OCR Shop XTR/API User Documentation Reference Manual

Generated by Doxygen 1.3.2

Tue Aug 19 11:07:51 2003

Contents

Chapter 1

Vividata™'s OCR Shop XTR™/API Documentation v.5.1

Updated: August 19, 2003

The OCR Shop XTR™/API Beta Release documentation is archived at `http://www.vividata.com/xtrapi-beta`

The OCR Shop XTR™/API 5.01 Release documentation is archived at `http://www.vividata.com/xtrapi-5.01`

Main Documentation

Additional Information

1.1 Introduction

Main user documentation for the OCR Shop XTR™/API.

The OCR Shop XTR™/API Software Developer's Kit is the latest in Vividata's series of character recognition and image processing products. The OCR Shop XTR™/API provides unsurpassed recognition accuracy for Sun Solaris™and Linux™production environments and for developers looking to embed powerful OCR functionality into custom applications. Vividata™has applied its dozen years of experience in document imaging, character recognition, and Unix technology, to combine the ScanSoft174 SDK Version 5.0 with Vividata™'s extensive image-processing technologies to develop the OCR Shop XTR™/API. Further, with options to output PDF and HTML documents that combine recognized text with embedded images, OCR Shop XTR™/API provides the flexibility, accuracy and power to meet the production needs of high-end custom development requirements.

In this documentation, "you" refers to the application developer and "development seat" refers to the application developer's machine, where the OCR Shop XTR™/API is installed. The "end user" refers to the application end user and the "target system" refers to the application end user's machine.

1.1.1 Features and Benefits

Powerful

- Recognizes text in virtually any typeface and size, from 5 to 72 points.
- Supports 56 different languages including English, Eastern and Western European languages, Cyrillic-based languages (Russian), the Baltic languages, Turkish and Greek.

Accurate

- Automatically segments images to correctly recognize text on pages with complex or irregular layouts, including tables, reverse video, and line art.
- Increases recognition accuracy through built-in lexical processing and user-defined lexicons.

Flexible

- Decodes more than a dozen different image formats including TIFF, jpeg, gif, raw bitmap, PDF and PostScript174.
- Outputs text in standard ASCII, Unicode, XDOC and other formats, and image areas in up to 15 different formats.

1.2 About Optical Character Recognition

OCR Shop XTR™/API utilizes ScanSoft174 OCR technology to accomplish its fast and accurate optical character recognition (OCR).

OCR is the process of transferring text from printed pages into a computer file on screen that can be edited without retyping. It is the process of translating bitmap image data into editable text. Text characters are designed by assigning a code corresponding to each key on the keyboard, be it a letter, number or symbol. There are a variety of different code sets in use, but the most common code set is the ASCII (American Standard Code for Information Interchange) table of character equivalents.

1.3 Technical Specifications

The OCR Shop XTR™/API is available for these platforms:

- Sun™Solaris™SPARC174 (Solaris™2.7+)
- Linux™x86 (Kernel 2.0 and higher)

The recognition engine is based on the ScanSoft174 SDK 5.0

Note: All features are included with and are functional in the development seat. However, for end user run-time licenses, some features and input/output formats are supported as add-on options and are not licensed in a basic run-time license key. Please contact Vividata™, Inc. Sales and Support at (510) 658-6587 Ext. 107 for pricing information and details.

Supported input formats:

- GIF
- JPEG
- PBM
- PDF
- PNG
- PPM
- PostScript174
- Rasterfile
- SGI174-RGB
- TIFF
- XWD
- X11

Supported languages:

Text recognition uses a character set and one or more languages. English is the default language and all available languages are included with the development seat. For run-time licenses, additional languages are available as add-on language packs. Multiple languages may be specified for a single document, as long as they use the same character set.

Character sets:

- The character set defines the shape of the letters.
- Only one character set may be used at a time.
- English characters are an exception, and may be recognized in conjunction with any character set by using the special `dm_english_chars` value.
- The character set is chosen implicitely based on the language(s) selected by the user.

Languages:

- A language is associated with one character set.
- Only languages using the same character set may be specified concurrently.
- A document using a language with a dictionary is recognized based on the character shapes and with reference to the dictionary file specific to the chosen language.
- A document using a language without a dictionary is recognized based on the character shapes alone, without reference to a dictionary file.

- Languages with dictionaries:
  - czech (Central Europe - 1250)
  - danish (Latin 1 - 1252)
  - dutch (Latin 1 - 1252)
  - english (Latin 1 - 1252)
  - finnish (Latin 1 - 1252)
  - french (Latin 1 - 1252)
  - german (Latin 1 - 1252)
  - greek (Greek - 1253)
  - hungar (Hungarian; Central Europe - 1250)
  - italian (Latin 1 - 1252)
  - norsk (Norwegian; Latin 1 - 1252)
  - polish (Central Europe - 1250)
  - port (Portuguese; Latin 1 - 1252)
  - russian (Cyrillic -1251)
  - spanish (Latin 1 - 1252)
  - swedish (Latin 1 - 1252)
  - turkish (Turkish - 1254)

- Languages without dictionaries:
  - romanian (Central Europe - 1250)
  - estonian (Baltic - 1257)
  - afrikaans (Latin 1 - 1252)
  - albanian (Central Europe - 1250)
  - aymara (Latin 1 - 1252)
  - basque (Latin 1 - 1252)
  - breton (Latin 1 - 1252)
  - bulgarian (Cyrillic - 1251)
  - byelorussian (Cyrillic - 1251)
  - croatian (Central Europe - 1250)
  - faroese (Latin 1 - 1252)
  - flemish (Latin 1 - 1252)
  - friulian (Latin 1 - 1252)
  - gaelic (Latin 1 - 1252)

- galician (Latin 1 - 1252)
- greenlandic (Latin 1 - 1252)
- hawaiian (Baltic - 1257)
- icelandic (Latin 1 - 1252)
- indonesian (Latin 1 - 1252)
- kurdishlat (Kurdish Latin; Turkish - 1254)
- latin (Latin 1 - 1252)
- latvian (Baltic - 1257)
- lithuanian (Baltic - 1257)
- sorbianl (Sorbian - Lower; Central Europe - 1250)
- macedonianc (Cyrillic - 1251)
- malaysian (Latin 1 - 1252)
- piginenglish (Latin 1 - 1252)
- serbian (Cyrillic - 1251)
- ukranian (Cyrillic - 1251)
- catalan (Latin 1 - 1252)
- sbcroatian (Serbo-Croation; Central Europe - 1250)
- slovak (Central Europe - 1250)
- slovenian (Central Europe - 1250)
- swahili (Latin 1 - 1252)
- tahitian (Latin 1 - 1252)
- sorbianu (Sorbian - Upper; Central Europe - 1250)
- welsh (Latin 1 - 1252)
- frisianw (Frisian - West; Latin 1 - 1252)
- zulu (Latin 1 - 1252)

Supported output text document formats:

- iso text, 8bit text, Unicode text, XDOC, pdf, html (to be supported in a future release)

Supported output subimage (graphical) formats:

- tiff, ras, epsf, x11, tiff-pack, tiff-g31d, tiff-g32d, tiff-g42d, pal-tiff, jpeg, png, xwd, rgb, rgb-rle, pdf

1.4 Installation

For installation of the OCR Shop XTR™/API, you should have downloaded one distribution file from Vividata's ftp site.

For Linux:

- xtrapi-linux-5.1r19

For Solaris:

- xtrapi-sun_5x-5.1r19

To install the API:

1. Place the distribution file in any temporary directory on your development seat machine.
2. Log in as root on your development seat machine.
3. Make sure the environment variable `LD_LIBRARY_PATH` is set. Normally this environment variable is set to at least `/lib:/usr/lib`.
4. From the directory containing the distribution file, run the command:

   For Linux:

   ```
   ./xtrapi-linux-5.01r0
   ```

   For Solaris:

   ```
   ./xtrapi-sun_5x-5.01r0
   ```

5. This will install the OCR Shop XTR™/API, placing these files on your system:

`/opt/Vividata/`	All files in the API distribution
`/opt/Vividata/lib/`	Libraries, resource files
`/opt/Vividata/include/`	Headers
`/opt/Vividata/src/`	Source code for a sample program
`/opt/Vividata/ghostscript/`	Ghostscript resource files
`/opt/Vividata/bin/`	Binaries

In addition, symbolic links are created in /usr/local/include/ and /usr/local/lib/ to the include files and libraries in /opt/Vividata/.

The default location for the development seat installation is /opt/Vividata/. If you decide to install the Vividata™development seat elsewhere, you must set the environment variable `VV_HOME` before running the installer, installing the run-time license key, or using the OCR Shop XTR™/API functionality.

You can set the temporary directory used during installation and use of the OCR Shop XTR™/API by setting the environment variable `TMP_DIR` to a specific directory. `/tmp` is used by default.

You must install the run-time license key for testing included with your development seat before you can run a sample program. Please see the subsection Development Seat Licensing.

Please see the section on Basics of Usage for information on running the included sample program and using the API.

To uninstall:

To uninstall the API development seat, remove `/opt/Vividata`, as well as the symbolic links in `/usr/local/include/`:

- `vvlicense`
- `vvocr`
- `vvutil`

and the symbolic links in `/usr/local/lib/`:

- `libboost_thread.a`

- `libvvocr_communicator.a`
- `libvvocr_factory.a`
- `libvvutil.a`

1.5 API Design

The OCR Shop XTR™/API is designed to work as a client/server system, in which your application links statically with the provided communicator library, so that it may dynamically communicate with a daemon process containing the OCR engine. All function calls from your application are made to the statically linked libraries.

The main daemon process permits the creation of multiple instances of the OCR engine, serving one or more client programs. Furthermore, the API is designed in such a way that the OCR engine code is isolated from your application to increase process stability and to reduce dependencies, such that the OCR engine and your application do not share namespace, and a crash in your application will not affect the OCR engine.

1.6 Run-Time Usage

Run-Time usage describes the installation and use of your application on the end user's target system.

In order for the OCR engine to function, you must set up the licensing and resource files properly on the end user's target system.

- Run-Time Licensing
- Resource File Requirements
- Resource File Distributions

1.6.1 Run-Time Licensing

A run-time license key must be purchased for and installed on the target system in order for the OCR Shop XTR™/API software to run within your application.

Please see the section Licensing for information on how licensing works in general and specifically how to obtain and install purchased run-time licenses for an end user's target system.

1.6.2 Resource File Requirements

OCR Shop XTR™/API resource files must be installed on the target system in order for the OCR Shop XTR™/API software to run properly within your application.

You may install the resource files yourself in the process of installing your application, or we can provide a redistributable installer. The redistributable installer is similar to the development seat installer, but it installs only the appropriate resource files for a target machine. Please contact us for information on obtaining and using a redistributable installer.

The default location on the target system for the run-time resource files is `/opt/Vividata/`. If you decide to install the Vividata resource files elsewhere, you must set the environment variable `VV_HOME` before

running the installer, installing the run-time license key, or using the OCR Shop XTR™/API functionality. This ensures that the OCR Shop XTR™/API code can find the resource files and license key file.

1.6.3 Resource File Distributions

Below is a list of the resource files you should distribute with your application. These files are all included with your development seat.

Note: You are not permitted to redistribute the OCR Shop XTR™/API libraries or include files.

- `/opt/Vividata/bin/:`
  `gs*`
  `ocrxtrdaemon*`
  `vvlmreread*`
  `vvlmstop*`
- `/opt/Vividata/bin/linux/:`
  `gs*`
  `ocrxtrdaemon*`
  `vvlmreread*`
  `vvlmstop*`
- `/opt/Vividata/config/`
  `The license key file will be installed here.`
- `/opt/Vividata/ghostscript/:`
  `Fontmap`
  `Fontmap.GS`
- `/opt/Vividata/lib/images/:`
  `but-blank.gif`
  `but-next.gif`
  `but-prev.gif`
  `index.gif`
- `/opt/Vividata/lib/langs/:`
  `asciieng.lng`
  `BALTIC.shp`
  `CharSetTable.chr`
  `CYRILLIC.shp`
  `czech.lng`
  `danish.lng`
  `dutch.lng`
  `english.lng`
  `finnish.lng`
  `french.lng`

```
german.lng
greek.lng
GREEK.shp
hungar.lng
italian.lng
LATIN1.shp
LATIN2.shp
norsk.lng
polish.lng
port.lng
russian.lng
spanish.lng
swedish.lng
turkish.lng
TURKISH.shp
```

Note: Only language files (.lng) and shape packs (.shp) that will be used on the target system, as regulated by the run-time licensing, are required to be installed on the target system.

1.7 Licensing

Licensing affects both the development seat and the end user's target system. A license key is typically tied to a machine's hardware id, and regulates how many concurrent instances (OCR engines) may run, PDF/PS input, PDF output, HTML output (in a future release), and languages. Alternative licensing is possible under contract on a case-by-case basis; please contact us for more information.

- Development Seat Licensing
- Run-Time Licensing
- How to Obtain Run-Time License Keys
- How to Install Run-Time License Keys
- Troubleshooting License Problems

1.7.1 Development Seat Licensing

One development seat consists of:

- One copy of the API distribution, including libraries, headers, and sample code to be installed on one machine.
- One point of contact, meaning one application developer working with the API who can contact Vividata™for support in using the API.
- One run-time license key for testing on the development seat machine; this license key includes 5 licenses (instances), all features enabled, and all languages. You will receive the license key along with the API distribution.

Development seat licensing is regulated by the terms of the Vividata Developer License Agreement unless superceded by a previously executed contractual agreement.

How to obtain the run-time license key for testing which comes with your development seat:

A run-time license key with all features permitted will be emailed to you in conjunction with the development seat distribution.

How to install the development seat run-time license key for testing:

The license key comes encapuslated in a shell script. After installing the development seat distribution, you must install the run-time license key by running the shell script as root with the command: `sh vvlicense.sh`

This will place your license key in the file `/opt/Vividata/config/vvlicense.dat`.

If you are using a directory other than the default `/opt/Vividata/` for the development seat installation directory, you must set the environment variable `VV_HOME` to the correct directory before installing the license key.

1.7.2 Run-time Licensing

In order to run your application on the end user's target system, a "run-time" license key must be installed. A license key defines how many licenses (instances) are available on a particular machine, as well as what features and languages are permitted for those licenses. All licenses on the same machine must share the same feature set.

A run-time license is checked out at the first occurrence of one of these events:

- The input image file is opened (`vvEngAPI::vvOpenImageFile`)
- The input image data is read into memory (`vvEngAPI::vvReadImageData(const struct vvxtrImage * img)`)
- The output document is started (`vvEngAPI::vvStartDoc`)
- Preprocessing is run (`vvEngAPI::vvPreprocess`)
- Recognition is run (`vvEngAPI::vvRecognize`)

A run-time license is checked in when any one of these events occurs:

- The output document is ended (`vvEngAPI::vvEndDoc`)
- The OCR session is ended (`vvEngAPI::vvEndOCRSes`)
- The OCR engine is killed (`vvEngAPI::vvKill`)

A run-time license key is bound to the machine id (hostid on Solaris, or MAC addres on Linux) of the end user's target machine and controls these features:

- The number of licenses on the machine (one license is the same as one concurrent instance)
- PDF/PS input
- Language usage (corresponding to the `dm_language value`)
- PDF document output
- HTML output (supported in a future release)

The run-time license key should be installed on the target machine at the same time as your application. Under normal operating conditions, all licensing will work successfully if the license key file is in place when the application runs.

Please see the sections Run-Time Usage and How to Install a Run-Time License Key.

1.7.3 How to Obtain Run-Time License Keys

We offer an automated website to permit you to generate and download purchased run-time licenses for the end-users' target machines at your convenience. To obtain purchased run-time licenses through our website:

1. Go to `http://www.vividata.com/login.html`
2. Log in with your user id and password (if you do not already have your user id or password, please contact us.)
3. Create a new key by entering the appropriate information in the form.
4. Confirm that you understand that you are using one of your pre-purchased run-time licenses.
5. Confirm that you understand that by creating this key, you will decrement the number of run-time licenses you have available for download, based on your original run-time license purchase.
6. The license key file will be available for download from the webpage and also sent to you by email.

If you prefer, or if you have not already purchased run-time licenses and would like to do so, we are able to provide run-time licenses directly. Please contact Radcliffe Goddard at (510) 658-6587 Ext.107 or email `supporthuman@vividata.com`.

1.7.4 How to Install a Run-Time License Key

A run-time license key must be installed on the end user's target machine before running your application.

1. You should have a license key file specifically generated for the target machine. The license key itself is encapsulated in a shell script.
2. Log in as root on the target machine.
3. If you are using a directory other than the default `/opt/Vividata/` for Vividata™'s resource files and license key, you must set the environment variable `VV_HOME` to the correct directory.
4. Run the provided license key install file with the command:

   ```
   sh vvlicense.sh
   ```

5. The license key should now be installed on the target machine in `/opt/Vividata/config/vvlicense.dat`, or another directory as specified by `VV_HOME`.

1.7.5 Troubleshooting License Problems

- If the license manager enters a bad state, reset it by running the command:

  `/opt/Vividata/bin/vvlmstop`

  Then try running your application again.

- If `vvlmstop` does not seem to work, you can kill the license process by hand. On Linux, the process should be named "vvlicense" and on Solaris, it will be named "ocrxtrdaemon".

- If you installed the Vividata™resource files and license key in a directory other than the default `/opt/Vividata/`, you must set the environment variable `VV_HOME` to the correct Vividata installation directory.

- License errors are expected if you attempt to use features that are not permitted by the license or if you try to check out more licenses (instances) than you are have.

  In both cases, you should check which features are permitted by the key and then verify that you are not trying to access an unlicensed feature.

  You can check which features should be allowed on a target machine by checking the license key installed on the machine:

  1. Open `/opt/Vividata/config/vvlicense.dat` to see the license key string itself.
  2. Pass this key string to the `vvlmstatus` utility:
     `/opt/Vividata/bin/vvlmstatus -k <keystring>`

  or by looking up the machine's hardware id on our website:

  1. Go to `http://www.vividata.com/login.html`
  2. Log in with your user id and password (if you do not already have your user id or password, please contact us).
  3. Click on the link to look up a license key.
  4. Enter the hardware ID of the target machine in the form. The hardware ID on a Solaris machine is found with the `hostid` command; the hardware ID on a Linux machine is the ethernet card MAC address, found with the `ifconfig eth0` command.
  5. Click on "Lookup" and you will be presented with a list of keys which you have purchased for that machine, along with information on what features should be allowed.

  If you think there is a discrepancy between the features you purchased for the target machine and the features the key is permitting, please contact us.

- For any other questions regarding run-time license keys or licensing in general, please contact us.

1.8 Basics of Usage

The OCR Shop XTR™/API distribution consists of a number of include files and libraries.

Your application must include these headers:

- `vvocr/vvxtrAPI.h`
- `vvocr/vvxtrFactory.h`

Additional headers are included with the OCR Shop XTR™/API distribution for your reference and use.

Your application must link with these libraries:

- `libvvocr_factory.a`
- `libvvocr_communicator.a`
- `libvvutil.a`
- `libboost_thread.a`
- `libpthread.a`
- `libsocket.a` (for Solaris only)
- `libposix4.a` (for Solaris only)
- `libstdc++.a`

The first four of these libraries are included with the API distribution and are placed on your development seat system as part of the installation process; the other libraries you should already have as part of your operating system or as part of gcc.

The OCR Shop XTR™/API libraries are compiled with **gcc 3.3**, and we highly recommend using this compiler when linking with our libraries. Please let us know if you need to use a different compiler or encounter any problems.

Before you can run the sample program or your own application, the daemon must be started. To start the daemon, run the command:

- `/opt/Vividata/bin/ocrxtrdaemon`

A sample program is included with the OCR Shop XTR™/API distribution. To compile the sample program:

- Change to the `/opt/Vividata/src/` directory.
- Run the command: `gmake`
- It will create the executable `vvxtrSample`.

You can run the sample program with the command:

- `vvxtrSample localhost`

It will prompt you for information to run a test document through the engine. We have included a sample input file: `/opt/Vividata/src/letter.tif`.

The daemon may be instructed to print out log information by setting the environment variable `VV_DEBUG` prior to starting the daemon. (See the include file `vvLog.h` for appropriate values for `VV_DEBUG`.) Similarily, your application can print out verbose log information; please see the call to `vvLogSetLevel` in `vvxtrSample.cc` and the FAQ question How do I generate log output?.

When you begin code development, we recommend you use the sample program `vvxtrSample.cc`, the header `vvxtrAPI.h`, and the webpages at `http://www.vividata.com/xtrapi` as your primary sources for documentation and information on the use of the API. The Frequently Asked Questions also provides valuable information. Please contact us for any additional questions.

1.9 Notes

The OCR Shop XTR™/API version 5.1 was released on August 19, 2003.

The API features:

- Input via a file or an image passed through memory
- Document output to ASCII, Unicode, XDOC, and PDF through a file or memory
- OCR input from GIF, JPEG, PBM, PDF, PNG, PPM, PostScript174, Rasterfile, SGI174-RGB, TIFF, XWD, and X11 image files
- Subimage output to a file or memory
- Preprocessing
- Recognition by page or region
- Various options for controlling preprocessing, recognition, input and output
- Control over region properties and boundaries, region creation and removal
- Licensing control over code compiled with our libraries

Major updates in this release include:

- Fully supported PDF output
- Many documentation updates, including the addition of a faq and more detailed information for the engine calls
- A more complicated sample program to better demonstrate usage of the API
- Various bug fixes in the API and the ScanSoft engine
- Smoother handling of vvxtrImages
- Access to the processed image data for subimage output is available through the dm_output_img_source options
- Addition of timeouts for vvRecognize and vvPreprocess
- Improved handling of vvKill
- Better handling if more than one ocrxtrdeamon is started

We plan future improvements to be included in subsequent releases, including:

- HTML output
- Detailed tweaking of deskewing parameters and image output options.

1.10 Contact Information

For support questions, please contact us:

- Online support form: http://www.vividata.com/cgi-bin/support
- Phone: Radcliffe Goddard at (510) 658-6587 Ext.107
- Email: supporthuman@vividata.com

Vividata's website: http://www.vividata.com

Vividata™, Inc.
1300 66th Street
Emeryville, CA 94608
U.S.A.
Phone: (510) 658-6587
Fax: (510) 658-6597
Toll-free Sales: (800) 704-2640

Chapter 2

OCR Shop XTR/API User Documentation Module Index

2.1 OCR Shop XTR/API User Documentation Modules

Here is a list of all modules:

Chapter 3

OCR Shop XTR/API User Documentation Compound Index

3.1 OCR Shop XTR/API User Documentation Compound List

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

Chapter 4

OCR Shop XTR/API User Documentation File Index

4.1 OCR Shop XTR/API User Documentation File List

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

Chapter 5

OCR Shop XTR/API User Documentation Page Index

5.1 OCR Shop XTR/API User Documentation Related Pages

Here is a list of all related documentation pages:

Chapter 6

OCR Shop XTR/API User Documentation Module Documentation

6.1 Ocrxtrapi_public

Files

- file vvxtrAPI.h

  vvxtrAPI.h, Public Interface for OCRShop XTR API

- file vvxtrComm.h

  vvxtrComm.h, Information needed to communicate with the daemon without the client program.

- file vvxtrDefs.h

  vvxtrDefs.h, definitions and enumerations for OCRShop XTR API

- file vvxtrFactory.h

  vvxtrFactory.h, Function declarations for the engine factory.

Compounds

- class vvEngAPI

  Optical Character Recognition Engine API Class.

- class vvxtrImage

  An encapsulation of information about a bitmap.

- struct **vvxtrKeyValuePair**

Defines for conditions

Used to construct the vvxtr_state bitstring.

- State is a bit string corresponding to a bitwise OR of all conditions.
- Used in vvEngAPI::vvGetCond function.

- #define C_READY 0x400
  ready
- #define C_OCRSESOPEN 0x200
  OCR session is open.
- #define C_OUTDOCOPEN 0x100
  output document is open
- #define C_FILEOPEN 0x080
  input image is open
- #define C_IMAGELOADED 0x040
  input page is open
- #define C_PREPROCDONE 0x020
  preprocessing has been run on the current page
- #define C_RECOGDONE 0x010
  recognition has been run on the current page
- #define C_DOCREADY 0x008
  output document is ready for aquisition
- #define C_SUBIMAGEREADY 0x004
  output subimage is ready for aquisition
- #define C_ENGINEBUSY 0x002
  engine is currently busy
- #define C_ERROR 0x001
  error

Defines for Error codes

These codes are returned as a vvxtrStatus value when there is an error.

- #define VVXTR_KILL_PROCESS -1000
  Time for the daemon to die.
- #define VVXTR_PREVIOUS_ERROR -1001
  Requesting information on the last error.
- #define VVXTR_ERR -1
  General error.

- enum vvxtrErrorCodes {

  VVXTR_ERR_INVALID_SYNTAX = 1001, VVXTR_ERR_NO_INPUT, VVXTR_ERR_INVALID_STATE, VVXTR_ERR_INITINSTANCE_FAILURE, VVXTR_ERR_ENDINSTANCE_FAILURE, VVXTR_ERR_NO_INPUT_FILE_SPECIFIED, VVXTR_ERR_UNABLE_TO_LOAD_IMAGE_FILE, VVXTR_ERR_IMG_ACQ_FAILED,

  VVXTR_ERR_START_SESSION_FAILED, VVXTR_ERR_RECOGNITION_FAILED, VVXTR_ERR_OPEN_CHAR_SET_FAILURE, VVXTR_ERR_INVALID_SHAPE_PACK_PATH, VVXTR_ERR_BAD_LANGUAGE, VVXTR_ERR_OPENING_LANG_PACK, VVXTR_ERR_LOAD_LANG_FAILED, VVXTR_ERR_CREATE_LANG_GROUP,

  VVXTR_ERR_LANG_NOT_LICENSED, VVXTR_ERR_SET_DEF_LEX_CONSTRAINTS, VVXTR_ERR_WRITE_OUTPUT_FAILED, VVXTR_ERR_OPENING_OUTPUT_FILE, VVXTR_ERR_REGION_ECLIPSED, VVXTR_ERR_COULD_NOT_CONVERT_DEPTH, VVXTR_ERR_GETTING_IMG_REGION, VVXTR_ERR_END_SESSION_FAILED,

  VVXTR_ERR_OUTPUT_INV_REGION, VVXTR_ERR_NOT_IMPLEMENTED, VVXTR_ERR_INV_PDF_FORMAT, VVXTR_ERR_PDF_IMG_OUTPUT, VVXTR_ERR_PDF_END_OUTPUT, VVXTR_ERR_UNABLE_TO_WRITE_PDF, VVXTR_ERR_HOST_LOOKUP_FAILED, VVXTR_ERR_CONNECTION_FAILED,

  VVXTR_ERR_TRANSFER_FAILED, VVXTR_ERR_NO_SUBIMAGE, VVXTR_ERR_PP_FAILED, VVXTR_ERR_SET_OPTIONS_FAILED, VVXTR_ERR_REMOVE_FILE_FAILED, VVXTR_ERR_NO_DOC, VVXTR_ERR_INV_SUBIMG_FORMAT, VVXTR_ERR_SYNCHRONIZATION,

  VVXTR_ERR_NOENGINE, VVXTR_ERR_BUFFER_TOO_SMALL, VVXTR_ERR_INVALID_PAGE, VVXTR_ERR_MISSING_LANG, VVXTR_ERR_ENG_OUT_OF_MEMORY, VVXTR_ERR_INVALID_UOR_COUNT, VVXTR_ERR_INVALID_UOR_STRING, VVXTR_ERR_READONLY_VALUE,

  VVXTR_ERR_INVALID_REGION_ID, VVXTR_ERR_SET_REGION_UNSUCCESSFUL, VVXTR_ERR_REGION_HANDLER, VVXTR_ERR_ENGINE_KILLED, VVXTR_ERR_NO_LICENSE, VVXTR_ERR_NO_LICENSE_MANAGER, VVXTR_ERR_LICENSE_COMM_ERROR, VVXTR_ERR_NO_FEATURE }

  Time for the daemon to die.

Defines

- #define **VVXTRAPI_H** 1
- #define **commandtable**(A, B, C) A,
- #define **actiontable**(A, B) xtract_##A,
- #define **statustable**(key, action, type) dm_##key,
- #define **TRANSFER_TIMEOUT** 25
- #define **XTR_READY** "xtrReady"
- #define VV_DEBUG 1

  To turn on debug messages.

- #define **tablemacro**(key, value, type, kernelFunc, defaultUsed, defaultVal, kernelVal, write) dm_##key,
- #define **VVXTRFACTORY_H** 1

Typedefs

- typedef enum vvxtr_dataType VVXTR_DATATYPE

  Generic XTR datatype.

- typedef signed long int vvxtr_state

  Representation of the state as a bit-string.

- typedef signed long int vvxtr_cond

  Representation of a single condition in the state bit-string.

- typedef signed long int vvxtrStatus

  Representation of the engine status.

- typedef void * vvxtr_stdarg

  A generic argument in the XTR API.

Enumerations

- enum vvxtrCommandKeyEnum {

  query_status, get_value, set_value, do_action, **upload_block**, **download_block**, **get_version**, **add_word_to_lexicon**,

  set_hint, **set_lexical_constraints**, **set_region_properties**, **force_region_foreground**, **remove_region**, **VVXTR_NUM_COMMANDS** }
- enum vvxtrActionEnum {

  xtract_init_instance, xtract_end_instance, **xtract_start_ocr_ses**, **xtract_end_ocr_ses**, **xtract_start_doc**, **xtract_spool_doc**, **xtract_end_doc**, **xtract_capture_subimage**,

  xtract_read_image_data, **xtract_open_image_file**, **xtract_unload_image**, **xtract_close_image_file**, **xtract_preprocess**, **xtract_recognize**, **xtract_init_values**, **xtract_kill**,

  VVXTR_NUM_ACTIONS }
- enum vvxtrStatusEnum {

  dm_daemon_state, dm_engine_state, dm_words_seen, dm_words_recognized, dm_characters_seen, dm_characters_recognized, dm_percentage_done, dm_page_number,

  dm_region_number, dm_line_orientation, dm_skew_angle, dm_current_skew_angle, dm_skew_confidence, dm_smotation_angle, dm_text_orientation, dm_min_is_black,

  dm_most_complex, dm_byte_order, dm_error_code, dm_version, dm_bitmap_split, dm_line_doubled, dm_regionview_mode, dm_codepage,

  dm_lexicon, dm_most_cols, dm_most_rows, dm_most_cells, **VVXTR_NUM_STATUS_TYPES** }
- enum { **vvOcrListenPort** = 10101, **vvOcrLicensePort** = 10102 }
- enum **TransferType** { **ttInvalid** = 0, **ttDoc**, **ttImage**, **ttImageFile**, **ttDocFile**, **ttImageFileLocal**, **ttDocFileLocal** }
- enum vvxtr_dataType { vvxtr_none = 0, vvxtr_int, vvxtr_string, vvxtr_pib, vvxtr_ptr }

  Generic XTR datatype.

- enum vvxtrResponseEnum { vvNo = 0, vvYes = 1, vvAuto = 2, vvDetect = 3, vvCorrect = 4, vvManual = 5 }

  Generic settings used for many different options.

- enum vvxtrOutTextFormatEnum {

  vvTextFormatDefault = -1, vvTextFormatNone = 0, vvTextFormatXdoc = 1, vvTextFormatXdoclite = 3, vvTextFormatXdocplus = 4, vvTextFormatPost = 13, vvTextFormatIso = 102, vvTextFormatPdf = 104,

  vvTextFormat8bit = 105, vvTextFormatUnicode = 106, vvTextFormatHtmlWysiwygP = 107, vvTextFormatHtmlWysiwygS = 108, vvTextFormatHtmlTable = 109, vvTextFormatHtmlSimple = 110 }

  Output text formats.

- enum vvxtrOutGraphicsFormatEnum {

  vvSubimageFormatNone = -1, vvSubimageFormatTiff = 0, vvSubimageFormatRas = 1, vvSubimageFormatEpsf = 2, vvSubimageFormatX11 = 3, vvSubimageFormatTiffpack = 4, vvSubimageFormatTiffg31d = 5, vvSubimageFormatTiffg32d = 6,

  vvSubimageFormatTiffg42d = 7, vvSubimageFormatTifflzw = 8, vvSubimageFormatPaltiff = 31, vvSubimageFormatGif = 17, vvSubimageFormatJpeg = 19, vvSubimageFormatPng = 21, vvSubimageFormatXwd = 22, vvSubimageFormatRgb = 23,

  vvSubimageFormatRgbrle = 24, vvSubimageFormatEPdf = 28, vvSubimageFormatVvxtrImage = 30 }

  Output graphics formats.

- enum vvxtrFocusAreaEnum { vvFocusAreaPage = 0, vvFocusAreaRegion = 1 }

  How to specify the document area on which to focus.

- enum vvxtrVerifierModeEnum { vvVerifierModeWord = 1, vvVerifierModeChar = 0 }

  Verifier mode.

- enum vvxtrRecModeEnum { vvRecModeUnspecified = 1, vvRecModeStandard = 2, vvRecModeDegraded = 3 }

  Recognition mode.

- enum vvxtrTextOutNewlineEnum { vvTextOutNewlineUnix = 1, vvTextOutNewlineMac = 2, vvTextOutNewlinePC = 3 }

  Newline designation.

- enum vvxtrPDFFormatEnum { vvPDFFormatNormal = 1, vvPDFFormatText = 2, vvPDFFormatImgOnly = 3 }

  PDF format.

- enum vvxtrRegionTypeEnum {

  vvRegionTypeAny = 3511, vvRegionTypeIgnore = 3512, vvRegionTypeText = 3513, vvRegionTypeImage = 3514, vvRegionTypeVrule = 3515, vvRegionTypeHrule = 3516, vvRegionTypeHidden = 3517, vvRegionTypeRevid = 3518,

  vvRegionTypeHiddenimage = 3519 }

  Region types.

- enum vvxtrRegionSubtypeEnum {

  vvRegionSubtypeUnflavored = 0, vvRegionSubtypeTable = 1, vvRegionSubtypeTable_inset = 2, vvRegionSubtypeHeadline = 3, vvRegionSubtypeTimestamp = 4, vvRegionSubtypeLineart = 5, vvRegionSubtypeHalftone = 6, vvRegionSubtypeInset = 7,

vvRegionSubtypeCaption = 8, vvRegionSubtypePage_footer = 9, vvRegionSubtypePage_header = 10, vvRegionSubtypeVruling = 11, vvRegionSubtypeHruling = 12, vvRegionSubtypeNoise = 13, vvRegionSubtypeIpcorePictureMask = 14 }

Region subtypes.

- enum vvxtrRegionGrammarModeEnum { vvRegGrammarModeWord = 0, vvRegGrammarModeLine = 1 }

  Region grammar modes.

- enum vvxtrRegionLexmodeEnum { vvRegionLexmodeNolex = 0, vvRegionLexmodePreference = 1, vvRegionLexmodeAbsolute = 2 }

  Region lexical mode.

- enum vvxtrRegionForegroundEnum { vvRegionForegroundBlack = 0, vvRegionForegroundWhite = 1, vvRegionForegroundUnknown = 2 }

  Region foreground specification.

- enum vvxtrRegionOpacityEnum { vvRegionOpacityOpaque = 0, vvRegionOpacityTransparent = 1 }

  Region opacity.

- enum vvxtrRegionAbutmentEnum { vvRegionAbutmentUnknown = -1, vvRegionAbutmentNone = 0, vvRegionAbutmentLeft = 1, vvRegionAbutmentRight = 2, vvRegionAbutmentLeftAndRight = 3 }

  Region abutment.

- enum vvxtrLexicalConstraintModeEnum { vvLexConstraintModeReplace = 0, vvLexConstraintModeAdd = 1 }

  Lexical constraint mode.

- enum vvxtrDisplayAlignmentEnum { vvAlign8 = 8, vvAlign16 = 16, vvAlign32 = 32, vvAlign64 = 64 }

  Bit alignment of display.

- enum vvxtrVersionLocationEnum { vvLocal = 0, vvRemote = 1 }

  Version location.

- enum vvxtrImageSpecificationEnum { vvInputImage = 0, vvProcessedImage = 1 }

  Specify the original or processed image.

- enum vvInternalEngineState {

  vvS_NONE = 0x00000000, vvS_IDLE = 0x00000001, vvS_SESSION = 0x00000002, vvS_CANCEL = 0x00000004, vvS_PGCAN = 0x00000008, vvS_ACQUISITION_READY = 0x00000010, vvS_ACQUIRING = 0x00000020, vvS_RECOGNITION_READY = 0x00000040,

  vvS_RECOGNIZING = 0x00000080, vvS_PREPROCESS = 0x00000100, vvS_TEXT_OUTPUT = 0x00000200, vvS_REGION_MAPPING = 0x00000400, vvS_SHAPES_LOADED = 0x00000800, vvS_AWAITING_DRAW = 0x00001000, vvS_AWAITING_VERIFIER = 0x00002000, vvS_PATTERN_LOADED = 0x00008000,

vvS_BUFFER_FULL = 0x00010000, vvS_UNUSED_2 = 0x00020000, vvS_UNUSED_3 = 0x00040000, vvS_XDOC_LOADED = 0x00080000, vvS_PAGE_ANALYZED = 0x00200000, vvS_ANALYZING_PAGE = 0x00400000, vvS_PAGE_LAID_OUT = 0x00800000, vvS_CHARSET_TABLE_LOADED = 0x02000000,

vvS_ANY = 0x00ffffff, vvS_ERROR = 0x01000000, vvS_BLOCKED = (vvS_AWAITING_DRAW | vvS_AWAITING_VERIFIER), vvS_READY = (vvS_ACQUISITION_READY | vvS_RECOGNITION_READY), vvS_RECOGNITION = (vvS_RECOGNITION_READY | vvS_RECOGNIZING), vvS_ACQUISITION = (vvS_ACQUISITION_READY | vvS_ACQUIRING), vvS_PROCESSING = (vvS_ACQUIRING | vvS_PREPROCESS | vvS_RECOGNIZING | vvS_REGION_MAPPING | vvS_TEXT_OUTPUT) }

The internal engine state bit field values.

- enum vvxtrHint { vvHintLocalFilesystem, **vvNumberOfHints** }

  Hints to improve engine performance.

- enum vvxtrEngineVariables {

  dm_min_point, dm_max_point, dm_accept_thresh, dm_quest_thresh, dm_recmode, dm_recomp, dm_no_sloppy_manual, dm_no_hdr_ftr,

  dm_user_specified_order, dm_user_specified_regions, dm_find_headlines, dm_text_out_newline, dm_xdc_wconf, dm_xdc_cconf, dm_xdc_wbox, dm_xdc_cbox,

  dm_xdc_wbox_pixels, dm_metric, dm_pdf_format, dm_pdf_imgthresh, dm_pdf_img_nodict, dm_pdf_img_alphanum, dm_ls_quote, dm_rs_quote,

  dm_ld_quote, dm_rd_quote, dm_document_name, dm_questionable, dm_unrecognized, dm_force_single_col, dm_one_line_table_cells, dm_improved_single_col_detect,

  dm_region_type, dm_region_subtype, dm_region_stacking, dm_region_grammar_mode, dm_region_lexical_constraint_id, dm_region_lexmode, dm_region_foreground, dm_region_out_order,

  dm_region_name, dm_region_frame_left, dm_region_frame_right, dm_region_frame_top, dm_region_frame_bot, dm_region_uor_string, dm_region_uor_count, dm_black_threshold,

  dm_in_xres, dm_in_yres, dm_language, dm_english_chars, dm_char_set, dm_word_lexicon_id, dm_format_analysis, dm_double_dimension,

  dm_current_region, dm_focus_area, dm_pp_remove_halftone, dm_pp_auto_segment, dm_pp_rotate, dm_pp_fax_filter, dm_pp_auto_orient, dm_pp_invert,

  dm_pp_newspaper_filter, dm_pp_deskew, dm_pp_photometric_interp, dm_pp_dotmatrix_filter, dm_pp_autosetdegrade, dm_pp_recognition, dm_pp_segment_lineart, dm_pp_reverse_video,

  dm_pp_analyze_layout, dm_auto_flip, dm_in_filename, dm_in_curr_page, dm_in_num_pages, dm_in_format, dm_region_ids, dm_region_ids_text,

  dm_region_ids_image, dm_out_text_format, dm_out_graphics_format, dm_doc_memory_size, dm_subimage_memory_size, **dm_coord_space**, dm_output_img_source, dm_recognize_timeout,

  dm_preprocess_timeout, **VVXTR_ENGINE_VARIABLE_COUNT** }

  Data values.

Functions

- vvEngAPI * vvxtrCreateLocalEngine ()

  Create a statically-linked (local) instance of the library.

- vvEngAPI * vvxtrCreateRemoteEngine (const char *host)

Creates an OCR engine for the calling client program.

6.1.1 Define Documentation

6.1.1.1 #define VVXTR_KILL_PROCESS -1000

Time for the daemon to die.

No meaning in a statically linked library.

Definition at line 493 of file vvxtrDefs.h.

6.1.2 Typedef Documentation

6.1.2.1 typedef signed long int vvxtr_state

Representation of the state as a bit-string.

State is defined as a bitwise OR of all current conditions:

state =

C_READY | C_OCRSESOPEN | C_OUTDOCOPEN | C_FILEOPEN

C_IMAGELOADED | C_PREPROCDONE | C_RECOGDONE | C_DOCREADY

C_SUBIMAGEREADY | C_ENGINEBUSY | C_ERROR

For a state diagram please see the section Engine State and API call sequencing.

Definition at line 53 of file vvxtrDefs.h.

6.1.3 Enumeration Type Documentation

6.1.3.1 enum vvInternalEngineState

The internal engine state bit field values.

These values are used to construct the engine state returned from vvEngAPI::vvGetStatus, when dm_engine_state is passed as the first parameter. You may decipher the returned engine state by referring to this list.

This internal engine state is not often used by client programs, offering more detailed, internal information than is typically useful. More often a client program finds the vvxtr_state useful.

Enumeration values:

vvS_NONE no required state

vvS_IDLE waiting for startup

vvS_SESSION processing a document

vvS_CANCEL cancelling recognition

vvS_PGCAN cancelling text output

vvS_ACQUISITION_READY ready to acquire an image

vvS_ACQUIRING getting an image

vvS_RECOGNITION_READY ready to do recognition (also true during recognition)

vvS_RECOGNIZING doing recognition
vvS_PREPROCESS doing image preprocessing
vvS_TEXT_OUTPUT doing text formatting and output
vvS_REGION_MAPPING (internal) creating manual image regions
vvS_SHAPES_LOADED shapes loaded
vvS_AWAITING_DRAW waiting for user to get draw
vvS_AWAITING_VERIFIER waiting for user to call verfier
vvS_PATTERN_LOADED pattern loaded
vvS_BUFFER_FULL a complete image is in the buffer
vvS_UNUSED_2 not used
vvS_UNUSED_3 not used
vvS_XDOC_LOADED XDOC loaded.
vvS_PAGE_ANALYZED page analyzed
vvS_ANALYZING_PAGE analyzing page
vvS_PAGE_LAID_OUT page analysis has been run on the current page
vvS_CHARSET_TABLE_LOADED character set table loaded
vvS_ANY any state is ok
vvS_ERROR a fatal error condition exists
vvS_BLOCKED blocked
vvS_READY ready
vvS_RECOGNITION recognition in progress
vvS_ACQUISITION acquisition in progress
vvS_PROCESSING currently processing

Definition at line 443 of file vvxtrDefs.h.

6.1.3.2 enum vvxtr_dataType

Generic XTR datatype.

Enumeration values:
vvxtr_none none
vvxtr_int integer
vvxtr_string string
vvxtr_pib Vividata platform independent bitmap.
vvxtr_ptr pointer

Definition at line 32 of file vvxtrDefs.h.

6.1.3.3 enum vvxtrActionEnum

Enumeration values:
xtract_init_instance Initialize an instance of the engine.
xtract_end_instance End an instance of the engine.

Definition at line 34 of file vvxtrComm.h.

6.1.3.4 enum vvxtrCommandKeyEnum

Enumeration values:

query_status Query the status of the engine.

get_value Get a value from the engine.

Queries the engine for a specific value.

See also:

vvxtrStatusEnum

set_value Set a value in the engine.

do_action Execute an engine action.

See also:

vvxtrActionEnum

Definition at line 25 of file vvxtrComm.h.

6.1.3.5 enum vvxtrDisplayAlignmentEnum

Bit alignment of display.

- Used for ::dm_display_alignment
- Corresponds to directiv.h: <M_ALIGN8|M_ALIGN16|M_ALIGN32|M_ALIGN64>

Enumeration values:

vvAlign8 8 bit alignment

vvAlign16 16 bit alignment

vvAlign32 32 bit alignment

vvAlign64 64 bit alignment

Definition at line 364 of file vvxtrDefs.h.

6.1.3.6 enum vvxtrEngineVariables

Data values.

- Used for calls to vvEngAPI::vvSetValue and vvEngAPI::vvGetValue.

Enumeration values:

dm_min_point Minimum point size the OCR engine will recognize.

- Type: int
- Range: 5-72
- Default: 5

dm_max_point Maximum point size the OCR engine will recognize.

- Type: int
- Range: 5-72
- Default: 72

dm_accept_thresh Acceptability threshold.

The threshold above which a character is always considered acceptable. (XDOC output only)

When xdc_cconf is set to vvYes, XDOC output will include confidence value information for characters when their confidence values are between the questionability and acceptability thresholds.

This value should be set before calling vvEngAPI::vvRecognize.

More information on the XDOC output format is available in the core12xdc.pdf and kdoctxt.h documents, included with the distribution in /opt/Vividata/doc or available upon request.

- Type: int
- Range: 0-999
- Default: 999

See also:

dm_quest_thresh
dm_xdc_wconf
dm_xdc_cconf

dm_quest_thresh Questionability threshold.

For XDOC output, when the confidence value for a character is below the questionability threshold, then the questionable character mark is printed in the XDOC output file.

When xdc_cconf is set to vvYes, XDOC output will include confidence value information for characters when their confidence values are between the questionability and acceptability thresholds.

This value should be set before calling vvEngAPI::vvRecognize.

More information on the XDOC output format is available in the core12xdc.pdf and kdoctxt.h documents, included with the distribution in /opt/Vividata/doc or available upon request.

- Type: int
- Range: -1-999
- Default: 0

See also:

dm_accept_thresh
dm_xdc_wconf
dm_xdc_cconf

dm_recmode Recognition mode.

- Type: vvxtrRecModeEnum
- Default: vvRecModeUnspecified

dm_recomp Page recomposition enabled.

Page recomposition (dm_recomp) alters the way some things are written to an XDOC output file. Page recomposition is ONLY applicable to the XDOC output format. When page recomposition is on, captions, headers, and footers are output before the body text of the document. In addition, the structures of the pages will be included in the XDOC output. If page recomposition is off, then headers and footers are output at the end.

- Type: <vvNo|vvYes>
- Default:

dm_no_sloppy_manual Turn on merging of manual text regions.

- Type: <vvNo|vvYes>
- Default:

dm_no_hdr_ftr Output headers and footers as text.

- Type: <vvNo|vvYes>
- Default:

dm_user_specified_order User has specified the read order.

This value is used when automatic page segmentation was not used and page layout analysis is run. In this case, if dm_user_specified_order is on, then page layout analysis will not re-order the regions. Please read the documentation on dm_pp_analyze_layout for more explanation.

Should be set before calling vvEngAPI::vvPreprocess.

- Type: <vvNo|vvYes>
- Default: vvYes

See also:

dm_user_specified_regions

dm_user_specified_regions User has specified the regions.

This value is used when automatic page segmentation was not used and page layout analysis is run. In this case, if ::dm_user_specified_region is on, then page layout analysis will not re-form the regions. Please read the documentation on dm_pp_analyze_layout for more explanation.

Should be set before calling vvEngAPI::vvPreprocess.

- Type: <vvNo|vvYes>
- Default: vvYes

See also:

dm_pp_analyze_layout
dm_user_specified_order

dm_find_headlines Find headlines + out in xdoc.

- Type: <vvNo|vvYes>
- Default:

dm_text_out_newline Newline designation for output text.

- Type: vvxtrTextOutNewlineEnum
- Default: vvTextOutNewlineUnix

dm_xdc_wconf Enable word confidence output in XDOC format.

Word confidence values will range from 0 to 999.

Only has an effect when dm_out_text_format is vvTextFormatXdoc, vvTextFormatXdoclite, or vvTextFormatXdocplus.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_xdc_cconf Enable character confidence output in XDOC output format.

Character confidence values will range from 0 to 999.

Only has an effect when dm_out_text_format is vvTextFormatXdoc, vvTextFormatXdoclite, or vvTextFormatXdocplus.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_xdc_wbox Output word bounding boxes in XDOC output format.

Only has an effect when dm_out_text_format is vvTextFormatXdoc, vvTextFormatXdoclite, or vvTextFormatXdocplus.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_xdc_cbox Output character bounding boxes in XDOC output format.

Only has an effect when dm_out_text_format is vvTextFormatXdoc, vvTextFormatXdoclite, or vvTextFormatXdocplus.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_xdc_wbox_pixels Use pixel values for word bounding boxes in XDOC output format.

This should be used in conjunction with dm_xdc_wbox; turning on dm_xdc_wbox_pixels affects the word bounding box measurements but does not turn on the word bounding boxes.

Only has an effect when dm_out_text_format is vvTextFormatXdoc, vvTextFormatXdoclite, or vvTextFormatXdocplus.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_metric Use metric measurements.

Metric measurements are used for the output document, otherwise English measurements are used.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_pdf_format Type of PDF output text document.

Only for use when dm_out_text_format is vvTextFormatPdf. Should be set before calling vvEngAPI::vvRecognize.

- Type: vvxtrPDFFormatEnum
- Default: vvPDFFormatNormal

See also:

dm_output_img_source for information on how the graphics in the output PDF file will be compressed.
dm_pdf_imgthresh
dm_pdf_img_nodict
dm_pdf_img_alphanum

dm_pdf_imgthresh Word threshold for imagette output.

Only for use when dm_out_text_format is vvTextFormatPdf. Should be set before calling vvEngAPI::vvRecognize.

Not supported in the current release.

- Type: int
- Range: 0-100?
- Default: 0

dm_pdf_img_nodict Output imagette if word is not in the dictionary.

Only for use when dm_out_text_format is vvTextFormatPdf. Should be set before calling vvEngAPI::vvRecognize.

Not supported in the current release.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_pdf_img_alphanum Output imagette if word is alphanumeric.

Only for use when dm_out_text_format is vvTextFormatPdf. Should be set before calling vvEngAPI::vvRecognize.

Not supported in the current release.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_ls_quote Left single quote.

Should be set before call to vvEngAPI::vvRecognize.

- Type: char
- Default: ‘

dm_rs_quote Right single quote.

Should be set before call to vvEngAPI::vvRecognize.

- Type: char
- Default: ’

dm_ld_quote Left double quote.

Should be set before call to vvEngAPI::vvRecognize.

- Type: char
- Default: ”

dm_rd_quote Right double quote.

Should be set before call to vvEngAPI::vvRecognize.

- Type: char
- Default: ”

dm_document_name Document name (read-only).

- Type: string
- Default:

dm_questionable Character to be inserted before each uncertain character.

- Type: char
- Default: (none)

dm_unrecognized Character to be used instead of each unrecognized character.

- Type: char
- Default: ∼ (ASCII 126)

dm_force_single_col Force a single column.

When turned on, during region segmentation, multiple columns are merged into one single column.

This value must be set before the call to vvEngAPI::vvPreprocess, since it affects the region segmentation.

The effect of turning dm_force_single_col on is most apparent in a document with multiple columns and text output. When dm_force_single_col is off, each column is listed sequentially in the output text file, because the engine determined the columns are logically separate; when dm_force_single_col is on, the columns appear from the left to the right, giving the text file a similar appearance to the input image.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_one_line_table_cells Force one line table cells.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_improved_single_col_detect Improved single column detection.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_region_type Region type.

- Type: vvxtrRegionTypeEnum
- Default: automatically set by the engine

dm_region_subtype Region sub-type.

- Type: vvxtrRegionSubtypeEnum
- Default: automatically set by the engine

dm_region_stacking Region stacking order.

Modify this to change how regions overlap each other.

- Type: int
- Range: >= 0
- Default: automatically set

dm_region_grammar_mode Grammar mode.

- Type: vvxtrRegionGrammarModeEnum
- Default:

dm_region_lexical_constraint_id Lexical constraint id.

- Type: int
- Range:
- Default:

dm_region_lexmode Region lexical mode.

- Type: vvxtrRegionLexmodeEnum
- Default:

dm_region_foreground Photometric interpretation of region.

- Type: vvxtrRegionForegroundEnum
- Default: set automatically

dm_region_out_order Region output order.

Modify this to change the order in which the regions are written to the output.

- Type: int
- Range: >= 0
- Default: set automatically

dm_region_name Region name.

- Type: string
- Default:

dm_region_frame_left Left frame boundary of the current region, as specified by dm_current_region.

This value is calculated automatically after preprocessing and recognition. The area of a region may be composed of many rectangles, called a "union of rectangles" (UOR), with the result that the region is not rectangular in shape (see dm_region_uor_string). dm_region_frame_left is the coordinate of the left side of the leftmost rectangle in the region's UOR.

- Type: int
- Range: 0-2400

- Default: set automatically

dm_region_frame_right Right frame boundary of the current region, as specified by dm_current_region.

This value is calculated automatically after preprocessing and recognition. dm_region_frame_right is the coordinate of the right side of the rightmost rectangle in the region's UOR.

- Type: int
- Range: > 0
- Default: set automatically

dm_region_frame_top Top frame boundary of the current region, as specified by dm_current_region.

This value is calculated automatically after preprocessing and recognition. dm_region_frame_top is the coordinate of the top side of the uppermost rectangle in the region's UOR.

- Type: int
- Range: > 0
- Default: set automatically

dm_region_frame_bot Bottom frame boundary of the current region, as specified by dm_current_region.

This value is calculated automatically after preprocessing and recognition. dm_region_frame_bot is the coordinate of the lower side of the bottommost rectangle in the region's UOR.

- Type: int
- Range: > 0
- Default: set automatically

dm_region_uor_string Defines the area covered by a region through a list of rectangles (see the glossary entry on UOR).

The boundary of a region is set by setting:

- dm_current_region
- dm_region_uor_string
- dm_region_uor_count

Then by calling vvEngAPI::vvSetRegionProperties.

Please see the Frequently Asked Questions section of the main documentation page for an example of setting up a new region's boundaries.

One may also query the engine for the UOR string of the current region: Set dm_current_region, then call vvEngAPI::vvGetValue, passing dm_region_uor_string as the value.

- Type: string
- Format: x1,y1,x2,y2;x3,y3,x4,y4;x5,y5,x6,y6....
  (Rectangle coordinates are separated by commas; rectangles are separated by semicolons.)
- Default: NULL string (not a valid setting for a region)

dm_region_uor_count The number of rectangles in the current region's UOR string.

When setting the dm_region_uor_string for a region, it is critical to correctly set the dm_region_uor_count for that region, otherwise, the dm_region_uor_string will not be interpreted correctly.

The engine may be queried for this value using vvEngAPI::vvGetValue to find out the number of rectangles in the UOR of the current region, specified by dm_current_region.

- Type: int
- Range: > 0
- Default: none

dm_black_threshold Threshold to binarize multi-bit input image data.

This is the threshold used to determine which pixels are black and which are white on a page when converting an image from multiple bits per pixel to the 1-bit per pixel image processed by the OCR Engine itself.

Additional black_threshold options are provided for more sophisticated translations to a bi-tonal image. These include the value 101 which forces a random threshold to be used and the value 102 which directs the OCR Shop XTR to use the Floyd-Steinberg algorithm to determine which pixels are white and which are black.

Adjusting the black_threshold value can significantly affect the OCR Enginehs recognition of image regions.

- Type: int
- Range: 0-102
- Default: 60

dm_in_xres Input image x resolution.

Input image resolution. This overrides the resolution specified in the file itself.

The minimum resolution recognized by the OCR Engine is 70 dpi and the maximum is 900 dpi.

- Type: int
- Default: 300dpi, unless the input image itself specifies its resolution

dm_in_yres Input image y resolution.

Input image resolution. This overrides the resolution specified in the file itself.

The minimum resolution recognized by the OCR Engine is 70 dpi and the maximum is 900 dpi.

- Type: int
- Default: 300dpi, unless the input image itself specifies its resolution

dm_language Language pack(s) to load.

The parameter is a string with a comma-separated list of language pack names. (see below)

If multiple languages are specified in the dm_language parameter then they all must use the same set of shapes (same Code Page).

To recognize languages that use different sets of shapes on the same page, regions need to have their language specified separately using vvEngAPI::vvSetLexicalConstraints.∗

Note: Some languages produce output which is incompatible with some output formats. For example Russian cannot be represented by ASCII text.

Languages with dictionaries: `czech`, `danish`, `dutch`, `english`, `finnish`, `french`, `german`, `greek`, `hungar` (for Hungarian), `italian`, `norsk`, `polish`, `port` (for Portuguese), `russian`, `spanish`, `swedish`, `turkish`

Languages without dictionaries: `romanian`, `estonian`, `afrikaans`, `albanian`, `aymara`, `basque`, `breton`, `bulgarian`, `byelorussian`, `croatian`, `faroese`, `flemish`, `friulian`, `gaelic`, `galician`, `greenlandic`, `hawaiian`, `icelandic`, `indonesian`, `kurdishlat`, `latin`, `latvian`, `lithuanian`, `sorbianl`, `macedonianc`, `malaysian`, `piginenglish`, `serbian`, `ukranian`, `catalan`, `sbcroatian`, `slovak`, `slovenian`, `swahili`, `tahitian`, `sorbianu`, `welsh`, `frisianw`, `zulu`

Languages by Shape Pack/Code Page:

- Baltic (1257): Estonian, Latvian, Hawaiian, Lithuanian
- Central Europe (1250): Albanian, Polish, Croatian, Romanian, Slovenian, Czech, Serbo-Croatian, Sorbian - Lower, Hungarian, Slovak, Sorbian - Upper
- Cyrillic (1251): Bulgarian, Macedonian (Cyrillic), Serbian, Byelorussian, Russian, Ukranian

- Greek (1253): Greek
- Latin I (1252): Afrikaans, French, Malaysian, Aymara, Frisian - West, Norwegian, Basque, Friulian, Pigin English, Breton, Gaelic, Portugese, Catalan, Galician, Spanish, Danish, German, Swahili, Dutch, Greenlandic, Swedish, English, Icelandic, Tahitian, Faroese, Indonesian, Welsh, Finnish, Italian, Zulu, Flemish, Latin
- Turkish (1254): Kurdish (Latin), Turkish

Also see the Technical Specifications documentation.

- Type: string
- Default: english

See also:
vvEngAPI::vvStartOCRSes

dm_english_chars Include the english character set.

For use with character sets other than Latin 1. This allows recognition of Latin characters mixed in with a document containing primarily words written in another language and character set.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_char_set Lexical constraint character set.

If set, the OCR engine will only recognize the characters specified in the dm_char_set string. If the document being recognized only contains a few characters specified by this value, the resulting text file will contain characters from the dm_char_set along with the reject_char character for any characters not in the dm_char_set.

Setting to the null string will cause the OCR Engine to not be constrained.

dm_char_set is case-sensitive. For instance, to recognize both uppercase and lowercase d, both must be specified. Because ”s” and ”5” have similar character shapes, if ”s” (or ”S”) is specified than the number 5 will also be recognized in the output, even if it is not specified.

Also see the character set documentation.

- Type: string
- Default: not set

dm_word_lexicon_id ID number for the word lexicon.

Should be set before a call to vvEngAPI::vvSetLexicalConstraints. Corresponds to the constraintId passed to vvEngAPI::vvAddWordToLexicon.

- Type: int
- Range: 0-?
- Default: 0

More detailed information:

The dm_word_lexicon_id is used in the vvEngAPI::vvSetLexicalConstraints call, and it affects the recognition portion of the processing.

The dm_word_lexicon_id is the id number for the word lexicon, which identifies a group of words as one lexicon. Having multiple lexicon ids permits you to create several lexicons (groups of words), and then set individual regions or pages to use different word lexicons.

The process of using dm_word_lexicon_id is:

1. Create a word lexicon by calling the function vvEngAPI::vvAddWordToLexicon once for each word in the lexicon, passing: the word to add, the lexicon’s id number.
2. Set dm_word_lexicon_id to the id number for the lexicon you just created.
3. Set the focus area (region or page) and for a region, set the current region id number.

4. Call vvEngAPI::vvSetLexicalConstraints to associate the region or page with the word lexicon you just set up. Note that vvSetLexicalConstraints is used to set other lexical properties as well.

dm_format_analysis Analyze document format to determine layout of output document.

(Note: Turn format analysis off for correct PDF output.)

- Type: <vvNo|vvYes>
- Default: vvYes

dm_double_dimension Double non-square dimensions.

This value is used for non-square images (e.g., faxes which were transmitted at 200x100 dots per inch). When set, the dimensions of each image will be examined and pixels will be doubled in the dimension with the lower resolution.

If the image is already square (i.e., x-dpi equals y-dpi), no doubling is performed.

Should be set prior to calling

Note: this option is an undocumented feature in the ScanSoft engine.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_current_region Current region.

This is the region the engine is currently focused on. All region-specific settings and actions will affect the region specified by this value.

- Type: int
- Range: depends on the region ids in the current document
- Default: REGION_DEFAULTREGION

dm_focus_area Area on which to focus.

All actions and settings will affect the area set here.

- Type: vvxtrFocusAreaEnum
- Default: vvFocusAreaPage

dm_pp_remove_halftone Preprocessing option to remove image regions from output.

When set to vvYes, the OCR Engine removes all image regions from output including halftone and line art regions.

Should be set before calling vvEngAPI::vvPreprocess.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_pp_auto_segment Preprocessing option to perform auto segmentation.

When set, the OCR Engine performs the analysis to divide the different image and text areas of the document. These areas are referred to as ”regions”.

Should be set before calling vvEngAPI::vvPreprocess.

- Type: <vvNo|vvYes>
- Default: vvYes

See also:

dm_pp_analyze_layout
dm_current_region
vvEngAPI::vvSetRegionProperties

dm_pp_rotate Preprocessing option to perform a specific rotation.

Explicitly rotate the input image during pre-processing. Non-orthogonal rotation by an arbitrary angle is not supported by this value.

If dm_pp_auto_orient is set to vvCorrect and rotate is set to a non-zero value, the image will be rotated so that it is upright before recognition, even if this differs from the rotate value specified. The only effect of specifying both values is that the OCR Engine will favor the suggested rotation of the rotate value in determining the orientation of the page.

Should be set before calling vvEngAPI::vvPreprocess.

- Type: int
- Range: <0|90|180|270>
- Default: 0

dm_pp_fax_filter Preprocessing option to use the fax filter.

If the fax filter is set to vvAuto, the fax filter is only applied when needed.

Should be set before calling vvEngAPI::vvPreprocess.

- Type: <vvNo|vvYes|vvAuto>
- Default: vvNo

dm_pp_auto_orient Preprocessing option to orient the input document automatically.

When this is set to vvCorrect then the image is automatically rotated to correct its orientation. See the note for the dm_pp_rotate variable above.

Should be set before calling vvEngAPI::vvPreprocess.

- Type: <vvNo|vvCorrect|vvDetect>
- Default: vvNo

See also:

dm_auto_flip

dm_pp_invert Preprocessing option to invert the image data.

When set, the 1-bit per pixel image is inverted, so that white becomes black and black becomes white for use of the image in the recognition process.

This value affects the recognition process; it does not affect output of individual graphics elements.

Should be set before calling vvEngAPI::vvPreprocess.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_pp_newspaper_filter Preprocessing option to apply newspaper filters.

Should be set before calling vvEngAPI::vvPreprocess.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_pp_deskew Preprocessing option to automatically deskew image data.

Should be set before calling vvEngAPI::vvPreprocess.

- Type: <vvNo|vvCorrect|vvDetect>
- Default: vvNo

dm_pp_photometric_interp Preprocessing option to detect and/or correct the photometric interpretation.

When set, photometric interpretation is performed, i.e., the OCR Engine determines if the input document is overall reverse or normal video. If the document is mostly reverse video, then the OCR Engine will invert the entire image.

Should be set before calling vvEngAPI::vvPreprocess.

- Type: <vvNo|vvCorrect|vvDetect>
- Default: vvNo

dm_pp_dotmatrix_filter Preprocessing option to apply the dotmatrix filter.

Used to improve recognition for documents printed by a dot-matrix printer. When set to vvAuto, the dotmatrix filter is only applied if needed.

Should be set before calling vvEngAPI::vvPreprocess.

- Type: <vvNo|vvYes|vvAuto>
- Default: vvNo

dm_pp_autosetdegrade Preprocessing option to turn on degraded image processing.

This value affects the preprocessing portion of the program, and should be set before the call to vvEngAPI::vvPreprocess.

Setting dm_pp_autosetdegrade to vvYes turns on a preprocessing mode used for particularly degraded images, and will affect how the image is processed, and as a result the recognition results.

Should be set before calling vvEngAPI::vvPreprocess.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_pp_recognition Preprocessing recognition flag.

This flag is not required to recognize an image.

Should be set before calling vvEngAPI::vvPreprocess.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_pp_segment_lineart Preprocessing option to detect lineart regions and add them to the halftone mask, detecting it as an image instead of as text.

This will only have an effect if dm_pp_auto_segment is set to vvYes or the dm_pp_remove_halftone is set to vvYes.

Should be set before calling vvEngAPI::vvPreprocess.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_pp_reverse_video Preprocessing option to detect reverse video regions.

When set to vvYes, this value detects which regions of the page are reverse video so that the OCR Engine will know to invert the image before recognition. This option will affect the output on pages where reverse-video text exists and is detected.

Text will not be recognized in reverse video regions unless this option is set.

Should be set before calling vvEngAPI::vvPreprocess.

- Type: <vvNo|vvYes>
- Default: vvNo

dm_pp_analyze_layout Preprocessing option to analyze the page layout.

Page layout analysis (dm_pp_analyze_layout) is an intentionally flexible option in order to allow a complicated matrix of different desired features in a product that supports both automatic segmentation and manual segmentation.

Page layout analysis can be thought of as part of automatic page segmentation. Most of it runs before recognition (vvEngAPI::vvRecognize) and in conjunction with other preprocessing functions.

Page layout analysis tries to understand the layout of text on the page, the reading order, the structure of the tables, column structure, captions, headers, footers, insets, etc. In addition, page

layout analysis changes the regions that were found by automatic page segmentation. However, if automatic page segmentation was not used, page layout analysis can still run; the regions will not get re-ordered or re-formed if certain other settings are set (dm_user_specified_order and dm_user_specified_regions), but they will be analyzed.

If dm_pp_auto_segment is set, then page layout analysis is automatically run (it is considered part of the functionality of automatic page segmentation) unless dm_pp_analyze_layout is specifically turned off. However, if automatic page segmentation is not run, such as when manual page segmentation is used, page layout analysis must be run explicitly if the results from it are needed later. Page layout analysis can be run alone by setting dm_pp_analyze_layout to vvYes and turning off all of the other preprocessing options before calling vvEngAPI::vvPreprocess.

This value should be set before calling vvEngAPI::vvPreprocess.

- Type: <vvNo|vvYes>
- Default: depends on other options

dm_auto_flip Option to automatically flip upside down images.

This option should be set before calling vvEngAPI::vvRecognize.

When called, it causes recognition to be run a second time, after the first time fails because the image is upside down.

- Type: <vvNo|vvYes>
- Default: vvNo

See also:

dm_pp_auto_orient

dm_in_filename Input image filename.

This is the filename of an image to read from disk. Use along with vvEngAPI::vvOpenImageFile.

- Type: string
- Default: none

dm_in_curr_page Current page of the input document.

Use only with input image files which have multiple pages. Set after an image has been opened with vvEngAPI::vvOpenImageFile, and before a page is opened with vvOpenPage.

This is not relevant for image data passed directly to the engine through the vvPutImage function.

- Type: int
- Range: 0-99999
- Default: 0

dm_in_num_pages Number of pages in the input image document (read-only).

For PDF and PostScript input, the number of pages can not be determined without cycling through each page individually. For these input filetypes, dm_in_num_pages will be set to a very large integer. If you need to cycle through all pages of a PDF or PostScript input file, you can keep trying to process the pages until the engine returns an error that the current page does not exist.

Only relevant for images read from a file.

- Type: int
- Range: 0-99999

dm_in_format Format of input image document (read-only).

Only relevant for images read from a file.

- Type: vvxtrOutGraphicsFormatEnum

dm_region_ids List of all region ids in the image currently being processed (read-only).

- Type: string

dm_region_ids_text List all region ids of text regions in the image currently being processed (read-only).

- Type: string

dm_region_ids_image List all region ids of image regions in the image currently being processed (read-only).

- Type: string

dm_out_text_format Output text format.

- Type: vvxtrOutTextFormatEnum
- Default: vvTextFormatIso

See also:
dm_output_img_source for information on how the graphics in the output PDF file will be compressed.
dm_pdf_format

dm_out_graphics_format Output graphics format.

- Type: vvxtrOutGraphicsFormatEnum
- Default: vvSubimageFormatJpeg

dm_doc_memory_size Memory sized for an output document (read-only).

Provided so that the user knows how much space to allocated before calling vvEngAPI::vvAcquireDocMemory.

- Type: int
- Range: > 0

dm_subimage_memory_size Memory sized for an output subimage (read-only).

Provided so that the user knows how much space to allocated before calling vvEngAPI::vvAcquireSubimageMemory.

- Type: int
- Range: > 0

dm_output_img_source Image source for image output.

This value is used to specify whether the output image data should be drawn from the original input image or from the processed image from the OCR engine. Both the original input image and the processed image from the engine will be corrected for orientation and skew. The processed image could in addition be filtered, depending on the preprocessing options and image properties. The processed image from the engine always has a bit depth of 1.

For PDF output, the output file will use Flate compression for 1-bit input images and JPEG compression for 8 and 24-bit images when the input image is used for the output graphics. If the processed image is used for the output graphics in the PDF file, then CCITT 4 fax compression is used; note that this compression is not supported by all PDF readers and is not recommended for PDF output.

Used during calls to:

vvEngAPI::vvSpoolDoc (For vvTextFormatPdf and all HTML output formats)

vvEngAPI::vvCaptureSubimage

- Type: vvxtrImageSpecificationEnum
- Default: vvInputImage

dm_recognize_timeout A timeout for vvEngAPI::vvRecognize.

When this timeout is exceeded, the daemon will assume something is wrong and shut itself down immediately.

This timeout is intended as a last-resort action, to prevent the system from freezing or running out of resources if the OCR engine encounters a fatal problem. This is not a ”nice” timeout, because the engine state is lost and the client side program will only know that it can no longer communicate with the engine.

We recommend setting this timeout to be a high number, because you do not want it to be triggered while the engine is processing normally. Some degraded or complicated images do take a long time to process and should be permitted sufficient time.

- Type: int (seconds)
- Default: 0

dm_preprocess_timeout A timeout for vvEngAPI::vvPreprocess.

When this timeout is exceeded, the daemon will assume something is wrong and shut itself down immediately.

This timeout is intended as a last-resort action, to prevent the system from freezing or running out of resources if the OCR engine encounters a fatal problem. This is not a ”nice” timeout, because the engine state is lost and the client side program will only know that it can no longer communicate with the engine.

We recommend setting this timeout to be a high number, because you do not want it to be triggered while the engine is processing normally. Some degraded or complicated images do take a long time to process and should be permitted sufficient time.

- Type: int (seconds)
- Default: 0

Definition at line 589 of file vvxtrDefs.h.

6.1.3.7 enum vvxtrErrorCodes

Time for the daemon to die.

No meaning in a statically linked library.

Enumeration values:

VVXTR_ERR_INVALID_SYNTAX invalid syntax

VVXTR_ERR_NO_INPUT no input

VVXTR_ERR_INVALID_STATE invalid state

VVXTR_ERR_INITINSTANCE_FAILURE initialization of instance failed

VVXTR_ERR_ENDINSTANCE_FAILURE ending instance failed

VVXTR_ERR_NO_INPUT_FILE_SPECIFIED no input file was specified

VVXTR_ERR_UNABLE_TO_LOAD_IMAGE_FILE unable to load image data from the file

VVXTR_ERR_IMG_ACQ_FAILED image acquisition failed

VVXTR_ERR_START_SESSION_FAILED starting OCR session failed

VVXTR_ERR_RECOGNITION_FAILED recognition failed

VVXTR_ERR_OPEN_CHAR_SET_FAILURE unable to open character set

VVXTR_ERR_INVALID_SHAPE_PACK_PATH invalid shape pack path

VVXTR_ERR_BAD_LANGUAGE bad language pack

VVXTR_ERR_OPENING_LANG_PACK unable to open language pack
VVXTR_ERR_LOAD_LANG_FAILED unable to load language
VVXTR_ERR_CREATE_LANG_GROUP unable to create a language group
VVXTR_ERR_LANG_NOT_LICENSED language is not licensed; removing all languages
VVXTR_ERR_SET_DEF_LEX_CONSTRAINTS unable to set default lexical constraints
VVXTR_ERR_WRITE_OUTPUT_FAILED unable to write output
VVXTR_ERR_OPENING_OUTPUT_FILE unable to open output file
VVXTR_ERR_REGION_ECLIPSED region eclipsed
VVXTR_ERR_COULD_NOT_CONVERT_DEPTH unable to convert image depth
VVXTR_ERR_GETTING_IMG_REGION unable to get image region
VVXTR_ERR_END_SESSION_FAILED unable to end session
VVXTR_ERR_OUTPUT_INV_REGION error in output of inv region
VVXTR_ERR_NOT_IMPLEMENTED feature not implemented
VVXTR_ERR_INV_PDF_FORMAT invalid PDF format
VVXTR_ERR_PDF_IMG_OUTPUT error in PDF image output
VVXTR_ERR_PDF_END_OUTPUT error in finishing PDF output
VVXTR_ERR_UNABLE_TO_WRITE_PDF error in writing PDF output
VVXTR_ERR_HOST_LOOKUP_FAILED The remote name lookup failed.
VVXTR_ERR_CONNECTION_FAILED The remote connection failed.
VVXTR_ERR_TRANSFER_FAILED The data transfer failed.
VVXTR_ERR_NO_SUBIMAGE No subimage available to acquire.
VVXTR_ERR_PP_FAILED Preprocessing failed.
VVXTR_ERR_SET_OPTIONS_FAILED Set options failed.
VVXTR_ERR_REMOVE_FILE_FAILED Removal of temp file failed.
VVXTR_ERR_NO_DOC No doc available to acquire.
VVXTR_ERR_INV_SUBIMG_FORMAT Invalid subimage format.
VVXTR_ERR_SYNCHRONIZATION The client and server are out of synch.
VVXTR_ERR_NOENGINE The engine hasn't been allocated yet.
VVXTR_ERR_BUFFER_TOO_SMALL The buffer size is too small to receive data.
VVXTR_ERR_INVALID_PAGE Invalid page in input image was specified.
VVXTR_ERR_MISSING_LANG Add user lexicon missing language.
VVXTR_ERR_ENG_OUT_OF_MEMORY Internal OCR engine is out of memory.
VVXTR_ERR_INVALID_UOR_COUNT UOR count is invalid.
VVXTR_ERR_INVALID_UOR_STRING UOR string contains an invalid character.
VVXTR_ERR_READONLY_VALUE Can not write to a read-only value.
VVXTR_ERR_INVALID_REGION_ID Invalid region id.
VVXTR_ERR_SET_REGION_UNSUCCESSFUL Call to vvSetRegion was unsuccessful.
VVXTR_ERR_REGION_HANDLER Kernel error in the region handler.
VVXTR_ERR_ENGINE_KILLED Engine has been killed with vvKill; no further commands are possible.
VVXTR_ERR_NO_LICENSE No licenses are available for checkout.
VVXTR_ERR_NO_LICENSE_MANAGER License manager unavailable, and starting it was unsuccessful.
VVXTR_ERR_LICENSE_COMM_ERROR License manager communication error.
VVXTR_ERR_NO_FEATURE Feature not available.

Definition at line 498 of file vvxtrDefs.h.

6.1.3.8 enum vvxtrFocusAreaEnum

How to specify the document area on which to focus.

- Possible values of dm_focus_area.

Enumeration values:

vvFocusAreaPage Work on a page basis.

vvFocusAreaRegion Work on a region basis.

Definition at line 188 of file vvxtrDefs.h.

6.1.3.9 enum vvxtrHint

Hints to improve engine performance.

See also:

vvEngAPI::vvSetHint

Enumeration values:

vvHintLocalFilesystem The filesystem of the client is the same as the filesystem of the server.

Definition at line 571 of file vvxtrDefs.h.

6.1.3.10 enum vvxtrImageSpecificationEnum

Specify the original or processed image.

Specify the image data used for reference.

The coordinate systems and image data in the original input image and the image contained in the OCR engine may be different. The image contained in the engine is always 1-bit depth and may have been rotated or deskewed. As a result, it is important to be explicit about which image is used for reference when dealing with coordinates and the source for generating image output.

Used with dm_output_img_source.

Enumeration values:

vvInputImage Refer to original input image.

vvProcessedImage Refer to image data currently in the engine, which has been processed.

Definition at line 398 of file vvxtrDefs.h.

6.1.3.11 enum vvxtrLexicalConstraintModeEnum

Lexical constraint mode.

- Specifies whether lexical constraints should be added or replaced.
- Corresponds to directiv.h: CONSTRAINT_MODE_*

Enumeration values:

vvLexConstraintModeReplace Replace lexical constraints.

vvLexConstraintModeAdd Add lexical constraints.

Definition at line 353 of file vvxtrDefs.h.

6.1.3.12 enum vvxtrOutGraphicsFormatEnum

Output graphics formats.

- Possible values of dm_out_graphics_format.
- Correspond to Vividata's output formats.
- All can be used with vvEngAPI::vvAcquireSubimageFile and vvEngAPI::vvAcquireSubimageMemory, except vvEngAPI::vvSubimageFormatVvxtrImage, which can only be used with output via vvEngAPI::vvAcquireSubimageMemory.

Enumeration values:

vvSubimageFormatNone Not for use.

vvSubimageFormatTiff TIFF: single and multipage formats; group 3, group 4, lzw, or jpeg compression; supports 1, 8, or 24-bit depths.

vvSubimageFormatRas Rasterfile: native Sun bitmap; supports 1, 8, or 24-bit depths.

vvSubimageFormatEpsf Encapsulated PostScript: a PostScript document that contains only an image; levels 1,2,3.

vvSubimageFormatX11 X11: X Consortium created format; extensions .xpm, .xbm, .bm; either a PPM or PBM.

vvSubimageFormatTiffpack TIFF pack-bits.

vvSubimageFormatTiffg31d TIFF G31d: CCITTFAX3 compression; supports 1-bit depth.

vvSubimageFormatTiffg32d TIFF G32d: CCITTFAX3 compression; 2D encoding; supports 1-bit depth.

vvSubimageFormatTiffg42d TIFF G42d: CCITTFAX4 compression; supports 1-bit depth.

vvSubimageFormatTifflzw TIFF with LZW compression (not available).

vvSubimageFormatPaltiff PAL-TIFF.

vvSubimageFormatGif Graphics Interchange Format: supports 8-bit depth.

vvSubimageFormatJpeg Joint Photographics Experts Group File Interchange Format; supports 8, 24-bit depths.

vvSubimageFormatPng PNG: RGB-encoded bitmap with a simple header; not interlaced.

vvSubimageFormatXwd X Window Dump: Stores screen dumps created by xwd client process in X-Windows.

vvSubimageFormatRgb Silicon Graphics RGB (uncompressed); supports 8, 24-bit depths.

vvSubimageFormatRgbrle Silicon Graphics RGB with run-length encoding; supports 8, 24-bit depths.

vvSubimageFormatEPdf Encapsulated PDF (image only).

vvSubimageFormatVvxtrImage vvxtrImage format: for use with memory output only; supports 1, 8, and 24-bit depths

Definition at line 131 of file vvxtrDefs.h.

6.1.3.13 enum vvxtrOutTextFormatEnum

Output text formats.

Note that the XDOC format is a ScanSoft text output format which provides detailed information about the text, images, and formatting in a recognized document. The documentation files core12xdc.pdf and kdoctext.h, included with the development seat distribution in /opt/Vividata/doc, provide enough information to parse the XDOC format for use within an outside application.

- Used for dm_out_text_format.
- Corresponds to directiv.h: TEXTFORMAT_*

See also:

dm_pdf_format
dm_xdc_wconf
dm_xdc_cconf
dm_quest_thresh
dm_xdc_wbox
dm_xdc_cbox

What is the difference between text and image PDF files?

What is the XDOC format and how do I use it?

Enumeration values:

vvTextFormatDefault Default (not intended for use).

vvTextFormatNone None (not intended for use).

vvTextFormatXdoc Enhanced XDOC.

vvTextFormatXdoclite XDOClite (no format analysis)

vvTextFormatXdocplus XDOCplus (with style sheet data)

vvTextFormatPost PostScript (not supported).

vvTextFormatIso ASCII or ISO text (default newline character is the UNIX \n)

vvTextFormatPdf pdf_def (all varieties)

vvTextFormat8bit 8bit text

vvTextFormatUnicode Unicode text (UTF-16 Unicode version 2.0; default newline character is the Mac \r)

vvTextFormatHtmlWysiwygP HTML wysiwyg with styles (not supported).

vvTextFormatHtmlWysiwygS HTML wysiwyg with styles (not supported).

vvTextFormatHtmlTable HTML with columns in tables (not supported).

vvTextFormatHtmlSimple HTML with one column (not supported).

Definition at line 105 of file vvxtrDefs.h.

6.1.3.14 enum vvxtrPDFFormatEnum

PDF format.

- Possible values of dm_pdf_format.

- Only has an effect when dm_out_text_format == PDF.
- Corresponds to directiv.h: M_PDF_FLAVOR, PDF_*

Enumeration values:

vvPDFFormatNormal PDF with text and images mixed.

vvPDFFormatText PDF with invisible text behind a full page image.

vvPDFFormatImgOnly PDF with only a full page image.

Definition at line 238 of file vvxtrDefs.h.

6.1.3.15 enum vvxtrRecModeEnum

Recognition mode.

- Possible values of dm_recmode.
- Corresponds to directiv.h: RECMODE

Enumeration values:

vvRecModeUnspecified Unspecified.

vvRecModeStandard Standard.

vvRecModeDegraded Degraded text.

Definition at line 210 of file vvxtrDefs.h.

6.1.3.16 enum vvxtrRegionAbutmentEnum

Region abutment.

- Used for ::dm_region_abutment.
- Corresponds to directiv.h: REGION_PAGE_ABUTMENT

Enumeration values:

vvRegionAbutmentUnknown unknown

vvRegionAbutmentNone none

vvRegionAbutmentLeft left

vvRegionAbutmentRight right

vvRegionAbutmentLeftAndRight left and right

Definition at line 339 of file vvxtrDefs.h.

6.1.3.17 enum vvxtrRegionForegroundEnum

Region foreground specification.

- Used for dm_region_foreground.
- Corresponds to directiv.h: FWX_FOREGROUND_COLOR

Enumeration values:

vvRegionForegroundBlack black

vvRegionForegroundWhite white

vvRegionForegroundUnknown unknown

Definition at line 316 of file vvxtrDefs.h.

6.1.3.18 enum vvxtrRegionGrammarModeEnum

Region grammar modes.

- Used for dm_region_grammar_mode.
- Corresponds to directiv.h: REGION_*

Enumeration values:

vvRegGrammarModeWord Word grammar mode.

vvRegGrammarModeLine Line grammar mode.

Definition at line 292 of file vvxtrDefs.h.

6.1.3.19 enum vvxtrRegionLexmodeEnum

Region lexical mode.

- Used for dm_region_lexmode.
- Corresponds to directiv.h: REGION_<NOLEX|PREFERENCE|ABSOLUTE>

Enumeration values:

vvRegionLexmodeNolex no lexicon, lanuguage rules apply

vvRegionLexmodePreference prefer lexical verification

vvRegionLexmodeAbsolute require lexical verification

Definition at line 303 of file vvxtrDefs.h.

6.1.3.20 enum vvxtrRegionOpacityEnum

Region opacity.

- Used for ::dm_region_opacity.
- Corresponds to directiv.h: REGION_TRANSPARENCY

Enumeration values:

vvRegionOpacityOpaque opaque

vvRegionOpacityTransparent transparent

Definition at line 328 of file vvxtrDefs.h.

6.1.3.21 enum vvxtrRegionSubtypeEnum

Region subtypes.

- Used for dm_region_subtype.
- Corresponds to directiv.h: REGION_SETTABLE_SUBTYPE

Enumeration values:

vvRegionSubtypeUnflavored unflavored

vvRegionSubtypeTable table

vvRegionSubtypeTable_inset table that is an inset

vvRegionSubtypeHeadline headline

vvRegionSubtypeTimestamp time stamp

vvRegionSubtypeLineart line art

vvRegionSubtypeHalftone half tone

vvRegionSubtypeInset inset

vvRegionSubtypeCaption caption

vvRegionSubtypePage_footer footer

vvRegionSubtypePage_header header

vvRegionSubtypeVruling vertical ruling

vvRegionSubtypeHruling horizontal ruling

vvRegionSubtypeNoise noise

vvRegionSubtypeIpcorePictureMask picture mask

Definition at line 268 of file vvxtrDefs.h.

6.1.3.22 enum vvxtrRegionTypeEnum

Region types.

- Used for dm_region_type.

- Corresponds to directiv.h: REGION_CLASS

Enumeration values:

vvRegionTypeAny any defined

vvRegionTypeIgnore ignore this region

vvRegionTypeText text region

vvRegionTypeImage image region

vvRegionTypeVrule vertical rule

vvRegionTypeHrule horizontal rule

vvRegionTypeHidden private - not used

vvRegionTypeRevid private - not used

vvRegionTypeHiddenimage private - not used

Definition at line 250 of file vvxtrDefs.h.

6.1.3.23 enum vvxtrResponseEnum

Generic settings used for many different options.

Enumeration values:

vvNo Turn feature off.

vvYes Turn feature on.

vvAuto Use feature in automatic mode (the engine will decide whether to use it).

vvDetect Detect a condition, but do not correct it.

vvCorrect Detect and correct a condition.

vvManual Manual mode.

Definition at line 74 of file vvxtrDefs.h.

6.1.3.24 enum vvxtrStatusEnum

Enumeration values:

dm_daemon_state Daemon state.

dm_engine_state Internal engine state.

dm_words_seen Number of words seen on page.

dm_words_recognized Number of words recognized on page.

dm_characters_seen Number of characters seen on page.

dm_characters_recognized Number of characters recognized on page.

dm_percentage_done Fraction of page complete.

dm_page_number Current page number.

dm_region_number Current region number.

dm_line_orientation Orientation of text lines.

dm_skew_angle Last skew angle estimate.

dm_current_skew_angle Last skew angle estimate, post-correction.

dm_skew_confidence Confidence associated with skew angle estimate.

dm_smotation_angle Angle of last small angle rotation approximation performed.

dm_text_orientation Upside-down or rightside-up (0 = orientation unknown, 1 = portrait, 2 = rightside-up, -1 = landscape, -1 = upside-down).

dm_min_is_black Color of majority of pixels.

dm_most_complex Most vertices in region.

dm_byte_order Big or little endian (0 = big endian, 1 = little endian.

dm_error_code Internal engine error code.

dm_version Internal engine version.

dm_bitmap_split Is bitmap split in two parts?

dm_line_doubled Image line or column doubled (0 = not doubled, 1 = line doubled, 2 = column doubled).

dm_regionview_mode How are regions described.

dm_codepage Codepage used.

dm_lexicon Is there a lexicon or not? (0 = no lexicon, 1 = lexicon present).

dm_most_cols Most columns in region.

dm_most_rows Most rows in region.

dm_most_cells Most cells in region.

Definition at line 43 of file vvxtrComm.h.

6.1.3.25 enum vvxtrTextOutNewlineEnum

Newline designation.

The default for vvTextFormatUnicode text output is vvTextOutNewlineMac. The default for vvTextFormatIso text output is vvTextOutNewlineUnix.

- Possible values of dm_text_out_newline.
- Corresponds to directiv.h: M_NEWLINE_*

Enumeration values:

vvTextOutNewlineUnix UNIX style line feed ('\n').

vvTextOutNewlineMac Macintosh style carriage return ('\r').

vvTextOutNewlinePC PC style line feed plus carriage return ('\n"\r').

Definition at line 225 of file vvxtrDefs.h.

6.1.3.26 enum vvxtrVerifierModeEnum

Verifier mode.

- Possible values of ::dm_verifier_mode.
- Corresponds to directiv.h: VERIFIER_MODE

Enumeration values:

vvVerifierModeWord Word verifier mode.

vvVerifierModeChar Character verifier mode.

Definition at line 199 of file vvxtrDefs.h.

6.1.3.27 enum vvxtrVersionLocationEnum

Version location.

Specifies whether to get the version number for the local communicator or remote daemon.

Used with vvEngAPI::vvGetVersion.

Enumeration values:

vvLocal Client side library version.

vvRemote Daemon (ocrxtrdaemon) version.

Definition at line 379 of file vvxtrDefs.h.

6.1.4 Function Documentation

6.1.4.1 vvEngAPI* vvxtrCreateLocalEngine ()

Create a statically-linked (local) instance of the library.

For internal use or clients under special contract only.

Requires that you link with libvvocr_engine.a.

The returned object is *not* thread-safe.

This function allocates space for a new engine and the calling function should delete the retuned engine when it is finished with it to avoid a memory leak.

Returns:

A vvEngAPI object that refers to a local engine.

6.1.4.2 vvEngAPI* vvxtrCreateRemoteEngine (const char * *host*)

Creates an OCR engine for the calling client program.

More specifically, this function creates an instance of an OCR Shop XTR/API communicator object. It connects with the running ocrxtrdaemon to generate the OCR engine. The returned OCR engine will then be used by the client program to perform all OCR functionality.

The object returned is fully thread-safe, however the engine is not in itself multithreaded and may only be used for processing one image at a time.

vvxtrCreateRemoteEngine is generally called once on start up of the client program, or multiple times if multiple engines will be used concurrently. For the most efficient usage of the API, vvxtrCreateRemote-Engine should not in general be called on a file by file basis, since multiple files are usually processed within the same instance and OCR session.

It can be a good idea delete the old engine and then create a new engine every several hundred input images, to make sure everything is cleaned up regularly.

This function allocates space for the returned vvEngAPI pointer. The calling function is responsible for this allocated memory. You should delete the engine when you are finished with it in order to shut down the engine properly and to avoid memory leaks.

Parameters:

host Host where the ocrxtrdaemon is running. Can be in the format: ”ocr.host.com:port” where port is a specific port number to connect with.

Returns:

An OCR engine object. **Destroy this object by deleting it after you are finished with it.**

Referenced by CreateEngine().

Chapter 7

OCR Shop XTR/API User Documentation Class Documentation

7.1 vvEngAPI Class Reference

Optical Character Recognition Engine API Class.

`#include <vvxtrAPI.h>`

Public Member Functions

- virtual ~vvEngAPI ()
  Destructor.
- virtual vvxtrStatus vvKill ()=0
  Kill the engine.
- virtual vvxtrStatus vvInitInstance ()=0
  Initialize OCR engine instance.
- virtual vvxtrStatus vvEndInstance ()=0
  End OCR engine instance.
- virtual vvxtrStatus vvStartOCRSes ()=0
  Start an OCR session.
- virtual vvxtrStatus vvEndOCRSes ()=0
  End an OCR session.
- virtual vvxtrStatus vvStartDoc ()=0
  Start an output document.
- vvxtrStatus vvStartDoc (int fileFormat)
  Start an output document.

- virtual vvxtrStatus vvSpoolDoc ()=0
  Spool recognized data to the output doc.
- virtual vvxtrStatus vvEndDoc ()=0
  End the output document.
- virtual vvxtrStatus vvAcquireDocMemory (void *memPtr, int bufsize)=0
  Get the document output through memory.
- virtual vvxtrStatus vvAcquireDocFile (const char *filename)=0
  Get the document output as a file.
- virtual vvxtrStatus vvCaptureSubimage ()=0
  Capture a subimage.
- vvxtrStatus vvCaptureSubimage (int regionID)
  Capture a subimage.
- virtual vvxtrStatus vvAcquireSubimageMemory (void *memPtr, int bufsize)=0
  Get the image output through memory.
- virtual vvxtrStatus vvAcquireSubimageFile (const char *filename)=0
  Write the image output to a file.
- virtual vvxtrStatus vvOpenImageFile ()=0
  Open an input image document in the OCR engine.
- vvxtrStatus vvOpenImageFile (const char *fileName)
  Open the input image file passed as a parameter.
- virtual vvxtrStatus vvCloseImageFile ()=0
  Close the input image file currently open in the OCR engine.
- virtual vvxtrStatus vvReadImageData ()=0
  Read image data into the OCR engine.
- virtual vvxtrStatus vvReadImageData (const struct vvxtrImage *img)=0
  Read image data into the OCR engine.
- virtual vvxtrStatus vvUnloadImage ()=0
  Unload the page of input image data currently loaded in the OCR engine.
- virtual vvxtrStatus vvPreprocess ()=0
  Run preprocessing on the currently loaded image page.
- virtual vvxtrStatus vvRecognize ()=0
  Run recognition on the currently loaded image page.
- virtual vvxtrStatus vvInitValues ()=0
  Initialize engine values.

- virtual vvxtrStatus vvGetStatus (const int key, vvxtr_stdarg *result)=0

  Get engine status for the passed key.

- vvxtrStatus vvSetValue (const int key, int value)

  Set the integer value of the passed key in the engine.

- vvxtrStatus vvSetValue (const int key, char *value)

  Set the string value of the passed key in the engine.

- vvxtrStatus vvGetValue (const int key, int *result)

  Get an integer engine value.

- vvxtrStatus vvGetValue (const int key, char **result)

  Get a character or string engine value.

- virtual vvxtrStatus vvGetState (vvxtr_state *resultState)=0

  Get the engine state.

- virtual vvxtrStatus vvGetCond (const vvxtr_cond c, int *result)=0

  Get the value of an engine condition.

- virtual vvxtrStatus vvSetHint (vvxtrHint hint, bool set=true)

  Give the engine more information about its environment to possibly improve performance.

- virtual bool vvGetHint (vvxtrHint hint)

  Return the current setting of one of the hints.

- virtual void vvSigHUP ()

  A function to call when your client application receives a signal of type SIGHUP.

- virtual const char * vvGetStatusString (vvxtrStatus status=VVXTR_PREVIOUS_ERROR)

  Returns a string that describes the last error condition, if any detailed information about the error is available.

- virtual vvxtrStatus vvGetVersion (vvxtrVersionLocationEnum loc, const char **ver)=0

  Return the version.

- virtual vvxtrStatus vvAddWordToLexicon (const char *word, int constraintId)=0

  Add a word to a user lexicon.

- virtual vvxtrStatus vvSetLexicalConstraints (int mode)=0

  Set lexical constraints for an area.

- virtual vvxtrStatus vvSetRegionProperties (void)=0

  Set region information.

- virtual vvxtrStatus vvForceRegionForeground (int foregroundColor)=0

  Explicitly set the foreground photometric interpretation.

- virtual vvxtrStatus vvRemoveRegion (void)=0

  Remove a region.

Public Attributes

- bool m_hints [vvNumberOfHints]

  Current hint flags.

7.1.1 Detailed Description

Optical Character Recognition Engine API Class.

This class is the interface to using the OCR Shop XTR/API.

Use of the API generally consists of:

- Making a sequence of ”action” calls.
- Setting ”values” in the engine to control the actions.
- Passing image data to the engine.
- Receiving document and subimage data from the engine.

The basic sequence of actions is:

1. Initialization:	
vvEngAPI::vvInitInstance	Initialize the OCR instance.
vvEngAPI::vvStartOCRSes	Start an OCR session, loading appropriate languages.
vvEngAPI::vvInitValues	Initialize all ”values”.
2. Input image recognition:	
vvEngAPI::vvOpenImageFile	Open an input image file (required only for file input).
vvEngAPI::vvReadImageData	Read a page of image data into the engine.
vvEngAPI::vvPreprocess	Run preprocessing.
vvEngAPI::vvRecognize	Run recognition.
vvEngAPI::vvUnloadImage	Unload the image data from the engine.
vvEngAPI::vvCloseImageFile	Close the input image file (required only for file input).
3. Output a document:	
vvEngAPI::vvStartDoc	Start an output document.
vvEngAPI::vvSpoolDoc	Send data to the output document.
vvEngAPI::vvEndDoc	Close the output document.
vvEngAPI::vvAcquireDocMemory	Download the output document to an allocated memory block.
vvEngAPI::vvAcquireDocFile	Download the output document to a file.
4. Output a subimage:	
vvEngAPI::vvCaptureSubimage	Create a subimage from the input image.
vvEngAPI::vvAcquireSubimageMemory	Download the subimage to an allocated memory block.
vvEngAPI::vvAcquireSubimageFile	Download the subimage to a file.
5. Close down the engine:	
vvEngAPI::vvEndOCRSes	End the OCR session.
vvEngAPI::vvEndInstance	End the OCR instance.
vvEngAPI::vvKill	Completely shut down the engine.

Before each action call, the appropriate values should be set in the engine to control the action’s functionality.

See also:
vvEngAPI::vvSetValue

Opening and closing of the input image and the output document may be performed asynchronously, allowing for more complex results. For example, the one output document may remain open while multiple input images are recognized and output is spooled to the output document for each one.

The actions listed above do not have to be performed in the exact order specified. Please see the detailed description on state and the individual function documentation for more information.

Other API functions exist as well:

- vvEngAPI::vvKill Shuts down the OCR engine completely; may be called regardless of current state.
- vvEngAPI::vvGetStatus Get the status from the engine.

- vvEngAPI::vvSetValue Set a value in the engine.
- vvEngAPI::vvGetValue Get a value from the engine.
- vvEngAPI::vvGetState Get the current engine state.
- vvEngAPI::vvGetCond Get the value of a specific condition in the state.
- vvEngAPI::vvGetStatusString Get the status string.

Handling values:

Values should be set before they will be used by the engine. Please see the list of engine values and the documentation in vvxtrDefs.h and the sample program.

Set values with the vvEngAPI::vvSetValue function, and read values using vvEngAPI::vvGetValue.

Values are usually strings or numbers. Strings are represented as char ∗:

- When you pass a char∗ to vvEngAPI::vvSetValue, the engine will make its own copy of the passed string.
- When you get a char∗ value with vvEngAPI::vvGetValue, the engine passes you a pointer to memory it is responsible for. Do not write to this memory or attempt to delete it. This memory is temporary and not guaranteed to remain unchanged after the next call to any API function.

Handling data:

Input and output data may always be handled through files or through passed memory blocks.

Input data:

- You may open a file to use as input with the vvEngAPI::vvOpenImageFile, then you can read any page of that file using the vvEngAPI::vvReadImageData function call.
- If you want to pass image data directly to the engine through memory, you can directly call vvEngAPI::vvReadImageData and pass the data, and you do not need to call vvEngAPI::vvOpenImageFile, since there is not a file to read.
- The input image, whether passed as a file or through memory, must always be unloaded from the engine with the vvEngAPI::vvUnloadImage function call.
- When an input file was opened, it must always be closed with the vvEngAPI::vvCloseImageFile function call.

Output data:

- The output format value dm_out_text_format must be set before an output document is started. Available output formats are listed in the vvxtrOutTextFormatEnum.
- An output document is created (vvEngAPI::vvStartDoc), written to (vvEngAPI::vvSpoolDoc), and closed (vvEngAPI::vvEndDoc) without respect to how it will be acquired (through memory or written to a file).

- In the first method for acquiring an output document, you request the engine write it to a file with the vvEngAPI::vvAcquireDocFile function call.
- In the second method for acquiring an output document, you request the output document be placed in memory by calling vvEngAPI::vvAcquireDocMemory, passing a pre-allocated block of memory. The document's size is available as soon as the output document is closed (vvEngAPI::vvEndDoc).
- Subimage output works in a similar manner, except the creation of a subimage is atomic, being started and finished in the function call vvEngAPI::vvCaptureSubimage.
- You can acquire a subimage and write it to a file using the vvEngAPI::vvAcquireSubimageFile function call.
- You can acquire a subimage in memory by calling vvEngAPI::vvAcquireSubimageMemory, passing a pre-allocated block of memory. The subimage's size is available after the subimage is created (vvEngAPI::vvCaptureSubimage).
- Both output documents and subimages are available for acquisition as long as a new output document has not been started, or a new subimage has not been captured, respectively, and while the OCR session continues.

Engine State and API call sequencing

Conditions:

Condition	Abbreviation	Value	Description
C_READY	Re	0x400	ready
C_OCRSESOPEN	Se	0x200	OCR session is open
C_OUTDOCOPEN	Ou	0x100	output document is open
C_FILEOPEN	Fi	0x080	input image is open
C_IMAGELOADED	Im	0x040	input page is open
C_PREPROCDONE	Pr	0x020	preprocessing has been run on the current page
C_RECOGDONE	Rc	0x010	recognition has been run on the current page
C_DOCREADY	Do	0x008	output document is ready for acquisition
C_SUBIMAGEREADY	Su	0x004	output subimage is ready for acquisition
C_ENGINEBUSY	En	0x002	engine is currently busy
C_ERROR	Er	0x001	error

See also:
vvxtrDefs.h

State table for actions: vvEngAPI::vvInitInstance

```
   Conditions of interest:  Re| Se| Ou|   Fi| Im| Pr| Rc|   Do| Su| En| Er
111 1111 1111 = 7FF
   Required values:        !Re|!Se|!Ou|  !Fi|!Im|!Pr|!Rc|  !Do|!Su|!En|!Er
000 0000 0000 = 000
   Change in state:         Re|---|---|  ---|---|---|---|  ---|---|---|---
```

vvEngAPI::vvEndInstance

```
   Conditions of interest:  Re|---|---|  ---|---|---|---|  ---|---| En| Er
100 0000 0011 = 403
   Required values:         Re|---|---|  ---|---|---|---|  ---|---|!En|!Er
100 0000 0000 = 400
   Change in state:        !Re|---|---|  ---|---|---|---|  ---|---|---|---
```

vvEngAPI::vvStartOCRSes

```
   Conditions of interest:  Re| Se|---|  ---|---|---|---|  ---|---| En| Er
110 0000 0011 = 603
   Required values:         Re|!Se|---|  ---|---|---|---|  ---|---|!En|!Er
100 0000 0000 = 400
   Change in state:        ---| Se|---|  ---|---|---|---|  ---|---|---|---
```

vvEngAPI::vvEndOCRSes

```
   Conditions of interest:  Re| Se|---|  ---|---|---|---|  ---|---| En| Er
110 0000 0011 = 603
   Required values:         Re| Se|---|  ---|---|---|---|  ---|---|!En|!Er
110 0000 0000 = 600
   Change in state:        ---|!Se|---|  ---|---|---|---|  !Do|---|---|---
```

vvEngAPI::vvStartDoc

```
   Conditions of interest:  Re| Se| Ou|  ---|---|---|---|  ---|---| En| Er
111 0000 0011 = 703
   Required values:         Re| Se|!Ou|  ---|---|---|---|  ---|---|!En|!Er
110 0000 0000 = 600
   Change in state:        ---|---| Ou|  ---|---|---|---|  ---|---|---|---
```

vvEngAPI::vvSpoolDoc

```
   Conditions of interest:  Re| Se| Ou|  ---| Im| Pr| Rc|  ---|---| En| Er
111 0111 0011 = 773
   Required values:         Re| Se| Ou|  ---| Im| Pr| Rc|  ---|---|!En|!Er
111 0111 0000 = 770
   Change in state:        ---|---|---|  ---|---|---|---|  ---|---|---|---
```

vvEngAPI::vvEndDoc

```
   Conditions of interest:  Re| Se| Ou|  ---|---|---|---|  ---|---| En| Er
111 0000 0011 = 703
   Required values:         Re| Se| Ou|  ---|---|---|---|  ---|---|!En|!Er
111 0000 0000 = 700
   Change in state:        ---|---|!Ou|  ---|---|---|---|   Do|---|---|---
```

vvEngAPI::vvAcquireDocMemory

```
   Conditions of interest:  Re| Se| Ou|  ---|---|---|---|   Do|---| En| Er
111 0000 1011 = 70B
   Required values:         Re| Se| Ou|  ---|---|---|---|   Do|---|!En|!Er
110 0000 1000 = 608
   Change in state:        ---|---|---|  ---|---|---|---|  ---|---|---|---
```

vvEngAPI::vvAcquireDocFile

```
   Conditions of interest:  Re| Se| Ou|  ---|---|---|---|   Do|---| En| Er
111 0000 1011 = 70B
   Required values:         Re| Se| Ou|  ---|---|---|---|   Do|---|!En|!Er  110 0000 1000 = 608
   Change in state:        ---|---|---|  ---|---|---|---|  ---|---|---|---
```

vvEngAPI::vvCaptureSubimage

```
   Conditions of interest:  Re| Se|---|  ---| Im| Pr| Rc|  ---|---| En| Er
110 0111 0011 = 673
   Required values:         Re| Se|---|  ---| Im| Pr| Rc|  ---|---|!En|!Er
110 0111 0000 = 670
   Change in state:        ---|---|---|  ---|---|---|---|  ---| Su|---|---
```

vvEngAPI::vvAcquireSubimageMemory

```
   Conditions of interest:  Re| Se|---|  ---|---|---|---|  ---| Su| En| Er
110 0000 0111 = 607
   Required values:         Re| Se|---|  ---|---|---|---|  ---| Su|!En|!Er
110 0000 0100 = 604
   Change in state:        ---|---|---|  ---|---|---|---|  ---|---|---|---
```

vvEngAPI::vvAcquireSubimageFile

```
   Conditions of interest:  Re| Se|---|  ---|---|---|---|  ---| Su| En| Er
110 0000 0111 = 607
   Required values:         Re| Se|---|  ---|---|---|---|  ---| Su|!En|!Er
110 0000 0100 = 604
   Change in state:        ---|---|---|  ---|---|---|---|  ---|---|---|---
```

vvEngAPI::vvOpenImageFile

```
   Conditions of interest:  Re| Se|---|   Fi|---|---|---|  ---|---| En| Er
110 1000 0011 = 683
   Required values:         Re| Se|---|  !Fi|---|---|---|  ---|---|!En|!Er
110 0000 0000 = 600
   Change in state:        ---|---|---|   Fi|---|---|---|  ---|---|---|---
```

vvEngAPI::vvCloseImageFile

```
   Conditions of interest:  Re| Se|---|   Fi| Im|---|---|  ---|---| En| Er
110 1100 0011 = 6C3
   Required values:         Re| Se|---|   Fi|!Im|---|---|  ---|---|!En|!Er
110 1000 0000 = 680
   Change in state:        ---|---|---|  !Fi|---|---|---|  ---|---|---|---
   Change in state:        ---|---|---|   Fi|---|---|---|  ---|---|---|---
   Change in state:        ---|---|---|   Fi|---|---|---|  ---|---|---|---
```

vvEngAPI::vvReadImageData (from file)

```
   Conditions of interest:  Re| Se|---|   Fi| Im|---|---|  ---|---| En| Er
110 1100 0011 = 6C3
   Conditions of interest:  Re| Se|---|   Fi| Im|---|---|  ---|---| En| Er
110 1100 0011 = 6C3
   Conditions of interest:  Re| Se|---|   Fi| Im|---|---|  ---|---| En| Er
110 1100 0011 = 6C3
   Required values:         Re| Se|---|   Fi|!Im|---|---|  ---|---|!En|!Er
110 1000 0000 = 680
   Change in state:        ---|---|---|  ---| Im|---|---|  ---|---|---|---
```

vvEngAPI::vvReadImageData (from memory)

```
   Conditions of interest:  Re| Se|---|  ---| Im|---|---|  ---|---| En| Er
110 0100 0011 = 643
   Required values:         Re| Se|---|  ---|!Im|---|---|  ---|---|!En|!Er
110 0000 0000 = 600
   Change in state:        ---|---|---|  ---| Im|---|---|  ---|---|---|---
```

vvEngAPI::vvUnloadImage

```
   Conditions of interest:  Re| Se|---|  ---| Im|---|---|  ---|---| En| Er
110 0100 0011 = 643
   Required values:         Re| Se|---|  ---| Im|---|---|  ---|---|!En|!Er
110 0100 0000 = 640
   Change in state:        ---|---|---|  ---|!Im|!Pr|!Rc|  ---|---|---|---
```

vvEngAPI::vvPreprocess

```
   Conditions of interest:  Re| Se|---|  ---| Im|---|---|  ---|---| En| Er
110 0100 0011 = 643
   Required values:         Re| Se|---|  ---| Im|---|---|  ---|---|!En|!Er
110 0100 0000 = 640
   Change in state:        ---|---|---|  ---|---| Pr|---|  ---|---|---|---
```

vvEngAPI::vvRecognize

```
   Conditions of interest:  Re| Se|---|  ---| Im| Pr|---|  ---|---| En| Er
110 0110 0011 = 663
   Required values:         Re| Se|---|  ---| Im| Pr|---|  ---|---|!En|!Er
110 0110 0000 = 660
   Change in state:        ---|---|---|  ---|---|---| Rc|  ---|---|---|---
```

vvEngAPI::vvInitValues

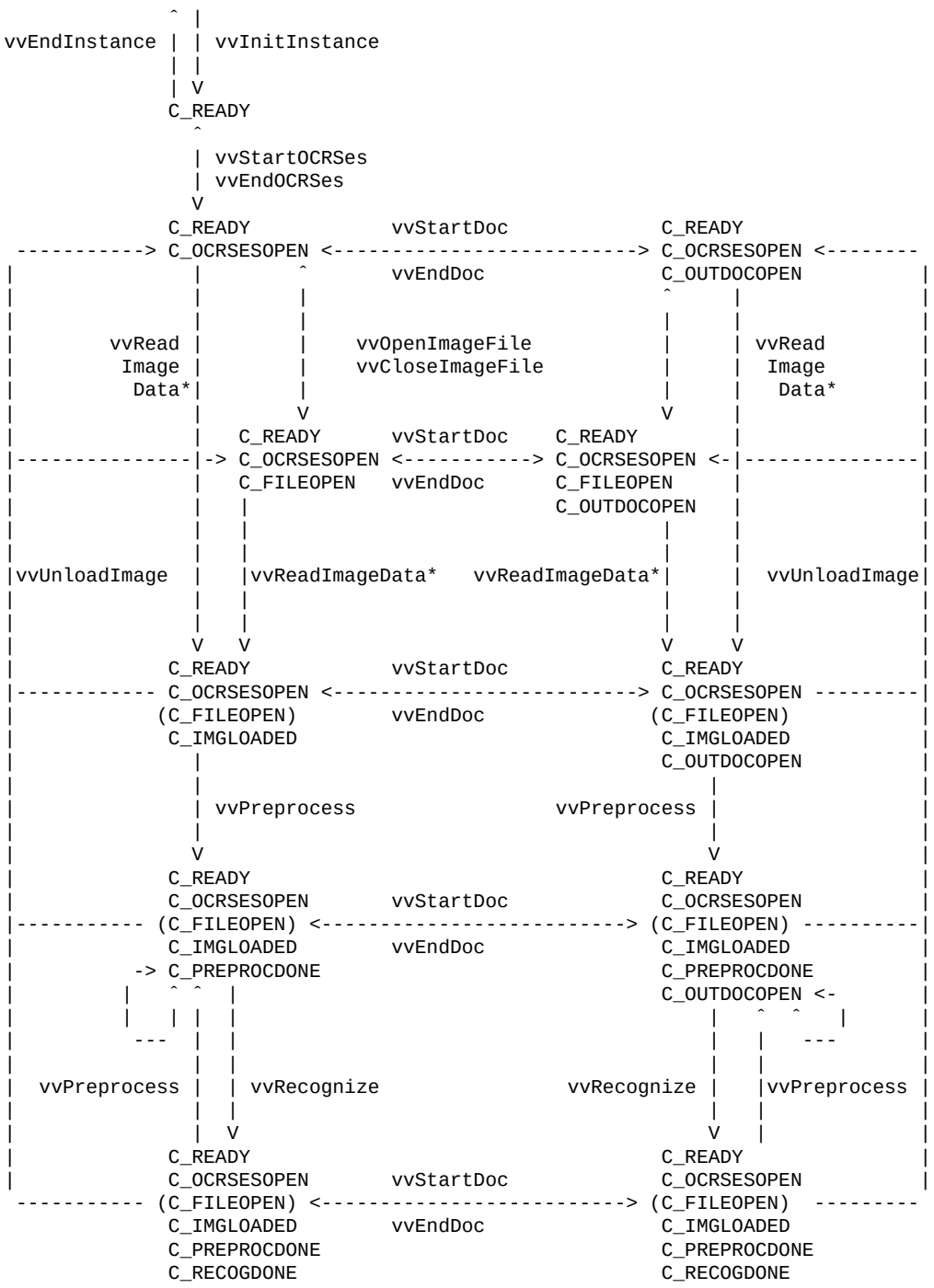
```
   Conditions of interest:  Re| Se|---|  ---|---|---|---|  ---|---| En| Er
110 0000 0011 = 603
   Required values:         Re| Se|---|  ---|---|---|---|  ---|---|!En|!Er
110 0000 0000 = 600
   Change in state:        ---|---|---|  ---|---|---|---|  ---|---|---|---
```

Commands:

- vvEngAPI::vvKill
- vvEngAPI::vvGetStatus
- vvEngAPI::vvSetValue
- vvEngAPI::vvGetValue
- vvEngAPI::vvGetState
- vvEngAPI::vvGetCond
- vvEngAPI::vvGetStatusString
- vvEngAPI::vvSetHint
- vvEngAPI::vvGetHint

- vvEngAPI::vvGetVersion
- vvEngAPI::vvAddWordToLexicon
- vvEngAPI::vvSetLexicalConstraints
- vvEngAPI::vvSetRegionProperties

Partial state diagram:



Remarks:

∗ vvEngAPI::vvReadImageData can be called by two methods: one with the image structure passed as a parameter, the other after having opened a file. vvEngAPI::vvReadImageData can not be called with zero parameters if an input image file is not currently open.

vvEngAPI::vvAcquireDocFile and vvEngAPI::vvAcquireDocMemory may be called at any time C_OUTDOCOPEN is false, C_READY is true, and C_OCRSESOPEN is true. However, it will only be meaningful if an output document has been created through vvEngAPI::vvStartDoc and vvEngAPI::vvEndDoc.

vvEngAPI::vvAcquireSubimageFile and vvEngAPI::vvAcquireSubimageMemory may be called at any time C_READY is true and C_OCRSESOPEN is true. However, it will only be meaningful if an output subimage has been created through vvEngAPI::vvCaptureSubimage.

Definition at line 426 of file vvxtrAPI.h.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 virtual vvEngAPI::~vvEngAPI () [inline, virtual]

Destructor.

Definition at line 434 of file vvxtrAPI.h.

7.1.3 Member Function Documentation

7.1.3.1 virtual vvxtrStatus vvEngAPI::vvAcquireDocFile (const char ∗ *filename*) [pure virtual]

Get the document output as a file.

Call this function after you have created an output document through the calls to vvEngAPI::vvStartDoc, vvEngAPI::vvSpoolDoc, and vvEngAPI::vvEndDoc. You must end the output document before you acquire it through this function.

This function writes the document output to a file, as specified by the passed filename. If the file already exists, it will be overwritten.

Precondition:

An output document must have been created and closed.

Parameters:

filename Name to use for output file.

Returns:

vvxtrStatus based on status

Referenced by outputDoc().

7.1.3.2 virtual vvxtrStatus vvEngAPI::vvAcquireDocMemory (void * *memPtr*, int *bufsize*) [pure virtual]

Get the document output through memory.

Call this function after you have created an output document through the calls to vvEngAPI::vvStartDoc, vvEngAPI::vvSpoolDoc, and vvEngAPI::vvEndDoc. You must end the output document before you acquire it through this function.

You are responsible for allocating the memory passed to this function, and you are responsible for deleting the memory in your client program. Allocate memory before calling this function, using the size obtained through finding the dm_doc_memory_size data value, then pass a pointer to the allocated memory. The buffer size (bufsize) you pass should correspond to both dm_doc_memory_size and the amount of memory you allocated.

Precondition:
Sufficient memory must be allocated in the passed buffer.
An output document must have been created and closed.

Parameters:
memPtr Pointer to preallocated memory.
bufsize Size of the buffer pointed to by memPtr.

Returns:
vvxtrStatus based on status

Referenced by outputDoc().

7.1.3.3 virtual vvxtrStatus vvEngAPI::vvAcquireSubimageFile (const char * *filename*) [pure virtual]

Write the image output to a file.

After capturing a subimage, you can obtain the image data as a file by calling this function. If the file already exists, it will be overwritten.

Precondition:
An image must have been captured through a call to vvEngAPI::vvCaptureSubimage.
dm_out_graphics_format must be set before calling this function to set the format of the image data.

Parameters:
filename Output filename

Returns:
vvxtrStatus based on status

Referenced by outputImg().

7.1.3.4 virtual vvxtrStatus vvEngAPI::vvAcquireSubimageMemory (void * *memPtr*, int *bufsize*) [pure virtual]

Get the image output through memory.

After capturing a subimage, you can obtain the image data through memory by calling this function.

You are responsible for allocating and later deleting the memory in the passed buffer.

Precondition:

An image must have been captured through a call to vvEngAPI::vvCaptureSubimage.
Sufficient memory must be allocated in the passed buffer by the client program. Find out the required memory size through the value dm_subimage_memory_size.
dm_out_graphics_format must be set before calling this function to set the format of the image data.

Parameters:

memPtr Pointer to allocated memory.

bufsize Size of the buffer pointed to by memPtr.

Returns:

vvxtrStatus based on status

Referenced by outputImg().

7.1.3.5 virtual vvxtrStatus vvEngAPI::vvAddWordToLexicon (const char * *word*, int *constraintId*) [pure virtual]

Add a word to a user lexicon.

Multiple words may be added to a user lexicon. Words will be grouped into lexicons based on the constraintId.

The constraintId should be used to set the dm_region_lexical_constraint_id or the dm_word_lexicon_id. The constraintId is associated with a page or a region through vvEngAPI::vvSetLexicalConstraints.

Parameters:

word Word to add the the user lexicon.

constraintId Specifies a group of user lexicon words; use with dm_region_lexical_constraint_id; set ::dm_wordlist_id to this before calling vvEngAPI::vvSetLexicalConstraints.

Returns:

vvxtrStatus

7.1.3.6 vvxtrStatus vvEngAPI::vvCaptureSubimage (int *regionID*) [inline]

Capture a subimage.

This is a helper function that sets the current region to the passed regionID, before calling vvEngAPI::vvCaptureSubimage. (It automatically calls vvSetValue() for dm_current_region.)

Returns:

vvxtrStatus based on status

Definition at line 685 of file vvxtrAPI.h.

References dm_current_region, vvCaptureSubimage(), vvSetValue(), vvxtr_stdarg, and vvxtrStatus.

7.1.3.7 virtual vvxtrStatus vvEngAPI::vvCaptureSubimage () [pure virtual]

Capture a subimage.

This function is the first step in obtaining image data for output. First, set which region you wish to capture using the value dm_current_region. Then call this function. After this function completes, you can obtain the requested image data through vvEngAPI::vvAcquireSubimageFile or vvEngAPI::vvAcquireSubimage-Memory.

Precondition:
You must set dm_current_region before calling this function.
You may also wish to set dm_output_img_source before calling this function.

Returns:
vvxtrStatus based on status

Referenced by outputImg(), and vvCaptureSubimage().

7.1.3.8 virtual vvxtrStatus vvEngAPI::vvCloseImageFile () [pure virtual]

Close the input image file currently open in the OCR engine.

This function closes the currently open input image file, releasing the engine to open another input image file in the future.

This function must be called while an image file is open (image files are opened with the function vv-EngAPI::vvOpenImageFile); you should not call this function if you are reading image data directly from memory rather than from an input file. Note that before the image file may be closed, you must unload any currently loaded page of image data with a call to vvEngAPI::vvUnloadImage.

See also:
vvEngAPI::vvOpenImageFile
vvEngAPI::vvReadImageData
* vvEngAPI::vvUnloadImage

Precondition:
An input image file must be open.

Returns:
vvxtrStatus based on status

Referenced by processOneDocument().

7.1.3.9 virtual vvxtrStatus vvEngAPI::vvEndDoc () [pure virtual]

End the output document.

After you are done spooling output and are ready to acquire your output, you end the output document by calling this function. At this point, you can no longer spool to the output, and you can call vvEngAPI::vv-AcquireDocMemory or vvEngAPI::vvAcquireDocFile in order to acquire a copy of the output document in your client program.

After vvEndDoc is called, the output document will remain in the engine until a new output document is started or the session is ended.

Memory management for the output document is handled internally in the engine; you do not need to be concerned with it in relation to this function.

Precondition:
An output document must be open.

Returns:
vvxtrStatus based on status

Referenced by outputDoc().

7.1.3.10 virtual vvxtrStatus vvEngAPI::vvEndInstance () [pure virtual]

End OCR engine instance.

Precondition:
An OCR instance must have been started.

Returns:
vvxtrStatus based on status

Referenced by runOCR().

7.1.3.11 virtual vvxtrStatus vvEngAPI::vvEndOCRSes () [pure virtual]

End an OCR session.

Precondition:
An OCR session must be open.

Returns:
vvxtrStatus based on status

Referenced by runOCR().

7.1.3.12 virtual vvxtrStatus vvEngAPI::vvForceRegionForeground (int *foregroundColor*) [pure virtual]

Explicitly set the foreground photometric interpretation.

Sets the photometric interpretation of the current region and processes the image to match the set interpretation.

If the passed foreground color matches the current setting, the image within the region remains unchanged, otherwise, the image within the region is inverted in order to agree with the passed color.

This function assumes the photometric interpretation is correct prior to the function call.

This function does not affect image output.

Uses the value of: dm_current_region

Parameters:
foregroundColor Foreground color (black or white) (use vvxtrRegionForegroundEnum)

Returns:
vvxtrStatus

7.1.3.13 virtual vvxtrStatus vvEngAPI::vvGetCond (const vvxtr_cond *c*, int * *result*) [pure virtual]

Get the value of an engine condition.

Parameters:

c The function will look up the value of this condition. (Possible values are C_READY, C_OCRSESOPEN, C_OUTDOCOPEN, C_FILEOPEN, C_IMAGELOADED, C_PREPROCDONE, C_RECOGDONE, C_DOCREADY, C_SUBIMAGEREADY, C_ENGINEBUSY, and C_ERROR.)

result The function will write the value of condition c here.

Returns:

vvxtrStatus based on status

See also:

Engine State and API call sequencing

7.1.3.14 virtual bool vvEngAPI::vvGetHint (vvxtrHint *hint*) [inline, virtual]

Return the current setting of one of the hints.

Parameters:

hint Hint to query (vvxtrHint).

Returns:

The current hint setting.

Definition at line 1025 of file vvxtrAPI.h.

References m_hints.

7.1.3.15 virtual vvxtrStatus vvEngAPI::vvGetState (vvxtr_state * *resultState*) [pure virtual]

Get the engine state.

Parameters:

resultState The function will write the current engine state here; see vvxtr_state.

Returns:

vvxtrStatus based on status

See also:

vvxtr_state to decipher the returned state bit string.
Engine State and API call sequencing

7.1.3.16 virtual vvxtrStatus vvEngAPI::vvGetStatus (const int *key*, vvxtr_stdarg * *result*) [pure virtual]

Get engine status for the passed key.

Parameters:

key The key for which we want the status. See vvxtrStatusEnum.

result The function will write the key's status here.

Returns:

vvxtrStatus based on status

7.1.3.17 virtual const char* vvEngAPI::vvGetStatusString (vvxtrStatus *status* = VVXTR_PREVIOUS_ERROR) [inline, virtual]

Returns a string that describes the last error condition, if any detailed information about the error is available.

Returns:

A temporary string pointer. The string will often be invalidated on the next API call.

Definition at line 1046 of file vvxtrAPI.h.

References VVXTR_ERR_BAD_LANGUAGE, VVXTR_ERR_BUFFER_TOO_SMALL, VVXTR_ERR_CONNECTION_FAILED, VVXTR_ERR_COULD_NOT_CONVERT_DEPTH, VVXTR_ERR_CREATE_LANG_GROUP, VVXTR_ERR_END_SESSION_FAILED, VVXTR_ERR_ENDINSTANCE_FAILURE, VVXTR_ERR_ENG_OUT_OF_MEMORY, VVXTR_ERR_ENGINE_KILLED, VVXTR_ERR_GETTING_IMG_REGION, VVXTR_ERR_HOST_LOOKUP_FAILED, VVXTR_ERR_IMG_ACQ_FAILED, VVXTR_ERR_INITINSTANCE_FAILURE, VVXTR_ERR_INV_PDF_FORMAT, VVXTR_ERR_INV_SUBIMG_FORMAT, VVXTR_ERR_INVALID_PAGE, VVXTR_ERR_INVALID_REGION_ID, VVXTR_ERR_INVALID_SHAPE_PACK_PATH, VVXTR_ERR_INVALID_STATE, VVXTR_ERR_INVALID_SYNTAX, VVXTR_ERR_INVALID_UOR_COUNT, VVXTR_ERR_INVALID_UOR_STRING, VVXTR_ERR_LOAD_LANG_FAILED, VVXTR_ERR_MISSING_LANG, VVXTR_ERR_NO_DOC, VVXTR_ERR_NO_FEATURE, VVXTR_ERR_NO_INPUT, VVXTR_ERR_NO_INPUT_FILE_SPECIFIED, VVXTR_ERR_NO_LICENSE, VVXTR_ERR_NO_LICENSE_MANAGER, VVXTR_ERR_NO_SUBIMAGE, VVXTR_ERR_NOENGINE, VVXTR_ERR_NOT_IMPLEMENTED, VVXTR_ERR_OPEN_CHAR_SET_FAILURE, VVXTR_ERR_OPENING_LANG_PACK, VVXTR_ERR_OPENING_OUTPUT_FILE, VVXTR_ERR_OUTPUT_INV_REGION, VVXTR_ERR_PDF_END_OUTPUT, VVXTR_ERR_PDF_IMG_OUTPUT, VVXTR_ERR_PP_FAILED, VVXTR_ERR_READONLY_VALUE, VVXTR_ERR_RECOGNITION_FAILED, VVXTR_ERR_REGION_ECLIPSED, VVXTR_ERR_REGION_HANDLER, VVXTR_ERR_REMOVE_FILE_FAILED, VVXTR_ERR_SET_DEF_LEX_CONSTRAINTS, VVXTR_ERR_SET_OPTIONS_FAILED, VVXTR_ERR_SET_REGION_UNSUCCESSFUL, VVXTR_ERR_START_SESSION_FAILED, VVXTR_ERR_SYNCHRONIZATION, VVXTR_ERR_TRANSFER_FAILED, VVXTR_ERR_UNABLE_TO_LOAD_IMAGE_FILE, VVXTR_ERR_WRITE_OUTPUT_FAILED, and VVXTR_PREVIOUS_ERROR.

7.1.3.18 vvxtrStatus vvEngAPI::vvGetValue (const int *key*, char ** *result*) [inline]

Get a character or string engine value.

Parameters:

key The key for which we want the value in the engine.

result The function will write the key's value here.

Returns:

vvxtrStatus based on status

See also:

vvxtrEngineVariables

Definition at line 970 of file vvxtrAPI.h.

References vvGetValue(), vvxtr_stdarg, and vvxtrStatus.

7.1.3.19 vvxtrStatus vvEngAPI::vvGetValue (const int *key*, int * *result*) [inline]

Get an integer engine value.

Parameters:

key The key for which we want the value in the engine.

result The function will write the key's value here.

Returns:

vvxtrStatus based on status

See also:

vvxtrEngineVariables

Definition at line 957 of file vvxtrAPI.h.

References vvxtr_stdarg, and vvxtrStatus.

Referenced by outputDoc(), outputImg(), processOneDocument(), and vvGetValue().

7.1.3.20 virtual vvxtrStatus vvEngAPI::vvGetVersion (vvxtrVersionLocationEnum *loc*, const char ** *ver*) [pure virtual]

Return the version.

Parameters:

loc Specifies whether to return the version of the local or remote system (see vvxtrVersionLocationEnum).

ver version

Returns:

vvxtrStatus

7.1.3.21 virtual vvxtrStatus vvEngAPI::vvInitInstance () [pure virtual]

Initialize OCR engine instance.

vvInitInstance should be called after starting an OCR session and at any point within an OCR session where you would like to reset the values. It is NOT required to call vvEngAPI::vvInitInstance prior to recognition of every file.

Precondition:
An OCR session must be started.

Returns:
vvxtrStatus based on status

Referenced by runOCR().

7.1.3.22 virtual vvxtrStatus vvEngAPI::vvInitValues () [pure virtual]

Initialize engine values.

Returns:
vvxtrStatus based on status

Referenced by runOCR().

7.1.3.23 virtual vvxtrStatus vvEngAPI::vvKill () [pure virtual]

Kill the engine.

vvKill should not normally be called. It is an asynchronous exit, so only use it if you need to exit for an unexpected reason or if the engine hangs.

Returns:
vvxtrStatus based on status

Referenced by CloseDown().

7.1.3.24 vvxtrStatus vvEngAPI::vvOpenImageFile (const char * *fileName*) [inline]

Open the input image file passed as a parameter.

This is a helper function that integrates the vvEngAPI::vvSetValue call in order to implicitely set the value of dm_in_filename. See the documentation for vvEngAPI::vvOpenImageFile.

Parameters:
fileName The file to open.

Returns:
vvxtrStatus based on the status of the file open or of the vvEngAPI::vvSetValue call.

Definition at line 759 of file vvxtrAPI.h.

References dm_in_filename, vvOpenImageFile(), vvSetValue(), vvxtr_stdarg, and vvxtrStatus.

7.1.3.25 virtual vvxtrStatus vvEngAPI::vvOpenImageFile () [pure virtual]

Open an input image document in the OCR engine.

This function opens an input image document file. A page of the input image document must be still opened before the engine can process any image data. This function should be called once for each input image file.

See the list of supported input file formats.

Precondition:

dm_in_filename must be set before this function is called, so that the engine knows what image file to open.

An input image file cannot currently be open.

Returns:

vvxtrStatus based on status

Referenced by openImageDocument(), and vvOpenImageFile().

7.1.3.26 virtual vvxtrStatus vvEngAPI::vvPreprocess () [pure virtual]

Run preprocessing on the currently loaded image page.

vvEngAPI::vvPreprocess should be called for every new image loaded into the engine, because it is an important step in the OCR process, where the engine segments the image and runs image processing over the image data in order to improve the ultimate OCR results.

Preprocessing is separate from recognition so that it can be run multiple times before recognition, if desired. In addition, preprocessing could be run without recognition at all, if for example, someone wanted to extract the image regions from a document but didn't care about recognizing text. However, calling vvEngAPI::vvPreprocess once and then vvEngAPI::vvRecognize once is by far the most common usage.

The preprocessing functions used in this step are intended as an aid to optical character recognition during the recognition step, and are not intended to affect or in any way clean up image output you may generate.

Precondition:

Image data must be loaded in the OCR engine (see vvEngAPI::vvReadImageData).

All preprocessing options should be set before this function is called. See all the values starting with "dm_pp_".

Returns:

vvxtrStatus based on status

Referenced by processOnePage().

7.1.3.27 virtual vvxtrStatus vvEngAPI::vvReadImageData (const struct vvxtrImage * *img*) [pure virtual]

Read image data into the OCR engine.

This function loads a page of input image data from the passed image structure into the OCR engine. It provides an alternative method for loading image data into the OCR engine, by allowing you to directly place the image data into memory and instead of reading it from a file.

After this function call, the loaded image data may be processed with, for example, vvEngAPI::vvPreprocess and then vvEngAPI::vvRecognize.

(Note: When this function is used, vvEngAPI::vvOpenImageFile does NOT need to be called first.)

Only one page of data can be handled at a time through this function.

You are responsible for memory management of the the passed vvxtrImage data structure.

Precondition:

No image data can currently be loaded in the OCR engine.

Returns:
vvxtrStatus based on status

7.1.3.28 virtual vvxtrStatus vvEngAPI::vvReadImageData () [pure virtual]

Read image data into the OCR engine.

This function reads one page of the currently open image file and loads it into the OCR engine in preparation for OCR processing. After this function call, the loaded image data may be processed with, for example, vvEngAPI::vvPreprocess and then vvEngAPI::vvRecognize.

Precondition:
The value dm_in_curr_page should be set before this function is called so that the engine knows which page of the input image file to read.

Returns:
vvxtrStatus based on status

Referenced by processOneDocument().

7.1.3.29 virtual vvxtrStatus vvEngAPI::vvRecognize () [pure virtual]

Run recognition on the currently loaded image page.

This function runs optical character recognition on the currently loaded image data; it is the step where the image data is converted into text and formatting information internally.

When the recognition step is run, the recognized text is stored internally and can be accessed by generating output (see vvEngAPI::vvWriteOutput). Each time vvRecognize is called, the internal buffer of recognized text is cleared before it is written; recognized text from previous calls to vvRecognize is no longer available.

Precondition:
Image data must be loaded in the OCR engine (see vvEngAPI::vvReadImageData).
Preprocessing must have been run on this image data (see vvEngAPI::vvPreprocess).
Set all recognition options before calling this function.

Returns:
vvxtrStatus based on status

Referenced by processOnePage().

7.1.3.30 virtual vvxtrStatus vvEngAPI::vvRemoveRegion (void) [pure virtual]

Remove a region.

Removes the current region, as specified by the dm_current_region value.

Precondition:
dm_current_region must be set prior to this call.

Returns:
vvxtrStatus

7.1.3.31 virtual vvxtrStatus vvEngAPI::vvSetHint (vvxtrHint *hint*, bool *set* = true) [inline, virtual]

Give the engine more information about its environment to possibly improve performance.

Not all hints will have an effect.

See also:
vvxtrHint

Parameters:
hint Hint to add or remove (vvxtrHint).

set True to set the hint. False to remove the hint.

Returns:
vvxtrStatus - VV_OK on success.

Definition at line 1016 of file vvxtrAPI.h.

References m_hints, and vvxtrStatus.

Referenced by runOCR().

7.1.3.32 virtual vvxtrStatus vvEngAPI::vvSetLexicalConstraints (int *mode*) [pure virtual]

Set lexical constraints for an area.

For a specific area, one or more of these lexical constraints may be set:

- character set
- language
- word list

The character set should be set in the dm_char_set value. If dm_char_set is NULL, then the character set lexical constraint will not be set.

The language(s) should be set in the dm_language value. If it is NULL, then the language lexical constraint will not be set.

The word list should be set up using vvEngAPI::vvAddWordToLexicon and setting ::dm_wordlist_id to the constraintId passed to the function. If ::dm_wordlist_id is -1, then the word list lexical constraint will not be set.

Lexical constraints may be set for the page or for a specific region. Lexical constraints for a specific region will override those set for the page. Specify the area through the value dm_focus_area and dm_current_region (when dm_focus_area == vvFocusAreaRegion).

Parameters:
mode Specifies whether the lexical constraints should be added or replaced for this area.

See also:
vvxtrLexicalConstraintModeEnum.

Returns:
vvxtrStatus

7.1.3.33 virtual vvxtrStatus vvEngAPI::vvSetRegionProperties (void) [pure virtual]

Set region information.

Set properties of the region specified by dm_current_region.

A new region may be created by setting region properties for an unused region id.

Note that if two regions overlap, one will obscure the other, based on the regions' values for dm_region_stacking.

Precondition:

These values will be read by this function and should be set as desired before calling this function:

- dm_region_type (vvxtrRegionTypeEnum)
- dm_region_subtype (vvxtrRegionSubtypeEnum)
- dm_region_stacking (number)
- dm_region_grammar_mode (vvxtrRegionGrammarModeEnum)
- dm_region_lexical_constraint_id (number corresponding to a user lexicon set up with vvEngAPI::vvAddWordToLexicon)
- dm_region_lexmode (vvxtrRegionLexmodeEnum)
- dm_region_foreground (vvxtrRegionForegroundEnum)
- dm_region_out_order (number)
- dm_region_name (string for region name)
- dm_region_uor_string (UOR boundary definition of region)
- dm_region_uor_count (number of rectangles in the dm_region_uor_string)

Returns:

vvxtrStatus

7.1.3.34 vvxtrStatus vvEngAPI::vvSetValue (const int *key*, char * *value*) [inline]

Set the string value of the passed key in the engine.

Parameters:

key The key whose value we wish to set.

value The function will set the key to this value in the engine.

Returns:

vvxtrStatus based on status

See also:

vvxtrEngineVariables

Definition at line 940 of file vvxtrAPI.h.

References vvSetValue(), vvxtr_stdarg, and vvxtrStatus.

7.1.3.35 vvxtrStatus vvEngAPI::vvSetValue (const int *key*, int *value*) [inline]

Set the integer value of the passed key in the engine.

Parameters:

key The key whose value we wish to set.

value The function will set the key to this value in the engine.

Returns:

vvxtrStatus based on status

See also:

vvxtrEngineVariables

Definition at line 927 of file vvxtrAPI.h.

References vvxtr_stdarg, and vvxtrStatus.

Referenced by openImageDocument(), outputImg(), processOneDocument(), runOCR(), vvCapture-Subimage(), vvOpenImageFile(), vvSetValue(), and vvStartDoc().

7.1.3.36 virtual void vvEngAPI::vvSigHUP () [inline, virtual]

A function to call when your client application receives a signal of type SIGHUP.

Will reread any local parameters (does not communicate across a link; use the utility vvlmreread to have a license daemon reread the license file).

Definition at line 1035 of file vvxtrAPI.h.

7.1.3.37 virtual vvxtrStatus vvEngAPI::vvSpoolDoc () [pure virtual]

Spool recognized data to the output doc.

After you recognize text, if you want to retain that text, you must spool it to an output document before running recognition again.

Before calling this function, you must start an output document with vvEngAPI::vvStartDoc, and that output document must still be open. You can call vvSpoolDoc multiple times for one output document, and writing output is independent of reading input documents. The output document remains open and you may spool to it until you close it with the function vvEngAPI::vvEndDoc.

This function is independent of the output format, or the eventual output destination (file or memory).

Precondition:

An output document must already be started.
dm_output_img_source should be set if you wish to use a different image source than the default (only applicable to compound output documents with embedded images).

Returns:

vvxtrStatus based on status

Referenced by outputDoc().

7.1.3.38 vvxtrStatus vvEngAPI::vvStartDoc (int *fileFormat*) [inline]

Start an output document.

Please see the documentation for vvEngAPI::vvStartDoc.

This is a helper function that calls vvSetValue to set dm_out_text_format prior to calling vvEngAPI::vvStartDoc()

Returns:
vvxtrStatus based on status

Definition at line 561 of file vvxtrAPI.h.

References dm_out_text_format, vvSetValue(), vvStartDoc(), vvxtr_stdarg, and vvxtrStatus.

7.1.3.39 virtual vvxtrStatus vvEngAPI::vvStartDoc () [pure virtual]

Start an output document.

An output document is handled internally in the engine. You begin it by calling this function, send data to it by calling vvEngAPI::vvSpoolDoc, and end it by calling vvEngAPI::vvEndDoc. The output document can span multiple input files and multiple pages of recognized image data, depending on how much information you would like in your output document.

When you call this function, if a previous output document existed, it is no longer available.

Memory management for the output document is handled internally throughout the sequence of vvEngAPI::vvStartDoc, vvEngAPI::vvSpoolDoc, and vvEngAPI::vvEndDoc. You do not need to allocate or delete memory until you acquire a completed output document; please see vvEngAPI::vvAcquireDocMemory and vvEngAPI::vvAcquireDocFile.

Precondition:
The file format should be specified prior to calling this function (see dm_out_text_format, vvxtrOutTextFormatEnum), or use the other version of vvStartDoc that takes the file format as a parameter.

Returns:
vvxtrStatus based on status

Referenced by outputDoc(), and vvStartDoc().

7.1.3.40 virtual vvxtrStatus vvEngAPI::vvStartOCRSes () [pure virtual]

Start an OCR session.

Sets default lexical constraints for the OCR session based on:

dm_language dm_char_set

This function should be called as a part of the start up process for a set of similar documents, after calling vvEngAPI::vvInitInstance. It may also be called any time after vvEngAPI::vvEndOCRSes, as long as vvEngAPI::vvEndInstance has not been called, ending the current instance.

Within one OCR session, internal training is built up on the recognized characters using the language (dictionary) information as reinforcement. This training is sensitive to:

- font

- point size
- density
- style

and helps the system recognize deformed characters when it has seen good versions before. Ending and restarting the OCR session flushes this training cache.

As a result of the training that takes place within an OCR session, one OCR session should be considered one ”job” in the workflow, where the separate input documents of that job are likely to have pages that are related in some sense. Processing of these similar documents should be braketed by calls to vvEngAPI::vvStartOCRSes and vvEngAPI::vvStartOCRSes.

It will improve the recognition results to permit training to persist between pages of documents from the same source or rendering environment (printer, carbon tape, toner cartridge, etc). Conversely, it will hurt the recognition results if you train over too much disparate material. When in doubt, it is probably best to err on the side of caution and start and end OCR sessions more frequently rather than less frequently. For example, you could start and end the OCR session after every 10 pages, with additional breaks where you know you should have them.

Precondition:
The instance must be initialized and an OCR session cannot currently be open.

Returns:
vvxtrStatus based on status

Referenced by runOCR().

7.1.3.41 virtual vvxtrStatus vvEngAPI::vvUnloadImage () [pure virtual]

Unload the page of input image data currently loaded in the OCR engine.

This function should be called after input image data has been loaded with vvEngAPI::vvReadImageData and after all processing on the image data is complete.

No more preprocessing or recognition may be performed after this function is called. However, after this function returns, you can load another page of input image data into the OCR engine by calling vvEngAPI::vvReadImageData again.

Calling this function does not affect your output document.

Precondition:
Image data must be currently loaded in the OCR engine (see vvEngAPI::vvReadImageData).

Returns:
vvxtrStatus based on status

Referenced by processOneDocument().

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

- vvxtrAPI.h

7.2 vvxtrImage Class Reference

An encapsulation of information about a bitmap.

`#include <vvxtrAPI.h>`

Public Member Functions

- vvxtrImage ()
  Constructor.
- void SetWidth (unsigned int width)
  Set the image width.
- void SetHeight (unsigned int height)
  Set the image height.
- void SetBytesPerLine (unsigned int bpl)
  Set the bytes per line of the image.
- void SetXDPI (unsigned short int xdpi)
  Set the horizontal DPI of the image.
- void SetYDPI (unsigned short int ydpi)
  Set the vertical DPI of the image.
- void SetBitsPerPixel (unsigned short int bpp)
  Set the bits per pixel of the image.
- void SetBitsPerSample (unsigned short int bps)
  Set the bits per sample of the image.
- void SetData (unsigned char *imgData)
  Set the location of the bitmap data.
- unsigned int GetWidth ()
  Get the current image width.
- unsigned int GetHeight ()
  Get the current image height.
- unsigned int GetBytesPerLine ()
  Get the number of bytes per line.
- unsigned short int GetXDPI ()
  Get the horizontal bit density.
- unsigned short int GetYDPI ()
  Get the vertical dot density.

- unsigned short int GetBitsPerPixel ()
  Get the number of bits per pixel.
- unsigned short int GetBitsPerSample ()
  Get the bits per sample of the image.
- unsigned char * GetData ()
  Get a pointer to the raw data.
- const unsigned char * **GetData** () const
- int GetImageSize () const
  Get the total image size.
- int GetTotalSize () const
  Get the total size, including the size of this structure.

Static Public Member Functions

- vvxtrImage * FromRaw (void *rawbuffer)
  A vvxtrImage factory that creates a vvxtrImage instance in-place in a raw data buffer.

Protected Attributes

- int size
  Size of the image header.
- unsigned int dwImgWidth
  image width in pixels
- unsigned int dwImgHeight
  image height in pixels
- unsigned int dwBytesPerLine
  number of data bytes per X scanline
- unsigned short int wXDPI
  resolution in pixels of X scanline data
- unsigned short int wYDPI
  resolution in pixels of Y scanline data
- unsigned short int chBitsPerPixel
  image depth (1, 4, 8, 24)
- unsigned short int wBitsPerSample
  image bits per sample (1, 8)

- unsigned char * data

  image data

7.2.1 Detailed Description

An encapsulation of information about a bitmap.

The image header sent along with memory resident bitmaps.

The image needs to be in a chunky (i.e. not planar) form, at 1, 8, or 24 bits per pixel. 8-bit is for grayscale only.

The destructor does not delete the data, as this class does not own the data.

See also:
vvEngAPI::vvReadImageData(const struct vvxtrImage * img)

Definition at line 1271 of file vvxtrAPI.h.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 vvxtrImage::vvxtrImage () [inline]

Constructor.

Definition at line 1278 of file vvxtrAPI.h.

References size.

7.2.3 Member Function Documentation

7.2.3.1 vvxtrImage* vvxtrImage::FromRaw (void * *rawbuffer*) [inline, static]

A vvxtrImage factory that creates a vvxtrImage instance in-place in a raw data buffer.

Used to encapsulate a vvxtrImage with the raw bitmap data in one contiguous buffer, which could be used, for example, to transmit the data in one block.

This function is primarily for internal use; if its purpose is not clear to you, then you probably don't need to use it.

Parameters:
rawbuffer Pointer to the data buffer, which must be at least sizeof(vvxtrImage)+sizeof(targetimage)

Returns:
A pointer to the initialized memory region.

Definition at line 1415 of file vvxtrAPI.h.

References SetData().

Referenced by outputImg().

7.2.3.2 unsigned short int vvxtrImage::GetBitsPerPixel () [inline]

Get the number of bits per pixel.

Returns:
Bits per pixel.

Definition at line 1370 of file vvxtrAPI.h.

References chBitsPerPixel.

7.2.3.3 unsigned short int vvxtrImage::GetBitsPerSample () [inline]

Get the bits per sample of the image.

Returns:
Bits per sample.

See also:
SetBitsPerSample

Definition at line 1377 of file vvxtrAPI.h.

References wBitsPerSample.

7.2.3.4 unsigned int vvxtrImage::GetBytesPerLine () [inline]

Get the number of bytes per line.

Returns:
Bytes per line.

Definition at line 1352 of file vvxtrAPI.h.

References dwBytesPerLine.

7.2.3.5 unsigned char* vvxtrImage::GetData () [inline]

Get a pointer to the raw data.

Returns:
Raw data pointer.

Definition at line 1384 of file vvxtrAPI.h.

References data.

7.2.3.6 unsigned int vvxtrImage::GetHeight () [inline]

Get the current image height.

Returns:
Image height in pixels.

Definition at line 1346 of file vvxtrAPI.h.

References dwImgHeight.

7.2.3.7 int vvxtrImage::GetImageSize () const [inline]

Get the total image size.

Returns:
Image size in bytes.

Definition at line 1392 of file vvxtrAPI.h.

References dwBytesPerLine, and dwImgHeight.

Referenced by GetTotalSize().

7.2.3.8 int vvxtrImage::GetTotalSize () const [inline]

Get the total size, including the size of this structure.

Used to allocate a buffer that includes this structure embedded at the start.

Returns:
sizeof(vvxtrImage)+GetImageSize()

Definition at line 1399 of file vvxtrAPI.h.

References GetImageSize().

7.2.3.9 unsigned int vvxtrImage::GetWidth () [inline]

Get the current image width.

Returns:
Image width in pixels.

Definition at line 1340 of file vvxtrAPI.h.

References dwImgWidth.

7.2.3.10 unsigned short int vvxtrImage::GetXDPI () [inline]

Get the horizontal bit density.

Returns:
Number of horizontal dots per inch.

Definition at line 1358 of file vvxtrAPI.h.

References wXDPI.

7.2.3.11 unsigned short int vvxtrImage::GetYDPI () [inline]

Get the vertical dot density.

Returns:
Number of vertical dots per inch.

Definition at line 1364 of file vvxtrAPI.h.

References wYDPI.

7.2.3.12 void vvxtrImage::SetBitsPerPixel (unsigned short int *bpp*) [inline]

Set the bits per pixel of the image.

Parameters:
bpp Bits per pixel

Definition at line 1319 of file vvxtrAPI.h.

References chBitsPerPixel.

7.2.3.13 void vvxtrImage::SetBitsPerSample (unsigned short int *bps*) [inline]

Set the bits per sample of the image.

Parameters:
bps Number of bits in each sample (i.e., in an RGB image, the number of bits in each of Red, Green, and Blue, so for a 24-bit RGB image, bits-per-sample would be 8)

Definition at line 1327 of file vvxtrAPI.h.

References wBitsPerSample.

7.2.3.14 void vvxtrImage::SetBytesPerLine (unsigned int *bpl*) [inline]

Set the bytes per line of the image.

Parameters:
bpl Bytes per line (width * bytes-per-pixel + bytes-between-lines)

Definition at line 1299 of file vvxtrAPI.h.

References dwBytesPerLine.

7.2.3.15 void vvxtrImage::SetData (unsigned char * *imgData*) [inline]

Set the location of the bitmap data.

Parameters:
imgData A pointer to the start of the raw bitmap data.

Definition at line 1333 of file vvxtrAPI.h.

References data.

Referenced by FromRaw().

7.2.3.16 void vvxtrImage::SetHeight (unsigned int *height*) [inline]

Set the image height.

Parameters:
height Height in pixels.

Definition at line 1292 of file vvxtrAPI.h.

References dwImgHeight.

7.2.3.17 void vvxtrImage::SetWidth (unsigned int *width*) [inline]

Set the image width.

Parameters:
width Width in pixels.

Definition at line 1286 of file vvxtrAPI.h.

References dwImgWidth.

7.2.3.18 void vvxtrImage::SetXDPI (unsigned short int *xdpi*) [inline]

Set the horizontal DPI of the image.

Parameters:
xdpi Dots per inch horizontally.

Definition at line 1306 of file vvxtrAPI.h.

References wXDPI.

7.2.3.19 void vvxtrImage::SetYDPI (unsigned short int *ydpi*) [inline]

Set the vertical DPI of the image.

Parameters:
ydpi Dots per inch vertically.

Definition at line 1313 of file vvxtrAPI.h.

References wYDPI.

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

- vvxtrAPI.h

Chapter 8

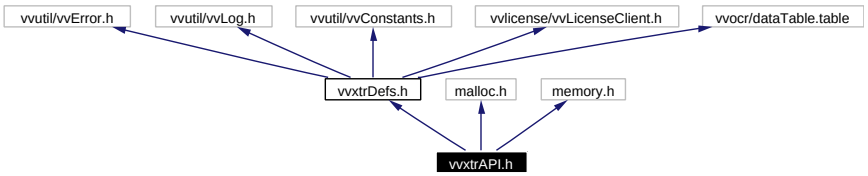
OCR Shop XTR/API User Documentation File Documentation

8.1 vvxtrAPI.h File Reference

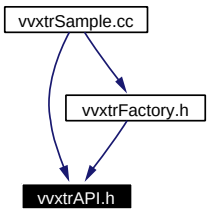
vvxtrAPI.h, Public Interface for OCRShop XTR API

```
#include "vvxtrDefs.h"
#include "malloc.h"
#include "memory.h"
```

Include dependency graph for vvxtrAPI.h:



This graph shows which files directly or indirectly include this file:



Compounds

- class vvEngAPI

Optical Character Recognition Engine API Class.

- class vvxtrImage

  An encapsulation of information about a bitmap.

Defines

- #define **VVXTRAPI_H** 1

8.1.1 Detailed Description

vvxtrAPI.h, Public Interface for OCRShop XTR API

This file needs to be included in a user program that uses the OCRShop XTR API.

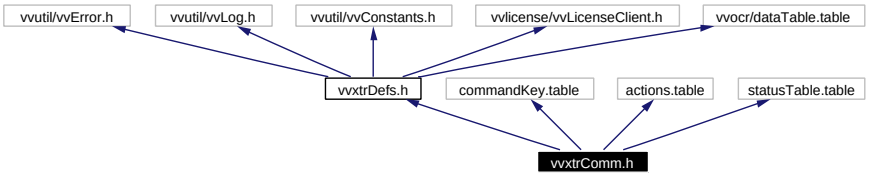
Definition in file vvxtrAPI.h.

8.2 vvxtrComm.h File Reference

vvxtrComm.h, Information needed to communicate with the daemon without the client program.

```
#include "vvxtrDefs.h"
#include "commandKey.table"
#include "actions.table"
#include "statusTable.table"
```

Include dependency graph for vvxtrComm.h:



Compounds

- struct **vvxtrKeyValuePair**

Defines

- #define **commandtable**(A, B, C) A,
- #define **actiontable**(A, B) xtract_##A,
- #define **statustable**(key, action, type) dm_##key,
- #define **TRANSFER_TIMEOUT** 25
- #define **XTR_READY** "xtrReady"

Enumerations

- enum vvxtrCommandKeyEnum {

  query_status, get_value, set_value, do_action, **upload_block**, **download_block**, **get_version**, **add_word_to_lexicon**,

  set_hint, **set_lexical_constraints**, **set_region_properties**, **force_region_foreground**, **remove_region**, **VVXTR_NUM_COMMANDS** }
- enum vvxtrActionEnum {

  xtract_init_instance, xtract_end_instance, **xtract_start_ocr_ses**, **xtract_end_ocr_ses**, **xtract_start_doc**, **xtract_spool_doc**, **xtract_end_doc**, **xtract_capture_subimage**,

  xtract_read_image_data, **xtract_open_image_file**, **xtract_unload_image**, **xtract_close_image_file**, **xtract_preprocess**, **xtract_recognize**, **xtract_init_values**, **xtract_kill**,

  VVXTR_NUM_ACTIONS }
- enum vvxtrStatusEnum {

  dm_daemon_state, dm_engine_state, dm_words_seen, dm_words_recognized, dm_characters_seen, dm_characters_recognized, dm_percentage_done, dm_page_number,

dm_region_number, dm_line_orientation, dm_skew_angle, dm_current_skew_angle, dm_skew_confidence, dm_smotation_angle, dm_text_orientation, dm_min_is_black,

dm_most_complex, dm_byte_order, dm_error_code, dm_version, dm_bitmap_split, dm_line_doubled, dm_regionview_mode, dm_codepage,

dm_lexicon, dm_most_cols, dm_most_rows, dm_most_cells, **VVXTR_NUM_STATUS_TYPES** }

- enum { **vvOcrListenPort** = 10101, **vvOcrLicensePort** = 10102 }
- enum **TransferType** { **ttInvalid** = 0, **ttDoc**, **ttImage**, **ttImageFile**, **ttDocFile**, **ttImageFileLocal**, **ttDocFileLocal** }

8.2.1 Detailed Description

vvxtrComm.h, Information needed to communicate with the daemon without the client program.

This file is only needed if someone wants to write their own client that communicates with the daemon. This file is not needed for normal usage of the API, where the user either links statically with the API or uses the daemon through the client (communicator) program.

Definition in file vvxtrComm.h.

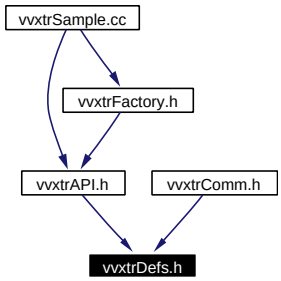
8.3 vvxtrDefs.h File Reference

vvxtrDefs.h, definitions and enumerations for OCRShop XTR API

```
#include <vvutil/vvError.h>
#include <vvutil/vvLog.h>
#include <vvutil/vvConstants.h>
#include <vvlicense/vvLicenseClient.h>
#include "vvocr/dataTable.table"
```

Include dependency graph for vvxtrDefs.h:

This graph shows which files directly or indirectly include this file:



Defines for conditions

Used to construct the vvxtr_state bitstring.

- State is a bit string corresponding to a bitwise OR of all conditions.
- Used in vvEngAPI::vvGetCond function.

- #define C_READY 0x400
  ready

- #define C_OCRSESOPEN 0x200
  OCR session is open.

- #define C_OUTDOCOPEN 0x100
  output document is open

- #define C_FILEOPEN 0x080
  input image is open

- #define C_IMAGELOADED 0x040

  input page is open

- #define C_PREPROCDONE 0x020

  preprocessing has been run on the current page

- #define C_RECOGDONE 0x010

  recognition has been run on the current page

- #define C_DOCREADY 0x008

  output document is ready for aquisition

- #define C_SUBIMAGEREADY 0x004

  output subimage is ready for aquisition

- #define C_ENGINEBUSY 0x002

  engine is currently busy

- #define C_ERROR 0x001

  error

Defines for Error codes

These codes are returned as a vvxtrStatus value when there is an error.

- #define VVXTR_KILL_PROCESS -1000

  Time for the daemon to die.

- #define VVXTR_PREVIOUS_ERROR -1001

  Requesting information on the last error.

- #define VVXTR_ERR -1

  General error.

- enum vvxtrErrorCodes {

  VVXTR_ERR_INVALID_SYNTAX = 1001, VVXTR_ERR_NO_INPUT, VVXTR_ERR_INVALID_STATE, VVXTR_ERR_INITINSTANCE_FAILURE, VVXTR_ERR_ENDINSTANCE_FAILURE, VVXTR_ERR_NO_INPUT_FILE_SPECIFIED, VVXTR_ERR_UNABLE_TO_LOAD_IMAGE_FILE, VVXTR_ERR_IMG_ACQ_FAILED,

  VVXTR_ERR_START_SESSION_FAILED, VVXTR_ERR_RECOGNITION_FAILED, VVXTR_ERR_OPEN_CHAR_SET_FAILURE, VVXTR_ERR_INVALID_SHAPE_PACK_PATH, VVXTR_ERR_BAD_LANGUAGE, VVXTR_ERR_OPENING_LANG_PACK, VVXTR_ERR_LOAD_LANG_FAILED, VVXTR_ERR_CREATE_LANG_GROUP,

  VVXTR_ERR_LANG_NOT_LICENSED, VVXTR_ERR_SET_DEF_LEX_CONSTRAINTS, VVXTR_ERR_WRITE_OUTPUT_FAILED, VVXTR_ERR_OPENING_OUTPUT_FILE, VVXTR_ERR_REGION_ECLIPSED, VVXTR_ERR_COULD_NOT_CONVERT_DEPTH, VVXTR_ERR_GETTING_IMG_REGION, VVXTR_ERR_END_SESSION_FAILED,

VVXTR_ERR_OUTPUT_INV_REGION, VVXTR_ERR_NOT_IMPLEMENTED, VVXTR_ERR_INV_PDF_FORMAT, VVXTR_ERR_PDF_IMG_OUTPUT, VVXTR_ERR_PDF_END_OUTPUT, VVXTR_ERR_UNABLE_TO_WRITE_PDF, VVXTR_ERR_HOST_LOOKUP_FAILED, VVXTR_ERR_CONNECTION_FAILED,

VVXTR_ERR_TRANSFER_FAILED, VVXTR_ERR_NO_SUBIMAGE, VVXTR_ERR_PP_FAILED, VVXTR_ERR_SET_OPTIONS_FAILED, VVXTR_ERR_REMOVE_FILE_FAILED, VVXTR_ERR_NO_DOC, VVXTR_ERR_INV_SUBIMG_FORMAT, VVXTR_ERR_SYNCHRONIZATION,

VVXTR_ERR_NOENGINE, VVXTR_ERR_BUFFER_TOO_SMALL, VVXTR_ERR_INVALID_PAGE, VVXTR_ERR_MISSING_LANG, VVXTR_ERR_ENG_OUT_OF_MEMORY, VVXTR_ERR_INVALID_UOR_COUNT, VVXTR_ERR_INVALID_UOR_STRING, VVXTR_ERR_READONLY_VALUE,

VVXTR_ERR_INVALID_REGION_ID, VVXTR_ERR_SET_REGION_UNSUCCESSFUL, VVXTR_ERR_REGION_HANDLER, VVXTR_ERR_ENGINE_KILLED, VVXTR_ERR_NO_LICENSE, VVXTR_ERR_NO_LICENSE_MANAGER, VVXTR_ERR_LICENSE_COMM_ERROR, VVXTR_ERR_NO_FEATURE }

Time for the daemon to die.

Defines

- #define **VVXTRDEFS_H** 1
- #define VV_DEBUG 1

  To turn on debug messages.

- #define **tablemacro**(key, value, type, kernelFunc, defaultUsed, defaultVal, kernelVal, write) dm_##key,

Typedefs

- typedef enum vvxtr_dataType VVXTR_DATATYPE

  Generic XTR datatype.

- typedef signed long int vvxtr_state

  Representation of the state as a bit-string.

- typedef signed long int vvxtr_cond

  Representation of a single condition in the state bit-string.

- typedef signed long int vvxtrStatus

  Representation of the engine status.

- typedef void * vvxtr_stdarg

  A generic argument in the XTR API.

Enumerations

- enum vvxtr_dataType { vvxtr_none = 0, vvxtr_int, vvxtr_string, vvxtr_pib, vvxtr_ptr }

  Generic XTR datatype.

- enum vvxtrResponseEnum { vvNo = 0, vvYes = 1, vvAuto = 2, vvDetect = 3, vvCorrect = 4, vvManual = 5 }

  Generic settings used for many different options.

- enum vvxtrOutTextFormatEnum {

  vvTextFormatDefault = -1, vvTextFormatNone = 0, vvTextFormatXdoc = 1, vvTextFormatXdoclite = 3, vvTextFormatXdocplus = 4, vvTextFormatPost = 13, vvTextFormatIso = 102, vvTextFormatPdf = 104,

  vvTextFormat8bit = 105, vvTextFormatUnicode = 106, vvTextFormatHtmlWysiwygP = 107, vvTextFormatHtmlWysiwygS = 108, vvTextFormatHtmlTable = 109, vvTextFormatHtmlSimple = 110 }

  Output text formats.

- enum vvxtrOutGraphicsFormatEnum {

  vvSubimageFormatNone = -1, vvSubimageFormatTiff = 0, vvSubimageFormatRas = 1, vvSubimageFormatEpsf = 2, vvSubimageFormatX11 = 3, vvSubimageFormatTiffpack = 4, vvSubimageFormatTiffg31d = 5, vvSubimageFormatTiffg32d = 6,

  vvSubimageFormatTiffg42d = 7, vvSubimageFormatTifflzw = 8, vvSubimageFormatPaltiff = 31, vvSubimageFormatGif = 17, vvSubimageFormatJpeg = 19, vvSubimageFormatPng = 21, vvSubimageFormatXwd = 22, vvSubimageFormatRgb = 23,

  vvSubimageFormatRgbrle = 24, vvSubimageFormatEPdf = 28, vvSubimageFormatVvxtrImage = 30 }

  Output graphics formats.

- enum vvxtrFocusAreaEnum { vvFocusAreaPage = 0, vvFocusAreaRegion = 1 }

  How to specify the document area on which to focus.

- enum vvxtrVerifierModeEnum { vvVerifierModeWord = 1, vvVerifierModeChar = 0 }

  Verifier mode.

- enum vvxtrRecModeEnum { vvRecModeUnspecified = 1, vvRecModeStandard = 2, vvRecModeDegraded = 3 }

  Recognition mode.

- enum vvxtrTextOutNewlineEnum { vvTextOutNewlineUnix = 1, vvTextOutNewlineMac = 2, vvTextOutNewlinePC = 3 }

  Newline designation.

- enum vvxtrPDFFormatEnum { vvPDFFormatNormal = 1, vvPDFFormatText = 2, vvPDFFormatImgOnly = 3 }

  PDF format.

- enum vvxtrRegionTypeEnum {

  vvRegionTypeAny = 3511, vvRegionTypeIgnore = 3512, vvRegionTypeText = 3513, vvRegionTypeImage = 3514, vvRegionTypeVrule = 3515, vvRegionTypeHrule = 3516, vvRegionTypeHidden = 3517, vvRegionTypeRevid = 3518,

  vvRegionTypeHiddenimage = 3519 }

  Region types.

- enum vvxtrRegionSubtypeEnum {

  vvRegionSubtypeUnflavored = 0, vvRegionSubtypeTable = 1, vvRegionSubtypeTable_inset = 2, vvRegionSubtypeHeadline = 3, vvRegionSubtypeTimestamp = 4, vvRegionSubtypeLineart = 5, vvRegionSubtypeHalftone = 6, vvRegionSubtypeInset = 7,

  vvRegionSubtypeCaption = 8, vvRegionSubtypePage_footer = 9, vvRegionSubtypePage_header = 10, vvRegionSubtypeVruling = 11, vvRegionSubtypeHruling = 12, vvRegionSubtypeNoise = 13, vvRegionSubtypeIpcorePictureMask = 14 }

  Region subtypes.

- enum vvxtrRegionGrammarModeEnum { vvRegGrammarModeWord = 0, vvRegGrammarModeLine = 1 }

  Region grammar modes.

- enum vvxtrRegionLexmodeEnum { vvRegionLexmodeNolex = 0, vvRegionLexmodePreference = 1, vvRegionLexmodeAbsolute = 2 }

  Region lexical mode.

- enum vvxtrRegionForegroundEnum { vvRegionForegroundBlack = 0, vvRegionForegroundWhite = 1, vvRegionForegroundUnknown = 2 }

  Region foreground specification.

- enum vvxtrRegionOpacityEnum { vvRegionOpacityOpaque = 0, vvRegionOpacityTransparent = 1 }

  Region opacity.

- enum vvxtrRegionAbutmentEnum { vvRegionAbutmentUnknown = -1, vvRegionAbutmentNone = 0, vvRegionAbutmentLeft = 1, vvRegionAbutmentRight = 2, vvRegionAbutmentLeftAndRight = 3 }

  Region abutment.

- enum vvxtrLexicalConstraintModeEnum { vvLexConstraintModeReplace = 0, vvLexConstraintModeAdd = 1 }

  Lexical constraint mode.

- enum vvxtrDisplayAlignmentEnum { vvAlign8 = 8, vvAlign16 = 16, vvAlign32 = 32, vvAlign64 = 64 }

  Bit alignment of display.

- enum vvxtrVersionLocationEnum { vvLocal = 0, vvRemote = 1 }

  Version location.

- enum vvxtrImageSpecificationEnum { vvInputImage = 0, vvProcessedImage = 1 }

  Specify the original or processed image.

- enum vvInternalEngineState {

  vvS_NONE = 0x00000000, vvS_IDLE = 0x00000001, vvS_SESSION = 0x00000002, vvS_CANCEL = 0x00000004, vvS_PGCAN = 0x00000008, vvS_ACQUISITION_READY = 0x00000010, vvS_ACQUIRING = 0x00000020, vvS_RECOGNITION_READY = 0x00000040,

  vvS_RECOGNIZING = 0x00000080, vvS_PREPROCESS = 0x00000100, vvS_TEXT_OUTPUT = 0x00000200, vvS_REGION_MAPPING = 0x00000400, vvS_SHAPES_LOADED = 0x00000800, vvS_AWAITING_DRAW = 0x00001000, vvS_AWAITING_VERIFIER = 0x00002000, vvS_PATTERN_LOADED = 0x00008000,

  vvS_BUFFER_FULL = 0x00010000, vvS_UNUSED_2 = 0x00020000, vvS_UNUSED_3 = 0x00040000, vvS_XDOC_LOADED = 0x00080000, vvS_PAGE_ANALYZED = 0x00200000, vvS_ANALYZING_PAGE = 0x00400000, vvS_PAGE_LAID_OUT = 0x00800000, vvS_CHARSET_TABLE_LOADED = 0x02000000,

  vvS_ANY = 0x00ffffff, vvS_ERROR = 0x01000000, vvS_BLOCKED = (vvS_AWAITING_DRAW | vvS_AWAITING_VERIFIER), vvS_READY = (vvS_ACQUISITION_READY | vvS_RECOGNITION_READY), vvS_RECOGNITION = (vvS_RECOGNITION_READY | vvS_RECOGNIZING), vvS_ACQUISITION = (vvS_ACQUISITION_READY | vvS_ACQUIRING), vvS_PROCESSING = (vvS_ACQUIRING | vvS_PREPROCESS | vvS_RECOGNIZING | vvS_REGION_MAPPING | vvS_TEXT_OUTPUT) }

  The internal engine state bit field values.

- enum vvxtrHint { vvHintLocalFilesystem, **vvNumberOfHints** }

  Hints to improve engine performance.

- enum vvxtrEngineVariables {

  dm_min_point, dm_max_point, dm_accept_thresh, dm_quest_thresh, dm_recmode, dm_recomp, dm_no_sloppy_manual, dm_no_hdr_ftr,

  dm_user_specified_order, dm_user_specified_regions, dm_find_headlines, dm_text_out_newline, dm_xdc_wconf, dm_xdc_cconf, dm_xdc_wbox, dm_xdc_cbox,

  dm_xdc_wbox_pixels, dm_metric, dm_pdf_format, dm_pdf_imgthresh, dm_pdf_img_nodict, dm_pdf_img_alphanum, dm_ls_quote, dm_rs_quote,

  dm_ld_quote, dm_rd_quote, dm_document_name, dm_questionable, dm_unrecognized, dm_force_single_col, dm_one_line_table_cells, dm_improved_single_col_detect,

  dm_region_type, dm_region_subtype, dm_region_stacking, dm_region_grammar_mode, dm_region_lexical_constraint_id, dm_region_lexmode, dm_region_foreground, dm_region_out_order,

  dm_region_name, dm_region_frame_left, dm_region_frame_right, dm_region_frame_top, dm_region_frame_bot, dm_region_uor_string, dm_region_uor_count, dm_black_threshold,

  dm_in_xres, dm_in_yres, dm_language, dm_english_chars, dm_char_set, dm_word_lexicon_id, dm_format_analysis, dm_double_dimension,

  dm_current_region, dm_focus_area, dm_pp_remove_halftone, dm_pp_auto_segment, dm_pp_rotate, dm_pp_fax_filter, dm_pp_auto_orient, dm_pp_invert,

  dm_pp_newspaper_filter, dm_pp_deskew, dm_pp_photometric_interp, dm_pp_dotmatrix_filter, dm_pp_autosetdegrade, dm_pp_recognition, dm_pp_segment_lineart, dm_pp_reverse_video,

  dm_pp_analyze_layout, dm_auto_flip, dm_in_filename, dm_in_curr_page, dm_in_num_pages, dm_in_format, dm_region_ids, dm_region_ids_text,

  dm_region_ids_image, dm_out_text_format, dm_out_graphics_format, dm_doc_memory_size, dm_subimage_memory_size, **dm_coord_space**, dm_output_img_source, dm_recognize_timeout,

  dm_preprocess_timeout, **VVXTR_ENGINE_VARIABLE_COUNT** }

Data values.

8.3.1 Detailed Description

vvxtrDefs.h, definitions and enumerations for OCRShop XTR API

This file is included by vvxtrAPI.h.

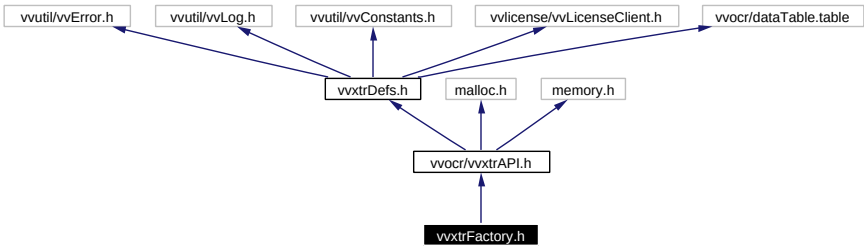
Definition in file vvxtrDefs.h.

8.4 vvxtrFactory.h File Reference

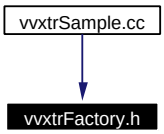
vvxtrFactory.h, Function declarations for the engine factory.

```
#include "vvocr/vvxtrAPI.h"
```

Include dependency graph for vvxtrFactory.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define **VVXTRFACTORY_H** 1

Functions

- vvEngAPI * vvxtrCreateLocalEngine ()
  Create a statically-linked (local) instance of the library.
- vvEngAPI * vvxtrCreateRemoteEngine (const char *host)
  Creates an OCR engine for the calling client program.

8.4.1 Detailed Description

vvxtrFactory.h, Function declarations for the engine factory.

This file needs to be included in a user program that uses the OCRShop XTR API.

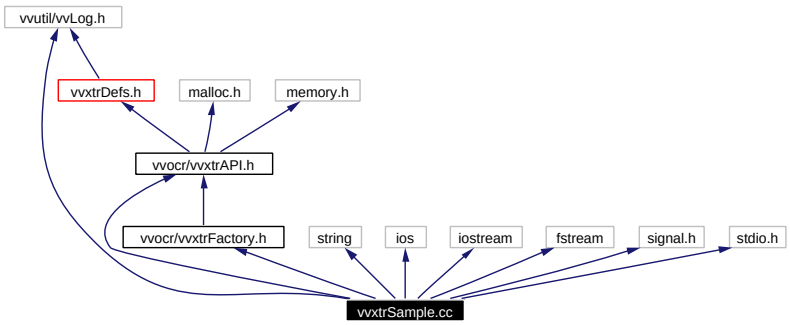
Definition in file vvxtrFactory.h.

8.5 vvxtrSample.cc File Reference

Sample file for Vividata XTR API.

```
#include <vvocr/vvxtrAPI.h>
#include <vvocr/vvxtrFactory.h>
#include <vvutil/vvLog.h>
#include <string>
#include <ios>
#include <iostream>
#include <fstream>
#include <signal.h>
#include "stdio.h"
```

Include dependency graph for vvxtrSample.cc:



Functions

- vvEngAPI * CreateEngine ()
  Function to create an OCR engine.
- vvxtrStatus runOCR (vvEngAPI *xtrEngine)
  Basic image processing example.
- vvxtrStatus processOneDocument (vvEngAPI *xtrEngine)
  Process one input document, which may have multiple pages.
- vvxtrStatus openImageDocument (vvEngAPI *xtrEngine, char *inputFilename)
  Open an input image file.
- vvxtrStatus processOnePage (vvEngAPI *xtrEngine)
  Process one page.
- vvxtrStatus outputDoc (vvEngAPI *xtrEngine)
  Output a document of OCR results (PDF, ASCII, unicode, or XDOC).

- vvxtrStatus outputImg (vvEngAPI *xtrEngine)

  Capture a subimage from the original input image, and obtain the output image through memory or in a file.

- void CloseDown ()

  Call this function on exit.

- void SigTermHandler (int param)

  SigTerm handler.

- void SigErrorHandler (int param)

  SigError handler.

- int **main** (int argc, char **argv)

Variables

- string **g_remoteEngine**
- vvEngAPI * **g_xtrEngine** = NULL

8.5.1 Detailed Description

Sample file for Vividata XTR API.

Definition in file vvxtrSample.cc.

8.5.2 Function Documentation

8.5.2.1 vvEngAPI * CreateEngine ()

Function to create an OCR engine.

This function allocates space for the new engine, which this program is subsequently responsible for deleting.

This is the only way to create an OCR engine: you will need to call the vvxtrCreateRemoteEngine function.

Returns:
A pointer to a new OCR engine (vvEngAPI)

Definition at line 171 of file vvxtrSample.cc.

References vvxtrCreateRemoteEngine().

8.5.2.2 vvxtrStatus openImageDocument (vvEngAPI * *xtrEngine*, char * *inputFilename*)

Open an input image file.

Parameters:
xtrEngine OCR engine pointer

inputFilename Name of TIFF file to open.

Returns:
vvxtrStatus

Definition at line 343 of file vvxtrSample.cc.

References dm_in_filename, vvEngAPI::vvOpenImageFile(), vvEngAPI::vvSetValue(), VVXTR_ERR, VVXTR_ERR_UNABLE_TO_LOAD_IMAGE_FILE, and vvxtrStatus.

Referenced by processOneDocument().

8.5.2.3 vvxtrStatus outputDoc (vvEngAPI * *xtrEngine*)

Output a document of OCR results (PDF, ASCII, unicode, or XDOC).

Parameters:
xtrEngine A pointer to the engine currently used in this sample program.

Returns:
vvxtrStatus

Definition at line 589 of file vvxtrSample.cc.

References dm_doc_memory_size, vvEngAPI::vvAcquireDocFile(), vvEngAPI::vvAcquireDocMemory(), vvEngAPI::vvEndDoc(), vvEngAPI::vvGetValue(), vvEngAPI::vvSpoolDoc(), vvEngAPI::vvStartDoc(), vvTextFormatIso, vvTextFormatPdf, vvTextFormatUnicode, vvTextFormatXdoc, VVXTR_ERR, and vvxtrStatus.

Referenced by processOnePage().

8.5.2.4 vvxtrStatus outputImg (vvEngAPI * *xtrEngine*)

Capture a subimage from the original input image, and obtain the output image through memory or in a file.

Parameters:
xtrEngine A pointer to the engine currently used in this sample program.

Returns:
vvxtrStatus

Definition at line 428 of file vvxtrSample.cc.

References dm_focus_area, dm_out_graphics_format, dm_region_ids, dm_region_ids_image, dm_region_ids_text, dm_subimage_memory_size, vvxtrImage::FromRaw(), vvEngAPI::vvAcquireSubimageFile(), vvEngAPI::vvAcquireSubimageMemory(), vvEngAPI::vvCaptureSubimage(), vvFocusAreaPage, vvFocusAreaRegion, vvEngAPI::vvGetValue(), vvEngAPI::vvSetValue(), vvSubimageFormatEPdf, vvSubimageFormatGif, vvSubimageFormatJpeg, vvSubimageFormatPng, vvSubimageFormatRgb, vvSubimageFormatTiffpack, vvSubimageFormatVvxtrImage, VVXTR_ERR, and vvxtrStatus.

Referenced by processOnePage().

8.5.2.5 vvxtrStatus processOneDocument (vvEngAPI * *xtrEngine*)

Process one input document, which may have multiple pages.

This function encompasses reading in the file, pre-processing, recognition, and document as well as subimage output.

Parameters:
xtrEngine OCR engine pointer

Returns:
vvxtrStatus

Definition at line 263 of file vvxtrSample.cc.

References dm_in_curr_page, dm_in_num_pages, openImageDocument(), processOnePage(), vvEngAPI::vvCloseImageFile(), vvEngAPI::vvGetValue(), vvEngAPI::vvReadImageData(), vvEngAPI::vvSetValue(), vvEngAPI::vvUnloadImage(), VVXTR_ERR_INVALID_PAGE, and vvxtrStatus.

Referenced by runOCR().

8.5.2.6 vvxtrStatus processOnePage (vvEngAPI * *xtrEngine*)

Process one page.

This function preprocesses and recognizes one page of image data. It then presents the user with the option of generating output of the recognized text or creating subimages from the original image.

Output documents can cover multiple pages and multiple input documents, but for the purposes of this example, we limit output documents to one page of recognized text.

Parameters:
xtrEngine OCR engine pointer

Returns:
vvxtrStatus

Definition at line 385 of file vvxtrSample.cc.

References outputDoc(), outputImg(), vvEngAPI::vvPreprocess(), vvEngAPI::vvRecognize(), VVXTR_ERR, and vvxtrStatus.

Referenced by processOneDocument().

8.5.2.7 vvxtrStatus runOCR (vvEngAPI * *xtrEngine*)

Basic image processing example.

This routine prompts a user to submit an image file for recognition, then the user can choose to output a text or subimage results. An arbitrary number of input images may be recognized, and the user can choose which page to recognize. All OCR takes place within one OCR session (vvEngAPI::vvStartOCRSes), so for the best results, all input images recognized within one run of this program should be from the same batch or source, so that the internal training within the OCR session is productive.

Returns:
vvxtrStatus

Definition at line 191 of file vvxtrSample.cc.

References dm_format_analysis, dm_pp_auto_segment, dm_pp_segment_lineart, processOneDocument(), vvEngAPI::vvEndInstance(), vvEngAPI::vvEndOCRSes(), vvHintLocalFilesystem, vvEngAPI::vvInitInstance(), vvEngAPI::vvInitValues(), vvEngAPI::vvSetHint(), vvEngAPI::vvSetValue(), vvEngAPI::vvStartOCRSes(), vvxtrStatus, and vvYes.

Chapter 9

OCR Shop XTR/API User Documentation Page Documentation

9.1 OCR Shop XTR/API Glossary

ADF	Automatic document feeder
ASCII	An acronym for American Standard Code for Information Interchange. A code in which the numbers from 0 to 127 stand for text characters. ASCII code is used for representing text inside a computer and for transmitting text between computers or between a computer and a peripheral device.
auto segmentation	The process in which the OCR Shop XTR determines where on a page different elements are such as where pictures are and where columns of text are. Divides the page up into regions.
binary image	A image that is represented using only one bit per pixel. Such images are also called black and white, monochrome, bi-level, or 1-bit.
bitmapped image	A collection of bits (dots) in memory that represent the scanned image. The display on the screen is a visible bitmapped image.
chunky bitmapped image	A bitmap that has an internal structure where all of the data for a particular pixel is stored in a ”chunk”, i.e. a contiguous set of bits. For example, a 24-bit chunky image has sets of three contiguous bytes that represent each pixel.
Code Page	Code Page is a Microsoft174 term. A code page is a particular mapping of a set of unsigned bytes to a set of visible characters (and space characters). Different code pages are used to represent in memory the characters in different languages. See http://www.microsoft.com/globaldev/reference/WinCP.asp for more details. Also see shape pack
compound document	A compound document is a set of one or more pages that consists of a mixture of text and images, for example pdf or html.
conversion filter	A program that translates one file format into another. For example, the mpage conversion filter can translate an ASCII file into a PostScript file.
digital image	A digital image is the way a picture or visual image of some object is represented in computer

9.2 OCR Shop XTR/API Frequently Asked Questions

Installation and set-up

Processing and Recognition

Custom regions

File Formats

Installation and set-up

1. **Why is the API not installed with my other Vividata softare?**

   Software using Vividata's older installer was installed by default in /usr/vividata on Linux and /opt/Vividata on Solaris. The new installer used for the API always installs in /opt/Vividata by default.

   You may control where the API installer places the API by setting the environment variable `VV_HOME` to the desired directory prior to running the installer.

   If you use some of Vividata's older software, you may already have the `VV_HOME` environment variable set in your environment. In this case, you should be careful when you install the API and later when you start the ocrxtrdaemon. If `VV_HOME` is inconsistent, then you will receive an error. To fix it, make sure `VV_HOME` matches the API installation directory when you start the ocrxtrdaemon.

2. **How do I generate log output?**

   Various levels of log output may be generated on both the client and daemon sides as output to `stdout` and `stderr`.

   To generate log output from your client program, call the function `vvLogSetLevel` (see an example in vvxtrSample.cc):

   ```
   void vvLogSetLevel( int logLevel );
   ```

   The log level should be set anywhere from 1 to 1000, where the higher the number, the more output will be generated:

- 1 - Error messages only
- 250 - Warning messages
- 500 - Informational messages
- 510 - More informational messages
- 515 - Debug information
- 520 - All information

By default, only error messages are printed.

Similarly, the OCR Shop XTR/API daemon (ocrxtrdaemon) can generate log output to `stdout` and `stderr`. Control the verbosity of the daemon log output by setting the environment variable VV_DEBUG to the desired log level before starting the ocrxtrdaemon process. For example, set VV_DEBUG to `1000` for the maximum debugging output.

For both the client and the daemon, you can save the log output to a file by piping it from the command line:

```
clientProgram >& client.log
ocrxtrdaemon >& daemon.log
```

3. **How do I customize where temp files are placed?**

   To make sure all temp files are placed in the directory of your choice, set these environment variables:

   - TMP
   - TMP_DIR
   - VV_TMPDIR

   to the directory where you wish to store the temp files. Make sure that you do this in the shell where you run the daemon program ocrxtrdaemon.

Processing and Recognition

1. **How can I improve recognition accuracy?**

   Typeset, high-quality printed pages return the best recognition accuracy. The following factors most affect text-recognition accuracy:

   - Preprocessing Settings
   - Recognition Parameters
   - Line Art and Photographic Regions
   - Document Quality
   - Scanning Process

   No single combination of preprocessing settings and recognition parameters always results in the quickest, most accurate recognition job. However, if you use the settings most appropriate to each document, OCR Shop XTR™/API's speed and accuracy will be maximized.

   OCR Shop XTR™/API may recognize some line-art graphics or areas of photographic regions as text if the artwork is poor and the lines resemble letter strokes. Adjusting the `dm_black_threshold` parameter may change how the OCR Engine differentiates between photographic regions and text regions. Individual regions can also be manually specified as graphical or textual content.

   OCR Shop XTR™/API recognizes characters in almost any font in sizes from 5 to 72 points. The engine interprets font size based on the image's dpi, so set the input image dpi carefully in order to guarantee the image's fonts fall within the recognized sizes.

   Following certain guidelines may improve recognition accuracy:

- The print should be as clean and crisp as possible.
- Characters should be distinct, separated from each other and not blotched together or overlapping.
- The document should be free of handwritten notes, lines and doodles.
- Anything that is not a printed character slows recognition, and any character distorted by a mark will be unrecognizable.
- Try to avoid highly stylized fonts. For example, OCR Shop XTR™may not recognize text in the Zapf Chancery174 font accurately.
- Try to avoid underlined text. Underlining changes the shape of descenders on the letters q, g, y, p, and j.

If you have control over the scanning process, you can improve recognition by eliminating skew and background noise. Some paper is so thin that the scanner reads text printed on the back side of the scanned page. Put a black piece of paper between the sheet and the lid of the scanner. By eliminating any need for the OCR Engine to deskew an image, recognition processing speed will improve.

2. **How can I improve performance?**

   Here are several items to consider which affect how fast the OCR Shop XTR/API processes your images:

   - One of the primary benefits of using the API is that the OCR engine does not need to be shut down and restarted between each page. Make sure that you do not needlessly shut down and restart the OCR engine and OCR session. Unless you want to switch languages, you should be able to recognize an unlimited number of pages within one OCR session.
   - The OCR Shop XTR/API works using a daemon to handle the OCR processing. This allows for the flexibility such that the daemon and client program do not have to run on the same system. However, if the OCR daemon and the client program do reside on the same filesystem, you should tell the daemon this in your client program so that it can optimize communications. Before starting the OCR session, make this function call to the OCR engine:

     ```
     vvERROR(xtrEngine->vvSetHint(vvHintLocalFilesystem));
     ```

     See `vvEngAPI::vvSetHint` and `vvHintLocalFilesystem`.
   - Preprocessing takes up a large portion of the total OCR processing time, and some of the preprocessing functions are among the most processor intensive functions. Carefully pick and choose which preprocessing options you have turned on if you are concerned about time. And pay attention to the default settings; the defaults provide the best OCR results under most conditions, not the best balance of fast processing time and results in your particular circumstances. For example, if you know all of your images will be properly rotated, turn off `dm_pp_auto_orient`. If you know the quality of your images is high, turn off degraded image processing `dm_pp_autosetdegrade`. If you are certain you do not want to use the fax filter, make sure it is off and not set to automatic (`dm_pp_fax_filter`. See `dm_pp_remove_halftone` and the other preprocessing options.
   - One particular preprocessing option that can take up significant time is deskewing. If your images are straight to start with, processing will be faster and you can turn off deskewing. If some or all of your images will be skewed, understand that it will take longer to process them because of the skew. See `dm_pp_deskew`

3. **What is the correct sequence of actions for processing multiple files?**

   When processing multiple input files, the sequence of operations is, for example:

   - vvEngAPI::vvxtrCreateRemoteEngine

- vvEngAPI::vvInitInstance
- vvEngAPI::vvStartOCRSes
- vvEngAPI::vvInitValues
- *For each file:*
  - vvEngAPI::vvOpenImageFile
  - *Set input page number (default is the first page).*
  - vvEngAPI::vvReadImageData
  - *Set options for preprocessing.*
  - vvEngAPI::vvPreprocess
  - *Set options for recognition.*
  - vvEngAPI::vvRecognize
  - vvEngAPI::vvStartDoc
  - vvEngAPI::vvSpoolDoc
  - vvEngAPI::vvEndDoc
  - vvEngAPI::vvAcquireDocFile
  - vvEngAPI::vvUnloadImage
  - vvEngAPI::vvCloseImageFile
- vvEngAPI::vvEndOCRSes
- vvEngAPI::vvEndInstance

Note on vvxtrCreateRemoteEngine:

vvEngAPI::vvxtrCreateRemoteEngine is different from vvEngAPI::vvInitInstance, vvEngAPI::vvRecognize, etc. because it is actually starting a new engine. It tells the ocrxtrdaemon to fork a new process that becomes the new engine. The ”action” functions such as vvEngAPI::vvInitInstance and vvEngAPI::vvRecognize work within that engine and cause the engine to change state.

You can call vvEngAPI::vvxtrCreateRemoteEngine multiple times to create multiple engines that all run concurrently. Each engine has its own state and is used individually. A call to vvEngAPI::vvStartDoc, for example, is made to one specific engine.

When to set options:

Options for preprocessing and recognition do not have to be set just before the vvEngAPI::vvPreprocess and vvEngAPI::vvRecognize calls. After being set, the values are retained in the engine until the ocr session is ended or vvEngAPI::vvInitValues is called.

Reading image data from memory:

If you are reading your image data from memory, then you do not need to call vvEngAPI::vvOpenImageFile or vvEngAPI::vvCloseImageFile. You just need to call vvEngAPI::vvReadImageData and vvEngAPI::vvUnloadImage.

Ordering of output actions versus input actions:

The sequence of actions used to start, write and close the output document do not have to take place in the exact order above – the output document is flexible with respect the input document. Recognition must take place before vvEngAPI::vvSpoolDoc may be called, but otherwise vvEngAPI::vvStartDoc can be called any time between vvEngAPI::vvStartOCRSes and vvEngAPI::vvSpoolDoc, and vvEngAPI::vvEndDoc may be called any time between vvEngAPI::vvSpoolDoc and vvEngAPI::vvEndOCRSes. vvEngAPI::vvSpoolDoc may be called multiple times to write multiple output pages.

For more information:

See `this page` for a list of the basic sequence of actions, further description of handling data, input and output, and actions.

Further down on the same page, the State table for actions describes in detail how the engine state works. Many of the actions can be considered stack-like: after you start an output document with vvEngAPI::vvStartDoc, you must close it with vvEngAPI::vvEndDoc before you can exit the OCR session; after you load image data into the engine with vvEngAPI::vvReadImageData, you must unload it with vvEngAPI::vvUnloadImage before you can load any new image data into the engine. Actions such as vvEngAPI::vvPreprocess and vvRecognize are a little different; they may be called multiple times and must obey a certain ordering – vvEngAPI::vvPreprocess must be called before vvEngAPI::vvRecognize, vvEngAPI::vvRecognize must be called before vvEngAPI::vvSpoolDoc.

Custom regions

1. **How do I tell the engine to not divide the image into regions?**

   Before calling `vvEngAPI::vvPreprocess`, make sure you set the value `dm_pp_auto_segment` to `vvNo`.

2. **How do I create my own regions?**

   When the engine runs preprocessing on an image during the `vvEngAPI::vvPreprocess` call, the engine will auto segment the input image if the preprocessing value `dm_pp_auto_segment` is set to `vvYes`, or it will not divide the image into regions if this value is set to `vvNo`. In either case, you can create user defined regions.

   To get a list of the current regions, get the value of `dm_region_ids` from the engine by using the `vvEngAPI::vvGetValue` function.

   To create a new region:

   - Set `dm_current_region` to an unused region id number.
   - Set up the properties of the new region using the `vvEngAPI::vvGetValue` function. The minimal set of values you should specify is:
     - `dm_region_uor_string` (See What is a UOR?.)
     - `dm_region_uor_count`
     - `dm_region_type`

     See the other values starting with ”<code>::dm_region</code>” for other region properties you can specify.
   - Finish setting up this new region in the OCR engine by calling the function `vvEngAPI::vvSetRegionProperties`.
   - Now if you query the engine again for the list of `dm_region_ids`, you should see your new region listed.

   A sample program to demonstrate creation of a new region is available upon request.

   Note that regions may also be deleted; see the function `vvEngAPI::vvRemoveRegion`.

3. **What is a UOR?**

   ”\ref UOR” stands for ”union of rectangles” and is used to describe the bounding box of a region. The value `dm_region_uor_string` defines the UOR for a region.

   The UOR for a region may include one or more rectangles. Use of multiple rectangles permits oddly shaped regions, important for documents where text and images appear closely together, in such a way that one rectangle cannot encompass an entire text region without including part of what should be an image region.

   To set the UOR for a region:

   Using the function `vvEngAPI::vvSetValue`, you must set the current region (`dm_current_region`), then set the UOR definition (`dm_region_uor_string`) and the number of rectangles

(dm_region_uor_count). Finally, the region information is committed in the OCR engine with a function call to vvEngAPI::vvSetRegionProperties.

Formatting the UOR string:

The UOR string (dm_region_uor_string) must be formatted correctly for your application to work correctly, and the region count (dm_region_uor_count) must be set accurately. In dm_region_uor_string, coordinates are separated by commas; rectangles are separated by semicolons; the string should contain no whitespace.

For example, if you want a region to consist of one rectangle with the coordinates (400,800) by (600,1400), then you would set dm_region_uor_string to `400,800,600,1400` and dm_region_uor_count to `1`.

In general, the format of the dm_region_uor_string should be:

`x1,y1,x2,y2;x3,y3,x4,y4;x5,y5,x6,y6`

to specify three rectangles defined conceptually:

rectangle one: (x1,y1) (x2,y2)

rectangle two: (x3,y3) (x4,y4)

rectangle three: (x5,y5) (x6,y6)

File Formats

1. **What are image PDFs versus text PDFs?**

   A PDF document consists of any combination of text and bitmap images embedded in a PDF file. It may also contain structural information used for formatting and interactive features such as hyperlinks.

   Because of the flexibility of the PDF file format, a PDF file may be used as an ”image” file or as a ”text” file. When used as an ”image”, PDF files are commonly used as optical character recognition (OCR) input, and when used as a ”text” document, PDF files are often used as OCR output.

   OCR is the process of converting image bitmap data into text data, so it should be clear which type of PDF files are appropriate as OCR input formats and OCR output formats.

   In general, one may identify three types of PDF documents:

   - ***Image-only PDF***

     Image-only PDF documents contain only a bitmap of a document and are produced by encapsulating a bitmap image in a ”pdf wrapper.” The result is an exact representation of the bitmap image.

     Image-only PDF file size is large because it consists solely of bitmap image data.

     Image-only PDF documents contain no searchable text; they may not be indexed and the text may not be copied.

     This format is used for OCR input, because it contains bitmap data and no text data.

   - ***Normal PDF***

     Normal PDFs contain text and embedded graphical elements. The text is scalable and can be searched, copied, and indexed.

     Normal PDF file size is small, because most data is textual and embedded graphics are usually small. Because text data is stored in normal PDF documents, the clarity is good due to scalable text, and the text is searchable.

     Normal PDFs must be generated by some sort of editor or an OCR application; they can not be generated directly from a scanner.

Normal PDFs are a common OCR output format, because they may contain recognized text and approximate the original bitmap image's formatting and embedded graphics. See the output format vvPDFFormatNormal.

Normal PDFs are not normally used for OCR input, because they already contain text data and therefore do not need to be recognized.

- ***Image+text PDF***

  Image+text PDFs are a hybrid between Normal PDFs and Image only PDFs and are used because they combine the best features of both. Like Image-only PDFs, Image+text PDFs display the entire original bitmap; everything visible in an Image+text PDF is bitmap data. However, Image+text PDFs also contain an invisible layer of text beneath the visible bitmap.

  Image+text PDF file size is large, because it contains a full-page bitmap.

  Image+text PDF text may be searched, copied, and indexed.

  Image+text PDFs are a common OCR output format, because they contain recognized text, allowing them to be searched and indexed, while at the same time they retain the exact appearance of the original scanned bitmap image. See the output format vvPDFFormatText.

  Image+text PDFs are not used for OCR input, because they already contain text data and therefore do not need to be recognized.

Note that Vividata's OCR applications will accept many Normal PDFs and Image+text PDFs as input. However, this can result in information loss, because the OCR application renders the input PDF file from text into bitmap data, then performs OCR on the bitmap data in order to convert it back to text. As a result, we do not recommend using Vividata's OCR applications to extract text from Normal PDFs and Image+text PDFs, and would instead suggest using a utility such as "pdftotext" to directly pull text data from a text PDF.

2. **What is the XDOC format and how do I use it?**

   The XDOC format is a ScanSoft text output format which provides detailed information about the text, images, and formatting in a recognized document.

   To use XDOC output, set the output document format to one of the following types of XDOC output:

   - vvTextFormatXdoc - Enhanced XDOC format
   - vvTextFormatXdoclite - XDOC format with no format analysis
   - vvTextFormatXdocplus - XDOC format with style sheet data

   The following values are associated with XDOC output and can be set to affect the information written to XDOC output:

   - dm_xdc_wconf
   - dm_xdc_cconf
   - dm_xdc_wbox
   - dm_xdc_cbox
   - dm_no_hdr_ftr
   - dm_quest_thresh
   - dm_accept_thresh
   - dm_xdc_wbox_pixels

   These files, included with the API in /opt/Vividata/doc, provide specific information about the XDOC format, and enough detail for a user to parse the output.

9.3 OCR Shop XTR/API Sample Usage

A sample file vvxtrSample.cc is available to demonstrate how to use the OCR Shop XTR™/API.

After you have compiled the sample code (see section Basics of Usage), here is how to run the sample program and what to expect:

1. Start the OCR Shop XTR™/API daemon in one shell:

   ```
   /opt/Vividata/bin/ocrxtrdaemon
   ```

   It should start and not do anything; we recommend you leave it in the foreground for debugging so you can watch the log and error messages.

2. Run the sample program in another shell with the command:

   ```
   ./vvxtrSample localhost
   ```

3. The sample program will prompt you for a file name; enter:

   ```
   letter.tif
   ```

4. It will then prompt you to enter the page number; letter.tif only has one page, so enter:

   ```
   1
   ```

5. The sample program runs preprocessing and recognition on the input image letter.tif, then asks you if you want to output a document. To write an output document, enter:

   ```
   y
   ```

6. Now choose a format; we suggest initially writing an ASCII output document, so enter:

   ```
   a
   ```

7. With the API and in the sample program, it is possible to acquire recognition output from the engine by asking the engine to write a file or by asking the engine to pass the output to the client application through memory. In this example, send the output to memory by entering:

   ```
   m
   ```

8. The sample program writes the output text to the screen. You should see all of the text contained within the letter.tif image; you can open letter.tif in an image viewer if you would like to confirm the expected output.

9. The sample program now asks if you would like to output an image. Enter:

   ```
   y
   ```

10. Choose an output format. For this example, we suggest you choose JPEG by entering:

    ```
    j
    ```

11. The sample program now lists all regions in the recognized image. The regions were created when the sample program called `vvEngAPI::vvPreprocess`, because the `dm_pp_auto_segment` value was turned on. Enter any region number listed to output that region as an image; for example, enter:

    ```
    3
    ```

12. As with document output, subimage output can be acquired from the engine by sending the data directly to a file or by having the engine pass the image data through memory. For example, choose to have the output written directly to a file by entering:

    ```
    f
    ```

13. You can write out other regions as images if you like, or to finish, enter:

    ```
    n
    ```

14. The sample program should end, with the last message:

    ```
    ******* Ending engine instance.
    ```

15. You may view the region that you wrote as an image file by opening the file ”outImg_3” in an image viewer.

Index