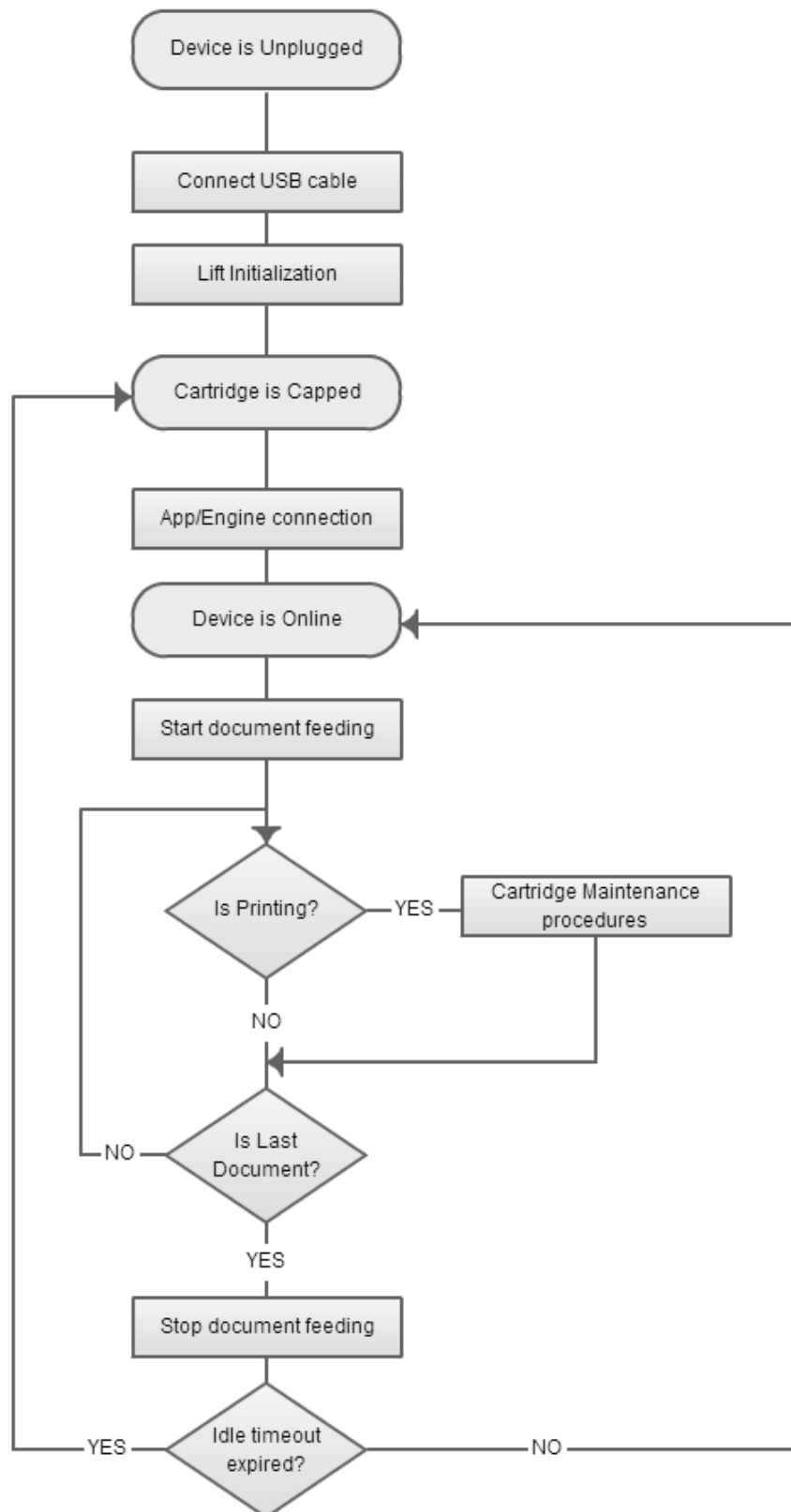
Slots 0, 1 and 2 are 100px high, while slot 3 is only 50px high: 100px conventionally contains 4 lines of plain text at 200 DPI. Each slot can be enabled or disabled for printing: for example, if the slot 1 is disabled, that slot is skipped during printing and the cartridge goes directly from slot 2 to 0.

If slot 2 and 3 are enabled, slot 3 height is limited to 50px. If the slot 2 is disabled slot 3 height is no longer limited, and all 100px can be printed.

Workflow

The following diagram describes the Mobile AGP-14 workflow, starting with the device powered off through the document processing operations.

The Cartridge Maintenance procedures include: purging, cleaning and capping operations that keep the AGP cartridge in the optimal condition to achieve maximum quality during the printing of documents. The maintenance procedures shall be executed both during the printing of documents and during normal document scan process (see next chapter “AGP-14 Inkjet Cartridge Maintenance Best Practices” for a description on how to interface and perform the maintenance operations by third parties applications).



AGP-14 API Interface

The AGP-14 uses the *SetPrinterParam()* function instead of the *SendPrinterData()* function. Refer to *VApiInterface.h* in the Software Developer Kit

```
BOOL SetPrinterParam( DWORD DeviceID, DWORD GroupID, DWORD ParamID,  
                     void* pData, DWORD DataLen );
```

(GroupID) refers to the different printer area settings:

```
// Printing Group Settings  
#define VAPI_MLP_GROUP_1    (0) // TOP area (4 lines)  
#define VAPI_MLP_GROUP_2    (1) // CENTER area (4 lines)  
#define VAPI_MLP_GROUP_3    (2) // BOTTOM HI area (4 lines)  
#define VAPI_MLP_GROUP_4    (3) // BOTTOM LO area (2 lines)  
#define VAPI_MLP_GROUP_ALL  (4) // ALL AREAS
```

(ParamID) The printer parameters:

```
// Parameters  
#define VAPI_MLP_PARM_ENABLE      (0) // Area Enabling (BOOL)  
#define VAPI_MLP_PARM_TRAILING_EDGE (1) // Trailing Edge position (BOOL)  
#define VAPI_MLP_PARM_FONT        (2) // Font setting (LOGFONT)  
#define VAPI_MLP_PARM_TEXT        (3) // Text to print (char*)  
#define VAPI_MLP_PARM_TEXT_POSITION (4) // Text horizontal offset from  
                                         edge (mm) (UINT32)  
  
#define VAPI_MLP_PARM_BMP_TYPE     (5) // Define Bitmap Image type  
                                         (PRT_SRC_FILE/PRT_SRC_MEM_PTR)  
#define VAPI_MLP_PARM_BMP          (6) // Path to the Bitmap Image or  
                                         pointer to DIB in memory  
                                         (void*)  
  
#define VAPI_MLP_PARM_BMP_POSITION (7) // Image horizontal offset from  
                                         edge (mm) (UINT32)  
#define VAPI_MLP_PARM_QUALITY      (8) // Printer Quality flag (UINT32)  
#define VAPI_MLP_PARM_DONE         (9) // All the parameters are set so  
                                         the printer buffer can be  
                                         formatted
```

To set the printer parameters for each area, one option is to use a structure called *PrinterParameters*:

```
typedef struct _PrinterParams  
{  
    BOOL    bEnableArea;  
    BOOL    bTrailingEdgeArea;  
    LOGFONT PrinterFontArea;  
    char    sTextArea[2000];  
    DWORD   nTextPosArea;  
    char    sImgPathArea[_MAX_PATH];  
    DWORD   nBmpPosArea;  
    DWORD   nQualityArea;  
}PrinterParameters;  
  
PrinterParameters MobileAgpParameters[4];
```

Valid in State: Smart Jet disabled

When the Smart Jet is disabled, the printer is an up-stream device. This means that the printer information have to be defined before the document feeding. Thus, the call before **StartFeeding** defines the printer data for the first document.

This function has to be called in DeviceOnline state, before the StartFeeding call, for the first document. For the next documents, in DeviceFeeding state, has to be called during WMPAR_SORTER_NEW_DOCUMENT.

The following calls, during WMPAR_SORTER_NEW_DOCUMENT message, define the printer data for the next document.

Example of printer sequence:

1. SetPrinterParam(...) for the 1st doc
2. StartFeeding(...)
3. During NEW_DOC message of the 1st doc call SetPrinterParam(...) for the 2nd doc
4. During NEW_DOC message of the nth doc call SetPrinterParam(...) for the n+1th doc

Valid in State: Smart Jet enabled

This function has to be called during the WMPAR_SORTER_SET_ITEM_OUTPUT message for all the machines.

When Smart Jet is enabled, the printer is a downstream device. This means that the printer information can be defined after MICR and/or OCR information are available.

Example of printer sequence:

1. Call StartFeeding(...)
2. During SET_ITEM_OUTPUT message of the 1st doc call SetPrinterParam(...) for the 1st doc
3. During SET_ITEM_OUTPUT message of the nth doc call SetPrinterParam(...) for the nth doc

Using the above defined PrinterParams structure a loop can be used to set all params for all areas. Please note that it is ***not*** required to set all areas/values just those used, because the VNEngine internal PrintingData object is created with default values (i.e. area disabled, offsets 0, text and image path NULL, quality default, ...).

```
for ( int i = 0; i < VAPI_MLP_AREA_MAX; i++ )
{
    BOOL bEnable = MobileAgpParameters[i].bEnableArea;
    SetPrinterParam(m_DeviceID, i, VAPI_MLP_PARM_ENABLE, &bEnable,
        (UINT32) sizeof(bEnable));

    BOOL bTrailingEdge = MobileAgpParameters[i].bTrailingEdgeArea;
    SetPrinterParam(m_DeviceID, i, VAPI_MLP_PARM_TRAILING_EDGE,
        &bTrailingEdge, (UINT32) sizeof(bTrailingEdge));

    DWORD Quality = MobileAgpParameters[i].nQualityArea;
```

Panini Vision API 4.5.0 Reference Manual Revision 1

```
SetPrinterParam(m_DeviceID, i, VAPI_MLP_PARM_QUALITY, &Quality,
                (UINT32) sizeof(Quality));

SetPrinterParam(m_DeviceID, i, VAPI_MLP_PARM_FONT,
                (void*)&MobileAgpParameters[i].PrinterFontArea,
                (UINT32) sizeof(LOGFONT));

DWORD dwImgSrcType = 0; //PRT_SRC_FILE
SetPrinterParam(m_DeviceID, i, VAPI_MLP_PARM_BMP_TYPE,
                &dwImgSrcType, (UINT32) sizeof(dwImgSrcType));

char *sText = MobileAgpParameters[i].sTextArea;
SetPrinterParam(m_DeviceID, i, VAPI_MLP_PARM_TEXT, sText,
                (UINT32) strlen(sText));

DWORD dwTextOffset = MobileAgpParameters[i].nTextPosArea;
SetPrinterParam(m_DeviceID, i, VAPI_MLP_PARM_TEXT_POSITION,
                &dwTextOffset, (UINT32) sizeof(dwTextOffset));

char *sImgPath = MobileAgpParameters[i].sImgPathArea;
SetPrinterParam(m_DeviceID, i, VAPI_MLP_PARM_BMP, sImgPath,
                (UINT32) strlen(sImgPath, 0));

DWORD dwImgOffset = MobileAgpParameters[i].nBmpPosArea;
SetPrinterParam(m_DeviceID, i, VAPI_MLP_PARM_BMP_POSITION,
                &dwImgOffset, (UINT32) sizeof(dwImgOffset));
}
```

After setting all needed parameters, the following call can be executed to trigger printing:

```
SetPrinterParam(m_DeviceID, VAPI_MLP_GROUP_ALL, VAPI_MLP_PARM_DONE, 0, 0);
```

NOTE: the GroupID value is irrelevant when setting the VAPI_MLP_PARM_DONE, hence it could be either one of the above Printing Group Settings values or the global VAPI_MLP_GROUP_ALL value as in the code sample.

To determine if the scanner has an AGP-14 printer installed, a specific device feature can be read (value 0 or 1) using the `GetDeviceFeature` function with ParamID value set to `DEVICE_FEATURE_PRINTER_MULTI_AREA`.

AGP-4 Compatibility mode

The AGP-14 mobile printer can be used as an AGP-4 printer when operated with the *SendPrinterData()* function call (see *SendPrinterData* chapter on page 118). This way the application can print up to four lines or a 100px height bitmap in the CENTER area (slot 2 position) without any code changes.

AGP-14 Inkjet Cartridge Maintenance Best Practices

Vision NeXt AGP 14 (VN14) devices are equipped with a cleaning module to maintain the best possible printing quality. Cleaning the cartridge avoids excessive ink buildup. Capping the cartridge prevents the cartridge nozzles from drying out. Vision neXt scanners equipped with single line or AGP-4 printers do not have cleaning modules and are excluded from these maintenance procedures.

The application manages the cartridge maintenance using parameters to govern the maintenance process. API function calls are used to invoke maintenance procedures.

The cartridge is always uncapped and cleaned when the scanner is connected (started up). During the cleaning procedure 100 droplets of ink are sprayed, the cartridge nozzles are passed over the wiper blade twice, and another 1,000 droplets of ink are sprayed.

The cartridge is always capped when the scanner is disconnected (shutdown). The inkjet cartridge is positioned in the cleaning module so that the nozzles are sealed to prevent them from drying out.

These procedures are automatically performed by the Vision neXt Engine.

The VisionAPI exports the Cap & Cleaning functionality to the user application at three different levels:

1. API Windows Messages are sent to the application to notify that a certain threshold is expired and a proper maintenance operation should be performed (no automatic maintenance operations are performed directly by the engine apart from the initial Purging at connect time and the final Cleaning & Capping at disconnect time).
2. VisionAPI Functions are exported to perform the maintenance procedure.
3. API Parameters are available to modify the basic thresholds and customize the default Vision NeXt Engine management.

Windows Message

The message coded as `WMPAR_CARTRIDGE_MAINTENANCE_REQUEST` is dispatched to the user application with two parameters:

- the identifier of the device that requested the maintenance operation
- the identifier of the requested operation coded as follow:
`CLEANING = 1`, `CAPPING = 2` and `PURGING = 3`

This message will be sent by the VisionAPI to the user application only once for each threshold expiration cycle. This message is asynchronous and does not wait for any acknowledgement from the user. The internal threshold counter is internally reset and a new cycle can start again.

In this way it's guaranteed the backward compatibility with older application that didn't handle the Cap & Cleaning of the cartridge; furthermore the application can deliberately skip the message to implement a completely custom policy using the available Cap & Cleaning commands.

When the application receives the message, it remembers the notification and perform the maintenance operation as soon as possible, with respect to the current status of the device, i.e. if the device is in the Feeding status no maintenance can be done: the application must stop the feeding, putting the device in the OnLine status and then perform the maintenance. After

the operation is completed is application responsibility to put back the device in Feeding state (if this is the case).

VisionAPI Functions

The function *ServicePrinterHead()* allows to perform a cartridge maintenance procedure. This function is available only in OnLine and Change Parameters states and the prototype is defined as follows:

```
BOOL ServicePrinterHead( DWORD DeviceID, DWORD ServiceID )
```

where the ServiceID parameter is coded as

- CLEANING = 1
- CAPPING = 2
- PURGING = 3

This function is synchronous, so the calling thread is blocked until the maintenance of the cartridge is completed. If the operation requested doesn't complete successfully, the function returns FALSE and a proper exception is raised.

The function *GetDeviceStatus()* allows to get current status of "readiness" of the device. The prototype of the function is defined as follows:

```
BOOL GetDeviceStatus( DWORD DeviceID, BYTE* pDeviceStatus )
```

where the pDeviceStatus parameter can return the values:

- DEVICE_STATUS_READY (0)
- DEVICE_STATUS_NOT_READY (1)
- DEVICE_STATUS_COVER_OPEN (2)
- DEVICE_STATUS_MAINTENANCE (3)
- DEVICE_STATUS_FULL_POCKET (4)

This function can be called to verify if the device is performing a maintenance procedure (return value 3) in multi-threaded applications.

Error Handling

The application is responsible for all error handling including errors encountered as a result of calling functions that cannot be started or completed as a result of the cartridge nest moving (i.e. Device Error 55 - DEVICE_ERR_LIFT).

Recovering from errors associated with the Vision NeXt normally requires the calling of *FreeTrack* to clear the error.

Cap & Cleaning Parameters

A set of new Extended Parameters allows the user to customize some of the threshold of the basic Cap & Cleaning management.

Parameter	Description	Default
EP_CC_CAPPING_IDLE_TIMEOUT	The number of seconds elapsed since the scanner has been idle before the cartridge should be capped.	300 sec
EP_CC_PURGE_DOC_THRES	The number of documents during scanning before the cartridge should be purged.	100 docs
EP_CC_CLEAN_DOC_THRES	The number of documents during scanning before the cartridge should be cleaned.	200 docs
EP_CC_DISCONNECT_TIMEOUT	The number of seconds after the scanner has been disconnected before the cartridge is capped.	0 sec (synch capping)

These new extended params can be read and set trough the usual *GetExtendedParameter* and *SetExtendedParameter* functions. Parameters are defined inside VApiExtendedParameters.h.

Firmware parameters

These parameters are sent to the device with the function call *SetSorterParameter*, and are retrieved from it with *GetSorterParameter*. When the device connects the default values of the sorter parameters are restored.

The following FW parameters are available:

- ID = 0 Double-Feeding Detection Enabling (0 – 1, default is 0).
Enable the Double-Feeding Detection.
- ID = 1 RESERVED FOR FUTURE USE.
- ID = 2 RESERVED FOR FUTURE USE
- ID = 3 RESERVED FOR FUTURE USE
- ID = 4 Hole Filter Length (5 mm – 20 mm, default is 5 mm).
To avoid holes in the paper indicating document end.
- ID = 5 RESERVED FOR FUTURE USE
- ID = 6 RESERVED FOR FUTURE USE
- ID = 7 Min document length (100 – 450 mm, default is 105).
This value is the minimum document length accepted.
- ID = 8 Max document length (100 – 450 mm, default is 240).
This value is the maximum document length accepted.
- ID = 9 Enable Full-Pocket detection (0-1, default is 1)
Enables the Full Pocket detection. When a Full Pocket condition is detected, the

machine stops and the application receives a `DEVICE_ERR_FULL_POCKET` exception.

- ID = 10 RESERVED FOR FUTURE USE
- ID = 11 RESERVED FOR FUTURE USE
- ID = 12 Enable feeder extra drive (0 – 1, default 0)

When a document which has holes inside has difficulties in being feeded, this parameters enables an “extra push” length to allow a proper feed without generating feed failure errors. This feature will introduce a delay during the check of the empty track condition.

Appendix H: API StartUp mechanism

The VisionAPI *StartUp()* function call is used to open a communication channel between the device and the application: if the StartUp is successful the Device Identification Number is returned.

However, the actual device started depends on what version of VisionAPI is used. Since different logic has been implemented to support the single/multi engine capability, a different sequence of function calls is required to correctly perform the StartUp operation.

The following table summarizes the different scenarios:

VisionAPI version	Engine type	Function call required	Will start up...
MVX API 1.x MVX API 2.x	MyVisionX only	1. StartUp(...)	1. The first available MVX/VX
VAPI 3.0 - 3.5	Multi-engine	1. StartUp(...) 2. VApiSetDeviceEngine(engineType) StartUp(...)	1. The first available MVX/VX (default) – Backward compatibility 2. The first available device (of selected engine type)
VAPI 3.6.x - 4.1.x	Multi-engine	1. StartUp(...) 2. VApiSetDeviceEngine(engineType) StartUp(...) 3. VApiGetDeviceList(...) VApiSelectDevice(serialNumber, deviceType) StartUp(...)	1. The first available MVX/VX (default) – Backward compatibility 2. The first available device (of selected engine type) – Backward compatibility 3. The selected device
VAPI 4.2.0+	Multi-engine	1. StartUp(...) 2. VApiSetDeviceEngine(engineType) StartUp(...) 3. VApiGetDeviceList(...) VApiSelectDevice(serialNumber, deviceType) StartUp(...)	1. a "dynamic selected" device (see below) - Backward compatibility 2. The first available device (of selected engine type) – Backward compatibility 3. The selected device

In VAPI 4.2.0 and above a new "dynamic default" device selection has been introduced. This avoids the limitation of starting up the first available VX device when the *StartUp()* is called without setting an Engine type or a SerialNumber first (i.e. NO *VApiSetDeviceEngine()* or *VApiSelectDevice()* called). The new logic follows this sequence:

1. Fetch the list of connected devices (if any) and start the first available device*
2. Otherwise the first available VX is started up as in legacy code

* The *VApiGetDeviceList()* function (used to find the first connected device) is creating the list using a strict query ordering on the Engines loaded (i.e. VX -> ID -> WI -> VXA4 -> VN). Hence the first available device started up (which is device[0] of the previous list) depends on the engines loaded as well as the devices present at connection time. Being that the VN the last Engine polled, it may happen that a VisionNext is never started up if other devices are connected.

Appendix I: API Logging mechanism

DebugView logging

The VisionAPI sends out log messages on the Win32 system debugger (using Win32 OutputDebugString API function) but only if the DebugView application (dbgview.exe) is running. Otherwise, if DebugView is not active, no log messages are generated.

DebugView monitor application can be downloaded from Windows Sysinternal web site (<https://technet.microsoft.com/en-us/sysinternals>)

From an application standpoint, the VisionAPI log messages can be enabled and captured by implementing a “dbgviewClass” window which listens to the OutputDebugString API calls using specific Win32 kernel objects.

A sample project (WinDebug) is distributed with the VisionAPI DevKit and contains:

- Monitor (command line example)
- WinMonitor (Win32 app example)

They both create a window with class “dbgviewClass” which enables the OutputDebugString from our API code and listens for those messages, printing out on prompt shell (command line example) or inside a ListBox (Win32 app example).

File logging

In VAPI 4.x a new file logging mechanism has been introduced to allow collecting logs in a transparent way without having to use the DebugView application.

In the \ProgramData\Panini folder, create a LOGF.DATA file (empty). The presence of this file is checked when the VisionAPI is started up and, if available, the log messages are written on a VisionAPI.LOG file in the same folder instead of usual DebugView trace. The file is opened in append mode so the user will need to delete it to have logging start from scratch.

New behaviour: In VAPI 4.4.x the file logging has been updated. In the LOGF.DATA file parameters can be added (with a .ini style format) to define the logging process.

The parameters structure is as follows (NOTE: the presence of the [LOG] section is mandatory):

```
[LOG]
maxsize=10240000
maxfiles=2
startTime=0900
endTime=1700
```

maxsize: maximum size in bytes of the VisionAPI.LOG file. After reaching the max size, the file will be re-opened in write mode (erasing all previous content)

maxfiles: maximum number of "rotation log" files, i.e. after the VisionAPI.LOG file reaches its max size, if the maxfiles number is >0, the current log file will be renamed appending -1, -2, ... etc. up to the maxfiles value, before creating a new VisionAPI.LOG file. In case the max number of files is reached, the oldest log file will be overwritten.

startTime, endTime: if both are specified (using the hhmm 24-hours format) the log file will be written if the current PC time is within the time range.

OEM logging

In VAPI 4.4.2 a new OEM logging support has been introduced to allow final user application to gather VisionAPI log messages and combine with their own with the aim of troubleshooting problems in an easier way by looking at one unique log trace.

A new API call has been added to the VisionAPI public interface and shall be used to register a callback function pointer that will be used during VisionAPI operations to route trace messages (passing back a char pointer) to the client application.

Here is the definition (available in VapiInterface.h):

```
typedef void (CALLBACK *LOGCALLBACKPTR)(char *); // OEM logging callback  
type  
VISION_API_DECL BOOL VISION_API VApiSetLogCallback( LOGCALLBACKPTR  
logCallback );
```

This is the function prototype that can be used on a client application:

```
static void CALLBACK LogCallBack(char *szMessage);
```

registered with the VAPI call:

```
VApiSetLogCallback( (LOGCALLBACKPTR)&LogCallBack );
```

NOTE: In order to get a full log (pretty similar to the DebugView one), the VapiSetLogCallback() shall be called as the very first VisionAPI call just after loading the DLL, even before the VApiLoad(), VapiGetDeviceList(), VApiSelectDevice() or StartUp(), otherwise important information like the API/Engine(s) versions or the engine(s) starting up debug messages will be lost.

NOTE: To avoid potential crashes during application exit, the callback pointer shall be reset by client app *before* unloading the VisionAPI DLL, by calling VApiSetLogCallback(NULL). After executing such function, VAPI log messages will no longer routed to the client app, allowing to safely free allocated resources.

Sample code is available in Barebones project sources (look for VapiSetLogCallback in BarebonesDlg.cpp file).

Appendix J: PDF document support

Vision APIs 4.5+ provides the possibility to encapsulate cheque images and MICR string into a PDF document.

The Vision PDF document creation APIs supplies 2 types of function:

- Functions to set the format of the PDF output
- Function (only one) to set the images and the MICR and to get back the PDF file as a byte stream.

Applications can avoid calling the functions to set the format of the PDF output as long as they like the default values (see below), so that just one call to PdfFunGetPdf is able to obtain the PDF output file (as a byte stream).

Functions to set the format of the PDF output

```
VISION_API_DECL void VISION_API PdfFunSetPageSize(int page_size);
```

Set the page dimensions of the PDF document. The page_size argument must be set to one of the following values:

- PANINIPDF_PAGE_SIZE_LETTER
- PANINIPDF_PAGE_SIZE_LEGAL
- PANINIPDF_PAGE_SIZE_A3
- PANINIPDF_PAGE_SIZE_A4
- PANINIPDF_PAGE_SIZE_A5
- PANINIPDF_PAGE_SIZE_B4
- PANINIPDF_PAGE_SIZE_B5
- PANINIPDF_PAGE_SIZE_EXECUTIVE
- PANINIPDF_PAGE_SIZE_US4x6
- PANINIPDF_PAGE_SIZE_US4x8
- PANINIPDF_PAGE_SIZE_US5x7
- PANINIPDF_PAGE_SIZE_COMM10

Default Page size is A4.

```
VISION_API_DECL void VISION_API PdfFunSetPageOrientation(int page_orientation);
```

Set the orientation of the page of the PDF document. The page_orientation argument must be set to one of the following values:

- PANINIPDF_PAGE_PORTRAIT
- PANINIPDF_PAGE_LANDSCAPE

Default orientation is PORTRAIT.

Panini Vision API 4.5.0 Reference Manual Revision 1

```
VISION_API_DECL void VISION_API PdfFunSetLeftMargin(int left_margin);
```

Set the left margin in millimeters (default 5mm).

```
VISION_API_DECL void VISION_API PdfFunSetRightMargin(int right_margin);
```

Set the right margin in millimeters (default 5mm).

```
VISION_API_DECL void VISION_API PdfFunSetTopMargin(int top_margin);
```

Set the top margin in millimeters (default 5mm).

```
VISION_API_DECL void VISION_API PdfFunSetBottomMargin(int bottom_margin);
```

Set the bottom margin in millimeters (default 5mm).

```
VISION_API_DECL void VISION_API PdfFunSetInterspace(int interspace);
```

Set the inter-space between the different items (front image, rear image and MICR) printed on the PDF. The inter-space argument must be set as millimeter (default 5mm).

```
VISION_API_DECL void VISION_API PdfFunSetCreatorString(char *creator, int str_dim);
```

Set the “creator” label embedded into the PDF document (default "Panini S.p.A.").

```
VISION_API_DECL void VISION_API PdfFunSetProducerString(char *producer, int str_dim);
```

Set the “producer” label embedded into the PDF document (default "Panini S.p.A.").

```
VISION_API_DECL void VISION_API PdfFunSetFontDim(double font_dim_mm);
```

Set the font dimension in millimeters (default 5mm).

Function to get the PDF output

```
VISION_API_DECL BOOL VISION_API PdfFunGetPdf(BYTE *front, int fdim, int ftype, BYTE *rear, int rdim, int rtype, char *micr, int mdim, BYTE **out_pdf, int *pdf_dim );
```

This function can be used to set which items have to be embedded into the PDF document and to get back the PDF file as a byte stream in memory (application has manage the memory buffer and to save it on a proper location on disk).

***front** is a pointer to a buffer containing the front image, if NULL the front image won't be embedded into the PDF output.

fdim is the dimension of the front image buffer in bytes.

ftype is the format of the front image and must be set to one of the following values:

- PANINIDPF_INPUT_TYPE_BMP
- PANINIDPF_INPUT_TYPE_JPG
- PANINIDPF_INPUT_TYPE_PNG

***rear** is a pointer to a buffer containing the rear image, if NULL the rear image won't be embedded into the PDF output.

rdim is the dimension of the rear image buffer in bytes.

rtype is the format of the rear image and must be set to one of the following values:

- PANINIDPF_INPUT_TYPE_BMP
- PANINIDPF_INPUT_TYPE_JPG
- PANINIDPF_INPUT_TYPE_PNG

***micr** is a pointer to a buffer containing the MICR string, if NULL the MICR string won't be embedded into the PDF output.

mdim is the dimension of the MICR string in bytes.

****out_pdf** is the output buffer allocated by the function that contains the PDF output

The buffer can be deleted with a standard C++ *delete* call, anyway the buffer is automatically deleted every time the *PdfFunGetPdf* is called and when the VisionApi is shutdown (so that the explicit deletion of the buffer can be in most cases avoided).

NOTE: the buffer is automatically deleted every time the function PdfFunGetPdf is called, if the user need to post process it in memory the buffer must copied before the PdfFunGetPdf is called again.

***pdf_dim** is the output integer that will contain the size of the out_pdf buffer.

Appendix J: Multiple Applications support

This section describes the requirements of the new feature of the VisionAPI that can be accessed from multiple applications at the same time. This feature applies to Vision X and Vision NeXt engines only.

Background

The enhancement comes from a customer request to be able to load and run the VisionAPI libraries and the underlying VX/VN engines/devices from two different applications at the same time to allow a specific use case in which cheque images and MICR codelines need to be acquired and managed by the two applications concurrently.

Because only one application is allowed to “own” our scanner at a time, in order to fulfill the previous scenario, the customer had to close one application before starting up the other one, spending a lot of time signing in/out between the two systems: This was creating a lot of delay for the customer and frustration for the teller operator.

Vision API changes

This feature implies the following modifications/enhancement to the VisionAPI:

- Allow multiple applications to load VisionAPI and the underlying engines from multiple applications at the same time.
 - Currently the VisionAPI can be loaded from multiple applications, but not the engines. This behavior is implemented using a named Mutex at engine library level.
 - When the engine is loaded by VisionAPI, if the Mutex already exists, the library refuses to load returning a `VAPI_ERR_INTERNAL (4)` error.
 - In order to remove this limitation the lock must be moved from the engine level to the device level. No more that one application at a time can own the device (i.e. call `StartUp` on that device). This means that when an application calls `StartUp()` a verification must be implemented in order to detect if the device is already in use by another application.
- Maintain backward compatibility for existing applications (i.e. same return values and behaviour). A new VisionAPI error code will be added to point out this error condition: `API_ERR_DEVICE_ALREADY_IN_USE (29)`
- Reuse `StartUp`/`ShutDown` functions without adding new functions to the VisionAPI interface (just new error code).
- `ShutDown` function shall release the device for other applications.

`StartUp` shall manage 3 different conditions:

- Device is free and must be initialized (standard condition) → return Device ID

- Device is busy (owned by another application) → return 0 (ERROR) and the API Error code is set to `API_ERR_DEVICE_ALREADY_IN_USE` (29)
- Device is already taken (zero-delay answer) → return Device ID

Applications changes

The new approach requires a modification of the final applications process too:

1. When an App2 operation is required, App1 must release the device first (calling ShutDown) and App2 shall then get control of the device (calling StartUp).
2. App2 should release the device ASAP (calling ShutDown).
3. As soon as App2 releases the scanner, App1 shall call StartUp to get back the device ownership.

NOTE: Application(s) can know if the device is already taken, looking at the DeviceID. If they have it (value \neq 0), they have control over device. Once the ShutDown is called the DeviceID shall be reset to 0. They would need to call StartUp on the device in order to re-acquire it. If the device is already under control the same DeviceID is returned and the calling delay should be 0.

PaniniUDS service

To allow different VisionAPI/Engines instances to operate the device(s), all the USB communication logic has been moved from within the VX/VN Engines to a new Windows Service that need to be installed on the client PC.

The service name is Panini USB Device Service (PaniniUDS) and its presence will be checked at device startup to set up the proper working mode. If the service is not installed or not started the VisionAPI will operate the device(s) in legacy mode, meaning one instance at a time

The Panini Universal Installer (PUI) for API 4.5.0 will take care of installing and this new service component alongside with the VisionAPI, DLLs and USB driver.

NOTE: The Panini UDS will be installed as an optional component (by default is not selected in the PUI) and, if not installed, no changes are required to existing applications.