Panini Vision API Version 3.0 Reference Manual rev. 2

March 31, 2008

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

Contents

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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.

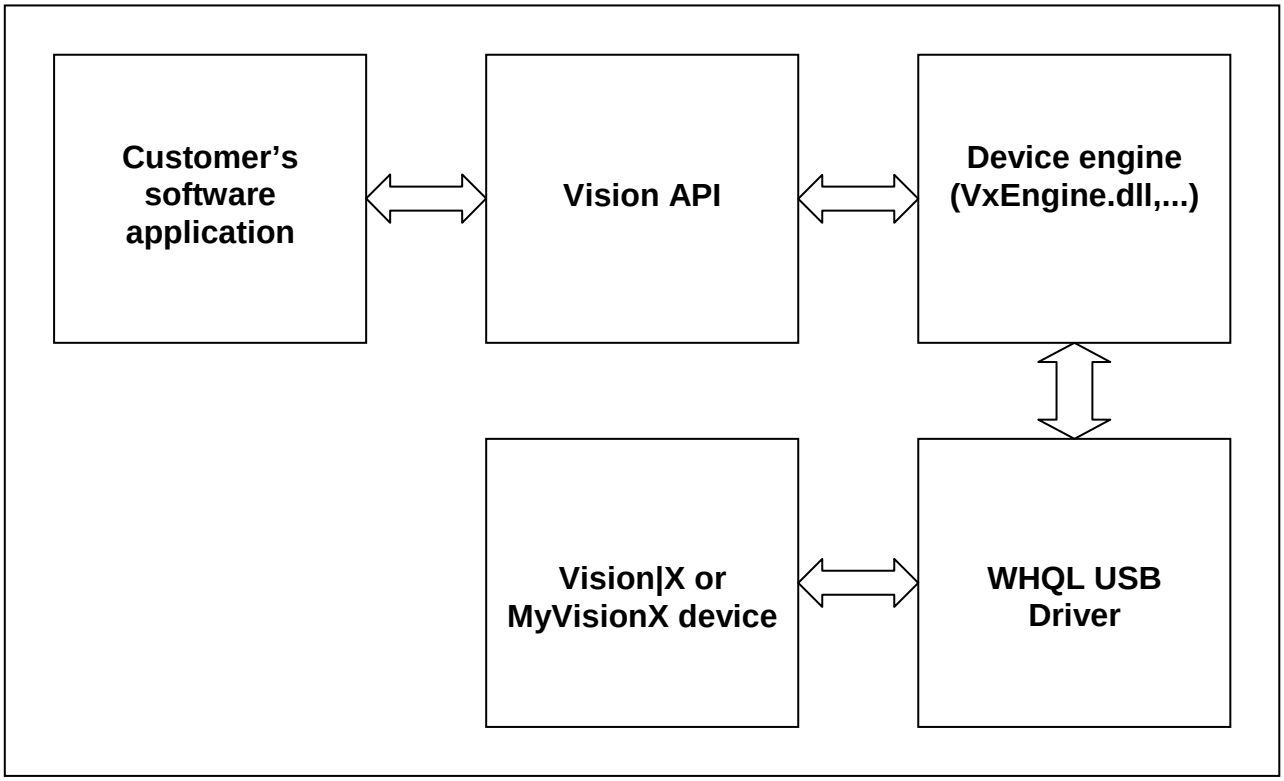
Panini Vision API philosophy is very simple: we provide our C++ header file with the prototypes of the function, the .dll and the .lib API files. The include file is written and tested with Visual C++ Microsoft compiler.

The interface is composed of two files.

1. V*isionAPI.dll* is the software interface exposed to the customers' application.
2. *VxEngine.dll* is the "device engine" that manages all the specific operations with the physical MyVisionX or Vision|X device.

The following chart shows the actual software organization:

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.



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

- VisionAPI.dll;
- VxEngine.dll (in the future this file could change to drive other Panini's devices);
- Baroc.dll (OCR engine, optional);
- AxBar32.dll (Barcode engine, optional);
- PaniniOCR.dll (OCR engine for MICR+OCR, 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 WHQL USB Driver.

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 Vision|X device supports all the engine's functions. In the

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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.

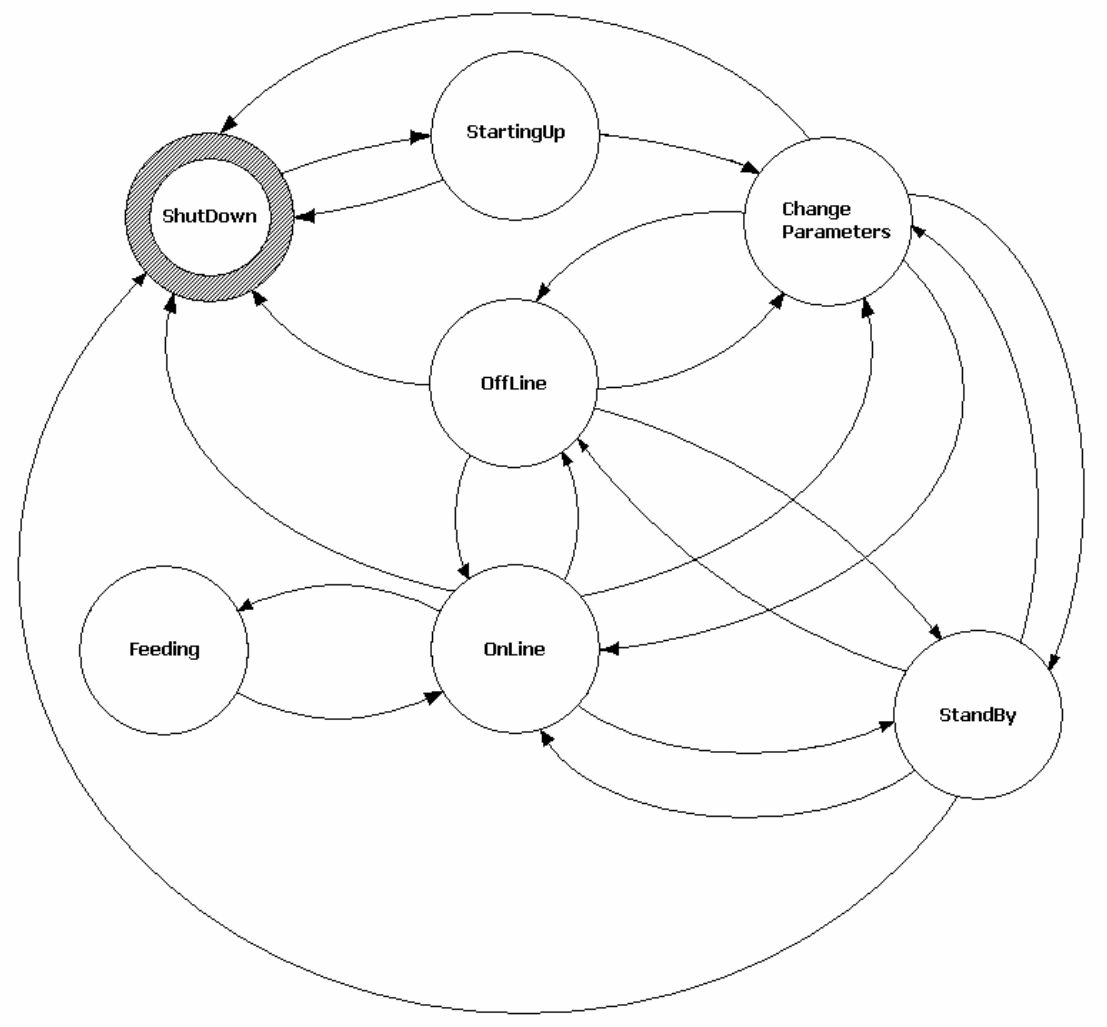
The “device” abstraction is substantially a Finite State Machine, in which every state describes the physical status of the reader. With this assumption, it’s easier to understand the real behaviour of the machine. The actual state of the Device can be queried at any time using the ***GetDeviceStateString***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

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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.

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

- ***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

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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_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.
4. ***WMPAR_SORTER_DOCUMENT_INSIDE_POCKET*** is sent when the document has been pocketed. The LPARAM of this message reports the document number.
5. ***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).
6. ***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.
7. ***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)

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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

- BOOL ***bMICREnable*** : Enabling of the MICR Recognition.
- UINT ***nMICRFont*** : Set the MICR font (MICR_FONT_CMC7 for CMC7 documents, MICR_FONT_E13B for E13B documents or 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 ***bMICRSaveSamples*** : Save MICR waveform 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 ***nMICRSpaces*** : Sets the number of spaces in the MICR codeline. (SPACES_NONE for no spaces between the check fields, SPACES_ONE for only one space or SPACES_ALL for all the spaces on the document).
- char ***cRejectSymbol*** : 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 ***nReserved*** : Must be set to 0.
- BOOL ***bReserved*** : Must be set to FALSE.

- IMAGE_PROPERTIES ***ImagePropertiesFront1*** : Image property structure for the first front image (see description below).

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

- IMAGE_PROPERTIES ***ImagePropertiesFront2*** : Image property structure for the second front image (see description below).
- IMAGE_PROPERTIES ***ImagePropertiesRear1*** : Image property structure for the first rear image (see description below).
- IMAGE_PROPERTIES ***ImagePropertiesRear2*** : Image property structure for the second rear image (see description below).

- SNIPPET_PROPERTIES ***SnippetProperties***[10] : Vector containing the property structure for the ten available snippets (see description below).

- BOOL ***bPrintEnable*** : Enable the Ink-Jet Printer.
  Valid values for this parameters are:
  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).
  These three values are defined in the header file.
  For the AGP Printer there are available 3 options to modify the printing quality and decrease the ink consumption.
  The options are the following:
  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).
  Example: bPrintEnable =
  PRINTER_ENABLE_LEADING |
  PRINTER_AGP_QUALITY NORMAL;
  this example enables Printer, with leading edge alignment and normal printer quality.
  To enable the Smart Jet printer the application has to add to the bPrintEnable field the following flag:
  PRINTER_ENABLE_SMART_JET.
  Example:
  bPrintEnable = PRINTER_ENABLE_LEADING |
  PRINTER_AGP_QUALITY NORMAL |

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

PRINTER_ENABLE_SMART_JET

- BOOL ***bOneDoc*** : if TRUE, just one document will be feed when ***StartFeeding*** has been called.
- UINT ***nFeedingMode*** : Select the feeder mode (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. DROP_FEED for Hand Drop Feeding Mode i.e. the feeder, even if it is empty, wait for document(s) to be inserted).

Image property structure description:

- UINT ***Format*** : Set the Compression Format. The available image formats are:
  FORMAT_NONE for no image
  FORMAT_GIV for G4 TIFF image
  FORMAT_JPEG for gray-level JPEG image format
  FORMAT_BMP for gray-level uncompressed bitmap format
  FORMAT_NATIVE_BMP for the gray-level bitmap of the image as it is retrieved from the scanners before any processing. It's intended to be used just for diagnostic purposes.
  FORMAT_JPEG_COLOR for color JPEG acquired with Panini Fast Color Technology.
  FORMAT_BMP_COLOR for color bitmap acquired with Panini Fast Color Technology.
  FORMAT_GIV_DROP_OUT_RED for G4 TIFF image acquired with the drop out of the red color.
  FORMAT_GIV_DROP_OUT_GREEN for G4 TIFF image acquired with the drop out of the green color.
  FORMAT_GIV_DROP_OUT_BLUE for G4 TIFF image acquired with the drop out of the blue color.
  FORMAT_BMP_TRUE_COLOR for True Color bitmap acquired with True Color mode.
  FORMAT_JPEG_TRUE_COLOR for color JPEG acquired with True Color mode.
  FORMAT_GIV_DOR_TRUE_COLOR for G4 TIFF image acquired with True Color mode for red drop out.
  FORMAT_GIV_DOG_TRUE_COLOR for G4 TIFF image acquired with True Color mode for green drop out.
  FORMAT_GIV_DOB_TRUE_COLOR for G4 TIFF image acquired with True Color mode for blue drop out.
  FORMAT_GIV_DROP_OUT_IR for G4 TIFF image acquired with the Infrared light

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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 image neither a FORMAT_GIV.
Fast color acquisition uses the nominal track speed. The True Color acquisition needs a track speed set to one third of the nominal speed.
A DEVICE_ERR_INVALID_PARAMETERS error will be obtained if not compatible images are requested.

- UINT ***Paging*** : Set the paging for the image (PAGING_ONLY_SINGLE for not including this image in a multi-page Tiff, PAGING_ONLY_MULTI for including it in the multi-page Tiff and not as a single image, or PAGING_SINGLE_MULTI for obtaining the image in each format). Bitmaps will not be included in the multi-page TIFF.
- UINT ***Resolution*** : Set the resolution of the image (100, 200 or 300 DPI are allowed), 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). 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).
- UINT ***ColorDepth*** : 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). For historical reason all values are still defined, but the only supported is 256. If an application sets a different value, it will be ignored.
- UINT ***Threshold*** : Set the Jpeg quality factor for Jpeg images (1-99) or the G4 threshold (1-9). For bitmap images, it won't be used.

Snippet Property structure description:

- BOOL ***Enable*** : Enable snippet.
  Valid values for this parameter are:
  SNIPPET_DISABLE – Snippet disable
  SNIPPET_ENABLE – Snippet enabled
  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…)
- BOOL ***Front*** : TRUE if the snippet is on the front of the document, FALSE if it is in the rear.
- Snippet ***Properties*** : Snippet structure (see below).

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

Snippet Structure description:

- UINT ***Xposition*** : X position of the Snippet. Horizontal position referrers to the LOWER-RIGHT corner for the FRONT side, and to the LOWER-LEFT corner for the REAR side.
- UINT ***Yposition*** : Y position of the Snippet.
- UINT ***Width*** : Snippet Width
- UINT ***Height*** : Snippet Height
- UINT ***Orientation*** : Snippet Orientation (see below what orientation is available).
- UINT ***Color*** : Snippet Color (SNIPPET_COLOR_GREY_LEVEL for grey level bitmap, SNIPPET_COLOR_BLACK_WHITE for monochromatic bitmap or 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 ***Compression*** : Not Implemented.
- BOOL ***Millimeters*** : TRUE if the dimensions are in mm, FALSE if they are in mils.

Note: If for both the X position and Y position, Width and Height is set to a zero value, 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 in bitmap uncompressed format.
The orientations are:

- 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.

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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_COLOR	FORMAT_GIV_DROP_OUT_RED FORMAT_GIV_DROP_OUT_GREEN FORMAT_GIV_DROP_OUT_BLUE
FORMAT_BMP_COLOR	FORMAT_GIV_DROP_OUT_RED FORMAT_GIV_DROP_OUT_GREEN FORMAT_GIV_DROP_OUT_BLUE
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_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_GIV_DROP_OUT_BLUE	FORMAT_GIV FORMAT_NATIVE_BMP FORMAT_BMP_COLOR FORMAT_JPEG_COLOR FORMAT_GIV_DROP_OUT_RED FORMAT_GIV_DROP_OUT_GREEN
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_TRUE_COLOR	FORMAT_BMP_COLOR

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

	FORMAT_JPEG_COLOR FORMAT_GIV_DROP_OUT_RED FORMAT_GIV_DROP_OUT_GREEN FORMAT_GIV_DROP_OUT_BLUE
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_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_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_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

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 information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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).

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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.

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 to manner 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

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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 information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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 addiction 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).

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.
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* filed of the *DeviceParameters* structure.

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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.

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:

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

- **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_ERR_NONE	0 : No error
USB_ERR_INVALID_DEVICE_NAME	1 : Invalid USB device name.
USB_ERR_OPEN_DRIVER	2 : Error opening driver
USB_ERR_CLOSE_DRIVER	3 : Error closing driver
USB_ERR_GET_PIPE_INFO	4 : Get pipe info failure
USB_ERR_GET_DEVICE_DESCRIPTOR	5 : Get device descriptor failure
USB_ERR_DEVICE_IO_CONTROL	6 : Device I/O Control call failure
USB_ERR_WRITE_BULK	7 : Write bulk pipe failure
USB_ERR_READ_BULK	8 : Read bulk pipe failure
USB_ERR_WRITE_CONTROL	9 : Write control failure
USB_ERR_READ_CONTROL	10 : Read control failure
USB_ERR_GET_LINK_STATE	11 : Get link state failure
USB_ERR_SEND_LINK_MESSAGE	12 : Send link message failure
USB_ERR_RECEIVE_LINK_MESSAGE	13 : Error receiving link message
USB_ERR_PROGRAM_FPGA	14 : Program FPGA error
USB_ERR_CHECK_FPGA	15 : Check FPGA programming done
USB_ERR_WRITE_E2PROM	16 : Error writing E2PROM
USB_ERR_READ_E2PROM	17 : Error reading E2PROM
USB_ERR_SET_TIME_LEDS	18 : Error setting LEDs time

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

USB_ERR_SET_GAINS	19 : Error setting LEDs gains
USB_ERR_SET_OFFSETS	20 : Error setting LEDs offsets
USB_ERR_WRITE_DMA_BULK	21 : Error writing DMA bulk pipe failure
USB_ERR_READ_DMA_BULK_PARAM	22 : Error reading DMA bulk pipe – parameter
USB_ERR_READ_DMA_BULK_SETUP	23 : Error reading DMA bulk pipe – setup
USB_ERR_READ_DMA_BULK_PHASE1	24 : Error reading DMA bulk pipe – phase 1
USB_ERR_READ_DMA_BULK_PHASE2	25 : Error reading DMA bulk pipe – phase 2
USB_ERR_READ_DMA_BULK_CLEAR	26 : Error reading DMA bulk pipe – clear
USB_ERR_READ_ID_CARD	27 : Error reading ID CARD EEPROM
USB_ERR_WRITE_ID_CARD	28 : Error writing ID CARD EEPROM
USB_ERR_SET_DPM	29 : Error setting DPM

Device Errors

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 : Wait feed done Timeout
DEVICE_ERR_WAIT_LAST_DOC_POCKETED_TIMEOUT	5 : Wait last document pocketed timeout
DEVICE_ERR_READ_SORTER_STATUS	6 : Read sorter status failure
DEVICE_ERR_SORTER_ERROR_PENDING	7 : Sorter error pending: photocell calibration failure or jammed document
DEVICE_ERR_COMMUNICATION_FAILURE	8 : Communication failure reported by the sorter
DEVICE_ERR_FEED_FAILURE	9 : Feed failure

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

DEVICE_ERR_FULL_POCKET	10 : Full pocket detected
DEVICE_ERR_SAFETY	11 : Open cover
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 : Set pocket command failure
DEVICE_ERR_GET_LAST_DOC_ID	16 : Failure reading last document ID
DEVICE_ERR_READ_MICR_SIZE	17 : Failure reading MICR waveform size
DEVICE_ERR_READ_MICR_WAVEFORM_CHUNK	18 : Failure reading MICR waveform chunk
DEVICE_ERR_DIFFERENT_MICR_SIZE	19 : Different returned MICR size
DEVICE_ERR_MICR_BUFFER_OVERFLOW	20 : MICR buffer Overflow error
DEVICE_ERR_MICR_READING	21 : MICR reading failure
DEVICE_ERR_ACQUISITION_FAILED	22 : Not able to acquire Image
DEVICE_ERR_COMPRESSION_ERR	23 : Error in Compression Thread
DEVICE_ERR_READ_ERROR	24 : Read sorter error failure
DEVICE_ERR_CREATE_COMPENSATION_TABLES	25 : Failure creating compensation tables
DEVICE_ERR_ID_CARD	26 : Failure Decoding the ID Card
DEVICE_ERR_DEVICE_ENABLING	27 : Failure Enabling, device not available
DEVICE_ERR_LOADING_OCR	28 : Failure loading OCR engine
DEVICE_ERR_CONNECTION_TIMEOUT	29 : Connection

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

	timeout
DEVICE_ERR_FEEDER_EMPTY	30 : Feeder empty
DEVICE_ERR_FREE_TRACK_FAILED	31 : Not able to free the track
DEVICE_ERR_SOFTWARE_OVERLOAD	32 : Internal error
DEVICE_ERR_STANDBY	33 : Internal error
DEVICE_ERR_NOT_COMPATIBLE_FW	34 : Internal error
DEVICE_ERR_INVALID_SEND_PRINTER	35 : Invalid data sent to Printer
DEVICE_ERR_UNSUPPORTED_ID_CARD	36 : The IDCard contains unsupported information
DEVICE_ERR_INVALID_PARAMETERS	37 : Application is setting wrong device parameters
DEVICE_ERR_FEEDER_LIMIT	38 : The feeder has reached the docs limitation
DEVICE_ERR_WAITING_FEEDER_EMPTY	39 : The device is waiting for feeder empty for docs limitation has been reached
DEVICE_ERR_FEEDER_LIMIT_RESET	40 : The feeder is ready to work after reset
DEVICE_ERR_NOT_AVAILABLE_MAX_DPM	41 : Not available DPM Number
DEVICE_ERR_NOT_CALIBRATED_IMAGE	42 : Not Calibrated Images
DEVICE_ERR_UPGRADE_UNSUCCESSFUL	43 : Error Upgrading IDCard
DEVICE_ERR_NO_PREREQUISITE_FOR_UPGRADE	44 : No Upgrade prerequisite found

API Errors

API_ERR_NONE	0 : No error
API_ERR_DEVICE_CREATE	1 : Device creation failure
API_ERR_BRIDGE_THREAD	2 : Failure creating or resuming Bridge Thread during API start-up or shut-down
API_ERR_POLLING_THREAD	3 : Failure creating or resuming Polling Thread during API start-up or shut-down
API_ERR_COMPRESSION_THREAD	4 : Failure Creating or Resuming Compression Thread during API start-up or shut-down

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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 : Buffer allocation failure
API_ERR_NO_USB_PORT_AVAILABLE	8 : No USB port available during the start-up of the API
API_ERR_LOG	9 : Error writing function call to Log file
API_ERR_INVALID_DEV_ID	10 : API method called with an invalid 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 inside the current API state
API_ERR_DEVICE	14 : Device Error
API_ERR_USB	15 : USB Error
API_ERR_FPGA_PROGRAMMING	16 : FPGA programming error
API_ERR_E2PROM_WRITE	17 : E2PROM write error
API_ERR_E2PROM_READ	18 : E2PROM read error
API_ERR_SET_TIME_LEDS	19 : Set time LED error
API_ERR_SET_GAINS	20 : Set gains error
API_ERR_SET_OFFSETS	21 : Set offsets error
API_ERR_SERIAL_PORT	22 : Error opening or closing serial port
API_ERR_SEND_SERIAL_COMMAND	23 : Error sending command through serial port
API_ERR_NOT_AVAILABLE_DEVICE	24 : Device not available

Sorter Errors

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

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

Peripheral Codes

GENERAL	0x00 : No Specific Peripheral
COMMUNICATION	0x01 : Error in Communication
READER_MODULE	0x02 : Error in MICR Reading 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_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 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_DOUBLE_FEEDING	0x08 : Double-feed 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

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.

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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.

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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
}
……
```

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

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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
}
```

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

These error’s codes are returned directly from the VisionAPI functions (VApiGetVersion, VApiSetDeviceEngine, VApiGetError and VApiGetErroString) 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.

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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.

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 );
……..
```

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.

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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)

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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.

Return Value: TRUE if successful.

Example

```
BYTE IDCardBuffer[200];
```


The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

```
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 53xxxxx
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

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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).

Return Value : TRUE if successful.

Example

```
FILE *pIDCardUpgradeFile;
```


The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

```
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 DEVICE_VISIONX	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		It’s the

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

				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 DEVICE_FEATURE_YES	This value identifies an SD device
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 DEVICE_PRINTER_AGP	This value identifies 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

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

				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
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

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.

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

Example

```
char SerialNumber[20];
DWORD FeatureValue;
……….
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(…);
      }
……..
```


The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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.

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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
```


The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

```
        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).

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 );
      ……
      }
```

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.

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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(…);
      }
……..
```

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;
```


The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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

GetApiErrorString

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(…);
      }
```

Device States Control Functions

GetDeviceState

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

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

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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) )
      {
      ……
```


The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

```
        // 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

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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


The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

```
      // “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.

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.

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(…);
      }
```


The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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(…);
      }
……..
```

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

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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.

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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;
      }
```

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).

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:
```


The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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

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.

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

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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

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. Vision|X support 50, 75 and 100 DPM.

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

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

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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)
            }
```


The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

SetAGPLines

BOOL S***etAGPLines***(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 ***GetIDCardDesciption*** 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 ***GetIDCardDesciption*** 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.

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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.

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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(…);
}
```


The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

```
else
{
      if( Pockets == 2 )
      {
            if( !SetMaxPocket( DeviceID, 2 ) )
            {
                  // Report error
                  GetApiErrorString( ApiErrorString, 200 );
                  MessageBox(…);
            }
      }
}
```

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("…");
            }
```


The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

```
            else if( !StartFeeding( m_DeviceID ) )
                  {
                  // Report error
                  GetApiErrorString( ApiErrorString, 200 );
                  MessageBox("…");
                  }
            }
```

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

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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.

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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.

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

```
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).

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:
```


The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

```
//  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
swicth( 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_E13BOXOCRBUK	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

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

OCR1_OCRAN	0x000A : OCRA numeric for use with OCRB1403
OCR1_BC128	0x0010 : Barcode Code 128
OCR1_BC39	0x0011 : Barcode Code 39
OCR1_BC2OF5	0x0012 : Barcode Interleaved 2 of 5
OCR1_BCUPCA	0x0013 : Barcode UPCA
OCR1_BCEAN13	0x0014 : Barcode EAN 13
OCR1_BCUPCE	0x0015 : Barcode UPCE
OCR1_BCEAN8	0x0016 : Barcode EAN 8
OCR1_BCPDF417	0x0017 : Barcode PDF417
OCR1_CMC7O	0x0018 : CMC7 Optical

The valid range for the Reject threshold is 10-100, lower value generates more rejects, and reasonable values are between 35 and 55.

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 ) )
```


The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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

SendPrinterData

BOOL ***SendPrinterData*** (DWORD DeviceID, DWORD dwHead, LOGFONT lf, char *sText, DWORD dwTextOffset, char *sImgPath, 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: <u>Smart Printer disabled</u>
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

<u>Smart Enabled</u>
This function has to be called during the

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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 **only for the AGP with 2 lines** enabled and **when there's no text to print** (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

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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
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 );
```

FreeImageBuffer

BOOL ***FreeImagesBuffer***(ImagesStruct *pImageStruct)

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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;
```


The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

```
            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***

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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) );
```


The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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***

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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)

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

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
}
```

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	Magcard decoding successful. The application has to call GetMagCardResult in order to obtain the result (ASCII string).

The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

MAGCARD_ERROR_PARITY	A parity error has been detected. Ask for a new read. No data returned.
MAGCARD_ERROR_STANDARD	The standard of the magcard is not supported. The application has to call GetMagCardResult in order to obtain the 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	The card has been passed in the wrong direction. No data returned.

GetMagCardResult

BOOL ***GetMagCardResult***(DWORD DeviceID, PCHAR sResult, DWORD BufferLen)

This function has to be called when the application receive the message WMPAR_MAGCARD. When the magcard has been decoded without error, it returns an ASCII string. When the standard of the magcard bit-stream it's not recognized, the function return the raw bit-stream data in order to permit to the application to decode the magcard by itself. When parity or direction error has been detected no data will be returned.
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 to allocate at least MAGCARD_MIN_BUFFER_LEN bytes.

Example

```
case WMPAR_MAGCARD:
{
  DWORD ErroCode = (DWORD)lParam;
  char sMagCardResult[MAGCARD_MIN_BUFFER_LEN];

  memset( sMagCardResult, 0, sizeof(sMagCardResult) );
  if( GetMagCardResult( DeviceID,sMagCardResult,sizeof(sMagCardResult) ) )
  {
    switch( ErroCode )
    {
      case MAGCARD_ERROR_NONE:
      {
        MessageBox( sMagCardResult, "MagCard reader result",
                        MB_OK|MB_ICONINFORMATION|MB_TOPMOST );
```


The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.

```
      }
      break;

      case MAGCARD_ERROR_PARITY:
        MessageBox( "Data corrupted!\nPlease, pass again the card.",
                        "MagCard reader result",
                       MB_OK|MB_ICONWARNING|MB_TOPMOST );
      break;

      case MAGCARD_ERROR_STANDARD:
      {
        MessageBox( "Card not supported!\nBitstream on disk.",
                         "MagCard reader result",
                          MB_OK|MB_ICONSTOP|MB_TOPMOST );

        FILE *f = fopen( "MagCardBitStream.bin", "w+" );
        if( f )
        {
          fwrite( sMagCardResult, 1, sizeof(sMagCardResult), f );
          fclose(f);
        }
      }
      break;

      case MAGCARD_ERROR_DIRECTION:
        MessageBox( "Wrong reading direction! ",
                      "MagCard reader result",
                        MB_OK|MB_ICONSTOP|MB_TOPMOST );
      break;
    }
  }
  else
  {
    // API error
  }
}
break;
```


The information contained herein is the confidential, unpublished property of Panini SPA and all unauthorized use and reproduction is prohibited. Copyright © 2005 by Panini Spa, Turin, Italy. All rights reserved.