PENTACON Developer Kit
Reference Manual

Version 3.0
July 2006

Copyrights and Trademarks

Contents

List of Tables and Figures

1 PENTACON Developer Kit License Agreement

IMPORTANT - READ THIS CAREFULLY BEFORE COPYING, INSTALLING OR USING.

Do not copy, install, or use the "Materials" provided under this license agreement ("Agreement"), until you have carefully read the following terms and conditions.

By copying, installing, or otherwise using the Materials, you agree to be bound by the terms of this Agreement. If you do not agree to the terms of this Agreement, do not copy, install, or use the Materials.

PENTACON Developer Kit License Agreement

LICENSE GRANT: Subject to the License Restrictions below, PENTACON Corporation ("PENTACON") grants to you the following non-exclusive, non-assignable royalty-free copyright licenses in the "Materials" below, which are identified specifically in License Definitions at the end of this Agreement, and in any updates thereto that PENTACON may offer in the future.

Developer Tools include developer documentation, installation or development utilities, and other materials. You may use them internally for the purposes of using the Materials as licensed hereunder, but you may not redistribute them.

Sample Source may include example interface or application source code. You may copy, modify and compile the Sample Source and distribute it in your own products in binary form.

Redistributable Code may be provided to you in either source or binary/object code form or both. You may copy, modify and compile Redistributable Code that you receive in source form. You may not modify Redistributable Code that you receive in binary/object code form. You may distribute Redistributable Code in your own products only in binary/object code form provided that your product adds significant and primary functionality to the Redistributable Code.

LICENSE RESTRICTIONS: You may not reverse-assemble, reverse-compile, or otherwise reverse-engineer any software provided solely in binary form.

Upon PENTACON's release of an update, upgrade or new version of the Materials, you will make reasonable efforts to discontinue distribution of the enclosed Materials and you will make reasonable efforts to distribute such updates, upgrades or new versions to your customers who have received the Materials herein.

Distribution of the Materials is also subject to the following limitations: You (i) shall be solely responsible to your customers for any update or support obligation or other liability which may arise from the distribution, (ii) do not make any statement that your product is "certified," or that its performance is guaranteed, by PENTACON, (iii) do not use PENTACON's name or trademarks to market your product without written permission, (iv) shall prohibit disassembly and reverse engineering, (v) shall not publish reviews of Materials designated as beta without written permission by PENTACON, and (vi) shall indemnify, hold harmless, and defend PENTACON and its suppliers from and against any claims or lawsuits, including attorney's fees, that arise or result from your distribution of any product.

COPYRIGHT: Title to the Materials and all copies thereof remain with PENTACON or its suppliers. The Materials are copyrighted and are protected by International copyright laws and treaty provisions. You will not remove any copyright notice from the Materials. You agree to prevent any unauthorized copying of the Materials. Except as expressly provided herein, PENTACON does not grant any express or implied right to you under PENTACON patents, copyrights, trademarks, or trade secret information.

REPLACEMENTS: The Materials are provided "AS IS" without warranty of any kind. If the media on which the Materials are furnished are found to be defective in material or workmanship under normal use for a period of thirty (30) days from the date of receipt, PENTACON's entire liability and your exclusive remedy shall be the replacement of the media. This offer is void if the media defect results from accident, abuse, or misapplication.

USER SUBMISSIONS: You agree that any material, information or other communication, including all data, images, sounds, text, and other things embodied therein, you transmit or post to a PENTACON website will be considered non-confidential ("Communications"). PENTACON will have no confidentiality obligations with respect to the Communications. You agree that PENTACON and its designees will be free to copy, modify, create derivative works, publicly display, disclose, distribute, license and sublicense through multiple tiers of distribution and licensees, incorporate and otherwise use the Communications, including derivative works thereto, for any and all commercial or non-commercial purposes.

LIMITATION OF LIABILITY: THE ABOVE REPLACEMENT PROVISION IS THE ONLY WARRANTY OF ANY KIND. PENTACON OFFERS NO OTHER WARRANTY EITHER EXPRESS OR IMPLIED INCLUDING THOSE OF MERCHANTABILITY, NONINFRINGEMENT OF THIRD-PARTY PENTACONLECTUAL PROPERTY OR FITNESS FOR A PARTICULAR PURPOSE. NEITHER PENTACON NOR ITS SUPPLIERS SHALL BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF PENTACON HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME JURISDICTIONS PROHIBIT THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

TERMINATION OF THIS LICENSE: PENTACON may terminate this license at any time if you are in breach of any of its terms and conditions. Upon termination, you will immediately destroy the Materials or return all copies of the Materials to PENTACON along with any copies you have made.

LICENSE DEFINITIONS: The Materials are classified as follows and are located in the following directories located on your system after the installation has successfully been completed:

- Developer Tools can be found in the directories **include**, **lib** and **doc**.
- Sample Source Code can be found in the directories **examples**.
- Redistributable Code can be found in the directories **bin** and **drivers**.

2 What's New

2.1 What's New in Version 3.0

Video Capture Library

- New support of video capture cards based on **Cx2388x** chip;
- New drivers package, version 4.1.0.7 (see "Drivers\Pentacon devices" directory);

DVPack™ Library

- New support of YUV411 Planar video format

2.2 What's New in Version 2.8

Video Capture Library

- New Plug-n-Play drivers for video capture boards (version 4.0.7.3);
- Deinterlace filter was added. The filter was designed to remove interlace from video (effect, that appear at joining of two half-frame into one). For working with deinterlace it is necessary to get the corresponding license for your HASP key.

DVPack™ Library

- No changes

2.3 What's New in Version 2.7

Video Capture Library

- The audio capture functions for Bt8xx devices were added. Audio capturing is performed in asynchronous mode (via busmaster DMA), similarly to video capturing. Digitized audio samples can be 8 or 16 bits and can be scaled to any arbitrary frequency (samples per second) up to 100 KHz. See chapter 6.1.7 for more information.

- The purpose of **RvclCancelIo** function was changed: since version 2.7 it cancels both video and audio asynchronous capture operations. To cancel only video capture operations for the specified device call **RvclCancelVideoCapture**. To cancel only audio capture operations call **RvclCancelAudioCapture**.

DVPack™ Library

- No changes.

2.4 What's New in Version 2.5

Video Capture Library

- All **RVCL_CF_XXX** constants are renamed to **RDN_CF_XXX**.

DVPack™ Library

- Since Version 2.5, PENTACON Developer Kit contains: (1) Video Capture Library, (2) DVPack™ Library.

2.5 What's New in Version 2.0

Video Capture Library

- The function of the quick channels switching with fields polarity saving was added. For that, the **dwVideoSource** field was added to the **SRvclFrame** structure. Now, at video capturing there is a possibility to set at once the video input number (0…3) from which the video image is to be received, without calling the **RvclSetVideoSource**. The RVCL library automatically supports the video signal field's synchronization on all Bt8xx device inputs. So, the problem with the fields synchronization bug (image jitter) at switching was eliminated.

- The YUV 4:2:2 (Packed) and YUV 4:1:1 (Packed) digital capture formats were added. At using these formats for digital capturing of color images the data stream, transmitted on bus, and buffers' size reduce in 1.5 and 2 times accordingly in the comparison to the RGB 24 format. See chapter 6.1.5 for more information.

- Since Version 2.0, every software developer, using PENTACON Developer Kit, is assigned by the unique identifier – customer code. Customer code is stored in the HASP key. The PENTACON CORPORATION technical support services inform every developer of the password, which is necessary for having the access to developer's key.

  Customer code and customer passwords ensure that all HASP keys, received from PENTACON Corporation by the developer, will use nobody but the developer. This will extend the protection safety of the programs based on PENTACON Developer Kit.

  The new **RvclInitializeEx** initialization function has been added to the RVCL library to support working with customers' passwords.

- Since Version 2.0, every HASP key, supplied by PENTACON Corporation for working with PENTACON Developer Kit, contains the data segment for user data storing. Developers are able to store any data in the user data area, which is necessary for their software. The **RDevKit** library has been added to the PENTACON Developer Kit to provide the programming of the key's user data area. In addition, the **RvclReadKeyData** function has been added to the RVCL library to provide reading data from the user data area.

- The sample code on C++ intended for the **PowerVN4** boards definition among all the Bt8xx devices installed on the computer was included in to the library description (*Appendix B: PowerVN4 Code Samples*).

- New sample **MultiScreen** was added. This sample illustrates: (a) stable video inputs switching, (b) YUV formats support.

- New sample **KeyEdit** was added. This sample illustrates reading and modification of the data from the user data area of the HASP key.

2.6 What's New in Version 1.3

Video Capture Library

- Watchdog Timer placed on the **PowerVN4** card.

3 Software Installation

3.1 Developer Kit Installation

To install PENTACON Developer Kit run **setup.exe** from the CD disc supplied. During the installation following directories will be created in the destination directory:

Table 3-1 Developer Kit Directories

Directory		Contents
bin		Binary files.
drivers	hasp	HASP key device driver files.
	Pentacon devices	All driver files for video/audio capture devices of **Power, Titan** and **Tiny** families by Pentacon based on **Bt8xx** and **Cx2388x** chips compatible with *Windows 2000/XP/Server 2003*.
include		C-language header files.
lib		Static library files (Microsoft Visual C format).
examples		Developer Kit sample applications. Use Microsoft Visual C++ and MFC library.
doc		Developer Kit documentation.

To install device drivers for video/audio capture devices of **Power, Titan** and **Tiny** families refer to "End-User Installation" chapter of this document.

3.2 End-User Installation

To install **Video Capture Library** on the end-user computer:
1. Install the HASP key device driver (chapter 3.2.1).
2. Install the video capture card device driver (chapter 3.2.2).

To install **DVPack™ Library** on the end-user computer:
1. Install the HASP key device driver (chapter 3.2.1).
2. Copy Intel Performance Library files into program folder or Windows system folder.

List of IPL files:

```
Cpuinf32.dll
ipl.dll
ipla6.dll
iplm5.dll
iplm6.dll
iplp6.dll
iplpx.dll
iplw7.dll
```

3.2.1 HASP Key Device Driver Installation

To install (remove) the HASP key device driver using the utility, run **hinstall.exe** from **drivers\hasp** directory with one or more of following options:

Table 3-2 HASP Install Flags

Option	Action
-i	Install the HASP device driver.
-r	Remove the HASP device driver.
-criticalmsg	Display only critical messages, such as the instruction to reboot.
-nomsg	Disable message display. No messages are displayed when this switch is used.

To get more information about **hinstall.exe** utility, run it without any options. You can run **hinstall.exe** utility automatically from your installation program when installing (uninstalling) software on the end-user computer.

Note:
When you install USB HASP key into your computer and switch it on, Windows 2000/XP/Server2003 will detect new hardware. Cancel the **Found New Hardware Wizard**, run **hinstall** with "**-i**" option and reboot your computer.

3.2.2 Video/audio Capture Devices Drivers Installation for Windows 2000/XP/Server2003

When you install video/audio capture card(s) (**Power, Titan** or **Tiny** family) into your computer and switch it on, Windows 2000/XP/Server 2003 will detect new hardware. Opens '**Found New Hardware Wizard**'. It will offer you to install device drivers.

Attention! Before clicking wizard 'Next' button, decide which way from two available you want to perform drivers' installation.
You can perform *automatic installation* or *manual* installation (using standard operational system wizard).

Automatic drivers installation (recommended)
You can perform automatic installation from your installation CD:

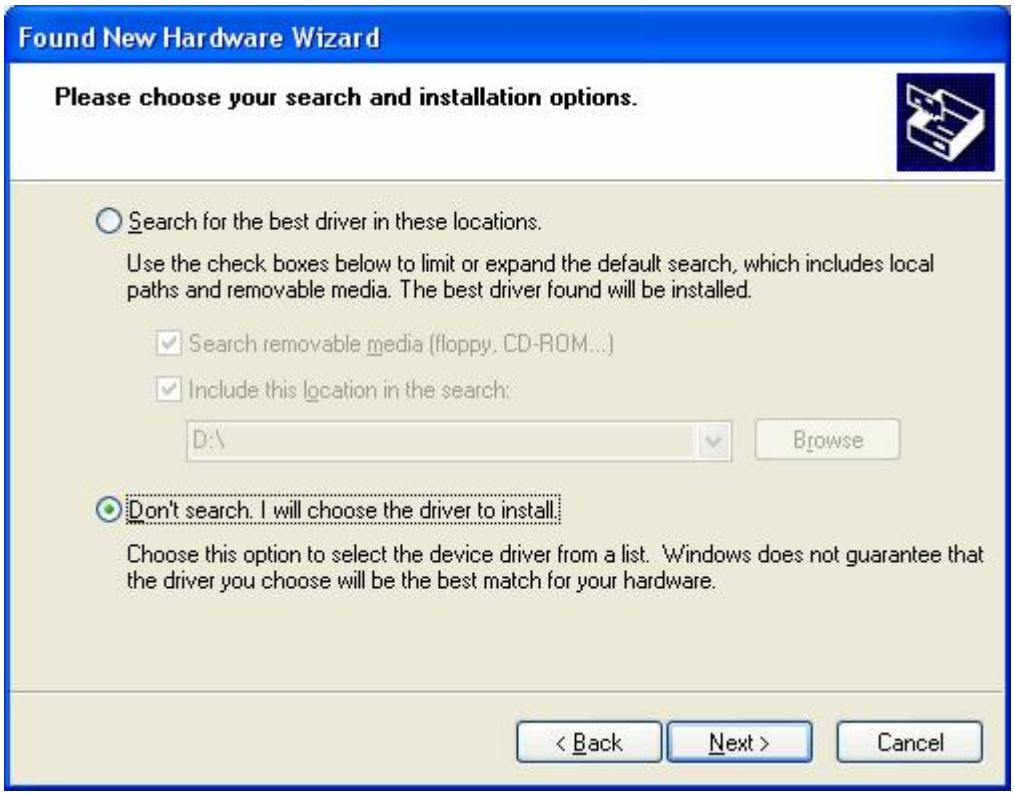
1. Do not click 'Next' button in 'Found New Hardware wizard' and do not close it. Insert installation CD in your CD-ROM.
2. Run Setup.exe from CD root.
3. Select language for working with setup application.
4. Select **Drivers** item.

5. Click **Install**.
6. Opens Drivers Installation wizard.
7. Follow wizard instructions.

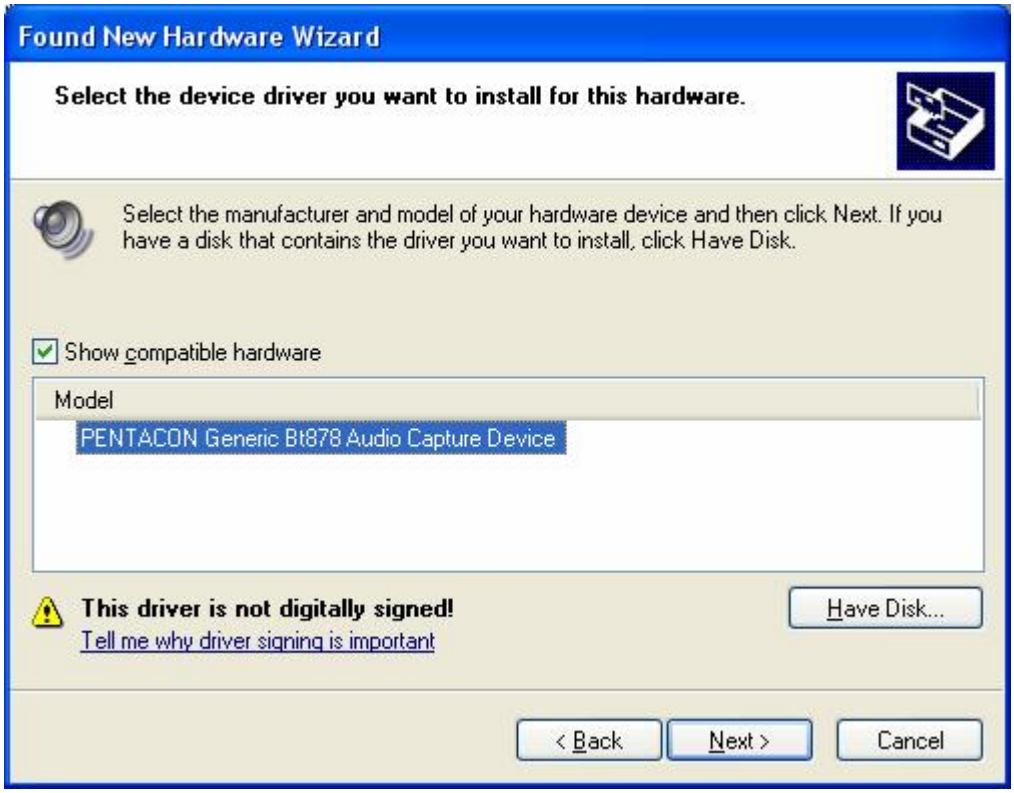
Manual drivers installation (recommended)

You can perform *manual* drivers installation using standard 'Found New Hardware Wizard':

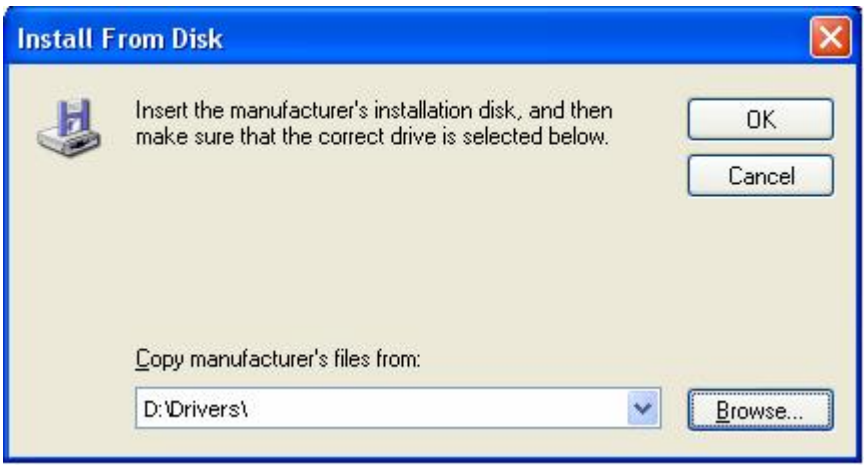
1. Select **Install from a list or specific location (Advanced)** option and click 'Next' button to open **Please choose your search and installation options** page. Select **Don't search. I will choose the driver to install** option and click 'Next' button.



2. Select **PENTACON Generic Bt878/Cx2388x Audio Capture Device** model from list and click 'Have a disk' button.



3. Opens **Install from disc** dialog.



Specify the path to your CD-ROM with installation CD in it or to the place with its copy on your hard drive or in network. Click **OK** to go to next page. Click 'Next' button.

Drivers set up is completed.

4 HASP Key

Customer code

Every software developer, using PENTACON Developer Kit, has the unique *customer code*, which is stored in the HASP key memory.

Customer passwords

The *customer passwords* are four integers assigned to each software developer. The passwords are based on your unique customer code and ensure that only you can access your HASP keys.

User Data Area

Every HASP key, supplied by PENTACON Corporation for working with Developer Kit, contains the *user data area*. Software developers can use this memory at their discretion. Current area size is 16 bytes.

1. The developer registers in the PENTACON Corporation technical support service and obtains the customer's passwords. To provide the protection safety it is strongly recommended to store the passwords secretly.
2. PENTACON Corporation supplies the developer with the HASP keys, containing his unique customer code.
3. Developer specifies the customer passwords at addressing from his program to the Video Capture Library (RVCL) and DVPack™ Library (RDVP). The library collates the customer code and customer passwords. The access to the library functions is possible only in case of passwords correspondence to the code.

Customer code and customer passwords ensure that all HASP keys received from PENTACON Corporation by the developer will use nobody but him. This will increase the protection safety of the programs based on the PENTACON Developer Kit.

Make sure to keep your passwords in a safe place, as they are used to access the RVCL and RDVP libraries, protect your application and HASP user data area.

For the demonstrational objectives PENTACON Corporation provides the HASP keys, containing Demo customer code. Passwords for this key are used by default in all PENTACON Developer Kit samples.

Demo passwords (hex):

```
9592CB4F
F3B62AC0
FCB8802F
9CE9CEF9
```

5 PENTACON Developer Kit Library

The PENTACON Developer Kit Library (RDevKit) is intended for programming the contents of the key's memory. This library is intended only for developers and is not distributed in conjunction with the software products, built on the base of Developer Kit. Using this library gives you the opportunity to enter to the HASP key memory any data, which can be read by your program with the help of RVCL or RDVP libraries in future.

5.1 RDevKit Design Guide

RDevKit library contains the following functions:

RdkitReadKeyData
Reads data from the user data area of the HASP key memory.

RdkitWriteKeyData
Writes data into the user data area of the HASP key memory.

The **KeyEdit** program, being a part of PENTACON Developer Kit samples, illustrates the HASP key programming process. **KeyEdit** enables reading and modification of the data from the user data area of the HASP key. Program uses the RDevKit library.

5.2 RDevKit Reference

Functions

Call Type
All RDevKit function has **__stdcall** call type.

Parameters
For all functions of RDevKit library, where address of SRdkitXXX structure is used as a parameter, it is necessary to define size of structure in bytes in **dwSize** member before function call.

Return Values
All RDevKit function returns BOOL. If the function succeeds, the return value is nonzero. If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RdkitReadKeyData

The **RdkitReadKeyData** function reads data from the HASP key user data area.

```
BOOL RvclReadKeyData(const SRdkitKeyData* pKeyData)
```

Parameters

pKeyData

[in] Pointer to **SRdkitKeyData** structure where reading parameters are set. It is necessary to define size of structure in bytes in **dwSize** member before function calling.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RdkitWriteKeyData

The **RdkitWriteKeyData** function writes data into the HASP key user data area.

```
BOOL RvclWriteKeyData(const SRdkitKeyData* pKeyData)
```

Parameters

pKeyData

[in] Pointer to **SRdkitKeyData** structure where writing parameters are set. It is necessary to define size of structure in bytes in **dwSize** member before function calling.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Structures

SRdkitKeyData

The **SRdkitKeyData** structure contains user data area parameters information.

```
typedef struct _SRdkitKeyData
{
	DWORD dwSize;
	DWORD pdwCustomerPassword[4];
	DWORD dwDataOffset;
	DWORD dwDataSize;
	BYTE* pDataPtr;
} SRdkitKeyData;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure using.

pdwCustomerPasswords

Customer passwords.

dwDataOffset

Offset from the beginning of the user data area (in bytes).

dwDataSize

Specifies the number of bytes to write to the user data area or the number of bytes to be read from the user data area.

pDataPtr

Pointer to the buffer containing the data to be written to the user data area or to the buffer that receives the data read from the user data area.

5.3 RDevKit Examples

All PENTACON Developer Kit samples use *demo passwords* for initializing the RDevKit library. Every developer can modify samples, by entering the passwords, obtained from PENTACON CORPORATION technical support service for working with HASP keys.

KeyEdit

Illustrates reading and modification of the data from the user data area of the HASP key.

6 Video Capture Library

Video Capture Library (RVCL) is intended for video signal digitizing when using video/audio capture cards of **Tiny**, **Power** or **Titan** families.

6.1 RVCL Design Guide

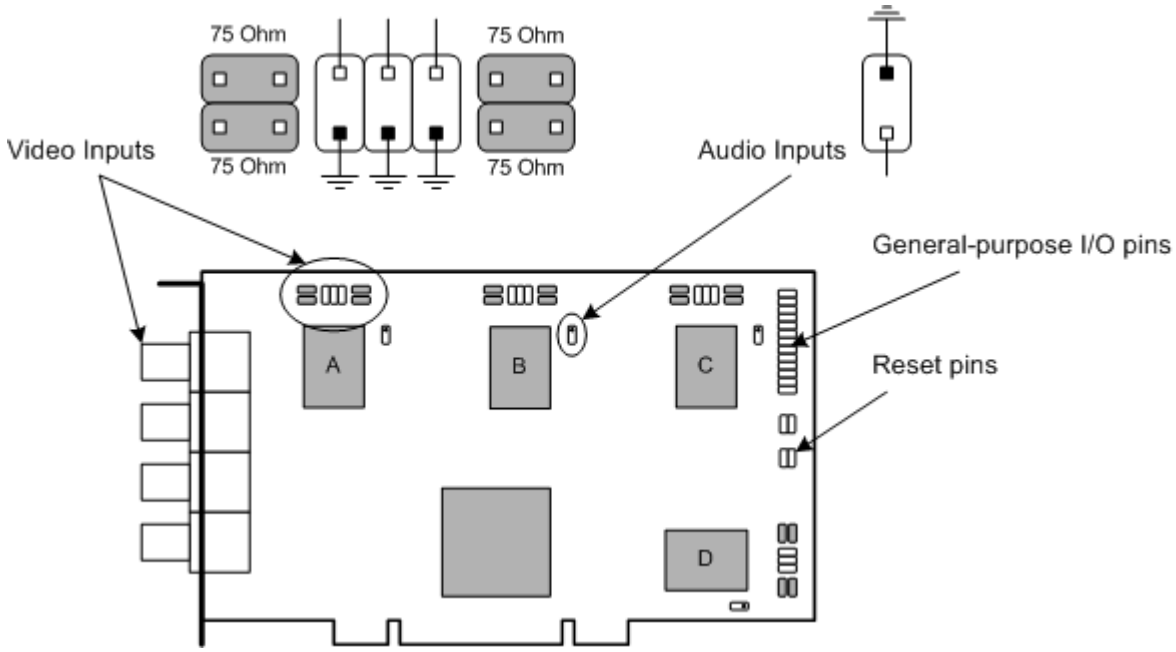
6.1.1 Capture Cards

- **TinyVN4/Pro/Pro2/Pro3** capture card contains 1 Conexant Bt848/849, or Conexant Fusion Bt878/879/878A or Cx2388x device (depending on modification).
- **PowerVN4/Pro/Pro2/Pro3/Pro4** capture card contains 4 independent Bt8xx (or Cx2388x depending on modification) devices (called A, B, C, D).
- **TitanVN8/Pro** capture card contains 8 independent Bt8xx (or Cx2388x depending on modification) devices (called A, B, C, D, E, F, G, H).

Each **Bt8xx** device has 4 video inputs and 1 audio input (878/879 only). Each **Cx2388** device has 4 video inputs and 2 audio inputs.
If only one video input is used (*real time mode*) maximum frame rate is equal to 25 frames per second for PAL (SECAM) standard and 30 frames per second for NTSC standard. In the case when more than one or all video signal inputs are used (*multiplexing mode*) maximum frame rate is 12 frames per second for PAL (SECAM) standard and 15 frames per second for NTSC standard. For example, if four video signal inputs are used, than frame rate is 12 / 4 = 3 frame per second for each input (PAL).

Power VN4
First video input of every **Bt8xx** device is connected to external connector. All other inputs (2, 3 and 4) of every device are located on additional part of card PCB. Each Bt8xx device can be used in multiplexing or non-multiplexing mode independently on each other. So maximum frame rate for **PowerVN4** video capture card is equal to 100 frames per second for PAL (SECAM) standard and 120 frames per second for NTSC standard.

Figure 6-1 PowerVN4 Card



As it was mentioned above **Bt8xx** capture devices are able to operate independently, some of them can be used in multiplexing mode and some in Real Time Mode (RTM). This feature provides flexibility of CCTV system based upon **Pentacon** video capture cards.

Besides the **Bt8xx** devices, **PowerVN4** card contains 22 general-purpose I/O pins. Every pin can be used as the input or the output independently of other pins. The scheme of contacts and voltage nominal are listed in the ***Appendix A: Videoblasters Pin Descriptions***. The D5 and D6 contacts can be used simultaneously as the outputs for the connection of light-emitting diodes. For that, there are contacts with the 50-Ohm resistors.

Also the Watchdog Timer is placed on the **PowerVN4** card. It allows controlling the computer operability on hardware level. The timer outputs are placed on the Reset pins. It is necessary to connect one of outputs to the motherboard Reset pin. It is possible to connect the Reset contacts to another output, placed on the computer case. These scheme will enable the Reset signal controlling both with the help of the **PowerVN4** card and manually.

PowerVN4 Pro, PowerVN4 Pro2, PowerVN4 Pro3 and **PowerVN4 Pro4** cards differ from each other by chips modification **Bt8xx** or **Cx2388x** chips placed on a card and their number and other features, for more information see ***Appendix C: Video Capture Cards Types*** .

TitanVN8/Pro
TitanVN8 is similar to **PowerVN4** card, but it has 8 independent **Bt8xx** devices and implemented as PCI64 (66 MHz) board. First video inputs of **Bt8xx** devices A, B, C and D are connected to external connector. **TitanVN8** card contains 44 general-purpose I/O pins. The D5 and D6 contacts can be used simultaneously as the outputs for the connection of light-emitting diodes. For that, there are contacts with the 50-Ohm resistors.

TitanVN8 Pro has 8 independent **Cx2388x** chips and implemented as PCI Express x1/x4/x8/x16 board. For more information about card technical characteristics see ***Appendix C: Video Capture Cards Types.***

TinyVN4
TinyVN4 has 1 **Bt8xx** or **Cx2388x** device and implemented as PCI 32(33 MHz) board. First video input of **Bt8xx** device is connected to external connector. Depending on modification (see ***Appendix C: Video Capture Cards Types***) **TinyVN4** card can contain 44 general-purpose I/O pins, D5 and D6 contacts can be used simultaneously as the outputs for the connection of light-emitting diodes.

6.1.2 Library Initialization

To initialize RVCL library call **RvclInitializeEx** function. If the function returns FALSE value further operation will be impossible. To get more information call **GetLastError.** The **RvclInitialize** function is obsolete.

To get a number of video capture device available call **RvclGetDevicesCount** function. Device index send to all library functions has to be in range 0…[Number of devices – 1].

To get library current version information call **RvclGetVersion** function. This function returns version and built numbers.

6.1.3 Video Multiplexer

Every Bt8xx type capture device installed on **PowerVN4** video capture card has own built-in video multiplexer for 4 video inputs. To switch multiplexer call **RvclSetVideoSource** function.

Bt8xx type capture device needs time interval to tune for video signal of input selected:
- Minimum 2 fields to define frame synchronization of new video signal.
- Minimum 8 fields to define polarity of new video signal.

After switching it is recommended to set up 2 fields time delay prior to frame grabbing. Bt8xx type capture device can define signal polarity incorrectly (to mix odd and even fields). While fast multiplexer switching this can cause image flattering. To avoid this effect it is necessary to set-up a delay at least for 8 fields.

To get a number of multiplexer input selected call **RvclGetDeviceStatus.** Set RVCL_STATUS_VIDEOSOURCE flag in **dwMask** member of **SRvclDeviceStatus** structure to 1.

For the frames capture in the *multiplexing mode* it is recommended to use the **dwVideoSource** field of the **SRvclFrame** structure instead of the **RvclSetVideoSource** function (see chapter 6.1.6).

6.1.4 Video Signal Formats

Bt8xx chips can digitize video signal of following formats: PAL, NTSC and SECAM.

Table 6-1 Video Signal Formats

	PAL	NTSC	SECAM
Maximum size of image digitized	768 * 576	640 * 480	768 * 576
Minimum size of image digitized	48 * 36	40 * 30	48 * 36
Number of fields per second	50	60	50
Number of frames per second	25	30	25
Field duration (ms)	20	16	20
Frame duration (ms)	40	33	40

To set video signal format call **RvclSetVideoSource** function.

To get video signal format set for Bt8xx device call **RvclGetDeviceStatus** function. Set RVCL_STATUS_VIDEOFORMAT flag in **dwMask** member of **SRvclDeviceStatus** structure to 1.

6.1.5 Color Formats

RVCL library allows diverse formats of video image digitizing: RGB, Grayscale, and YUV.

Table 6-2 Color Formats

Format	Developer Kit Code	Bits per pixel	Image Memory (bytes)	Width Alignment (pixels)
RGB 32	RDN_CF_RGB32	32	B0, G0, R0, 0, B1, G1, R1, 0, …	4
RGB 24	RDN_CF_RGB24	24	B0, G0, R0, B1, G1, R1, …	4
RGB 16	RDN_CF_RGB16	16 (5:6:5)	{B0[7:3],G0[7:2],R0[7:3]}, {B1[7:3],G1[7:2],R1[7:3]}, …	4
RGB 15	RDN_CF_RGB15	16 (5:5:5)	{B0[7:3],G0[7:3],R0[7:3],0}, {B1[7:3],G1[7:3],R1[7:3],0}, …	4
Grayscale	RDN_CF_Y8	8	Y0, Y1, Y2, Y3, …	4
YUV 4:2:2 Packed	RDN_CF_YUV422_PACKED	16	Y0, Cb0, Y1, Cr0, Y2, Cb2, Y3, Cr2, …	16
YUV 4:1:1 Packed	RDN_CF_YUV411_PACKED	12	Cb0, Y0, Cr0, Y1, Cb4, Y2, Cr4, Y3, Y4, Y5, Y6, Y7, Cb8, Y8, Cr8, Y9, Cb12, Y10, Cr12, Y11, Y12, Y13, Y14, Y15, …	16
YUV 4:2:2 Planar	RDN_CF_YUV422_PLANAR	16	Y0, Y1, Y2, Y3… Cb0, Cb1, Cb2, Cb3… Cr0, Cr1, Cr2, Cr3…	16
YUV 4:1:1 Planar	RDN_CF_YUV411_PLANAR	12	Y0, Y1, Y2, Y3… Cb0, Cb1, Cb2, Cb3… Cr0, Cr1, Cr2, Cr3…	16

YUV to RGB Conversion:

R = 1.164(Y-16) + 1.596(Cr-128)

G = 1.164(Y-16) - 0.813(Cr-128) - 0.391(Cb-128)

B = 1.164(Y-16) + 2.018(Cb-128)

Y range = [16,235]

Cr/Cb range = [16,240]

RGB range = [0,255]

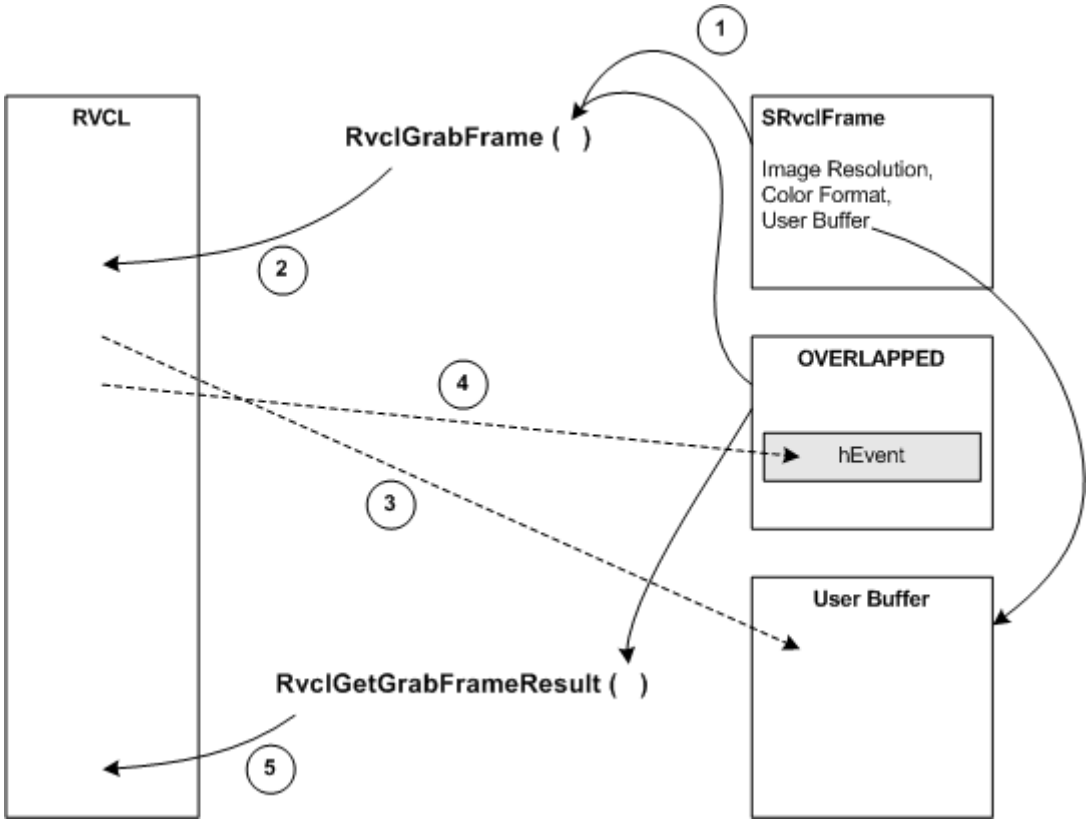
For the buffer size estimation (in bytes), necessary for the video image storage, use the following formula:
Buffer Size = (Image Width) * (Image Height) * (Number of bits per pixel) / 8

Note:
Required image width alignment (in pixels) is 4 for RGB and grayscale formats and 16 for YUV formats.

6.1.6 Video Capture

Video capture in RVCL library is performed is asynchronous mode.

Figure 6-2 RVCL Frame Capture



1. The application prepares **SRvclFrame** and **OVERLAPPED** structures and buffer for digitized image.

   The **SRvclFrame** structure must contain following parameters of capturing frame:
   - Image resolution in pixels.
   - Image color format (RGB, grayscale, YUV, etc. – see Table 6-2).
   - Number of video input (0…3) – when working in *multiplexing mode*.
   - Digitized image buffer address and size. Minimal size of buffer has to = (vertical image size in pixels) * (horizontal image size in pixels) * (number of bits per pixel, which defined by format) / 8.

   If vertical size of image is greater than number of lines in one field, the whole frame will be captured (odd and even fields). If vertical size of image is smaller or equal to number of lines in one field, only one field will be captured. In this case it is necessary to define what fields need to be captured: odd, even or nearest. To define it set flags in **dwFlags** member:
   - RVCL_GRAB_FIELD_ODD – capture odd field,
   - RVCL_GRAB_FIELD_EVEN – capture even field,
   - RVCL_GRAB_FIELD_NEAREST – capture nearest field.

   If RVCL_GRAB_NOSIGNAL flag is set in **dwFlags** member, the absents of video on device input will not be recognized as an error. Buffer will be filled by blue color (for color formats) or by gray color (for grayscale format). If RVCL_GRAB_NOSIGNAL is not set and there is no video signal on device input, **RvclGetGrabFrameResult** function will return FALSE and **GetLastError** function returns ERROR_NOT_READY.

   The **OVERLAPPED** structure must contain a handle to a manual-reset event object created by a call to the **CreateEvent** function. The system sets the state of the event object to nonsignaled when a call to the **RvclGrabFrame** function returns before the operation has been completed. The system

sets the state of the event object to signaled when the operation has been completed. For detailed information about **OVERLAPPED** structure refer to Win32 API documentation.

2. The application calls **RvclGrabFrame** function. If **RvclGrabFrame** function returns FALSE and **GetLastError** function returns ERROR_IO_PENDING (the most common case), the video capture operation is performed as an overlapped (asynchronous) operation.

   A thread can manage overlapped operations by either of two methods:
   - Use the **RvclGetGrabFrameResult** function to wait for the overlapped operation to be completed.
   - Specify a handle to the **OVERLAPPED** structure's manual-reset event object in one of the wait functions (**WaitForSingleObject**, **WaitForMultipleObjects**, etc.) and then call **RvclGetGrabFrameResult** after the wait function returns. The **RvclGetGrabFrameResult** function returns the results of the completed overlapped operation, and for functions in which such information is appropriate, it reports the actual number of bytes that were transferred.

   You can also use the **HasOverlappedIoCompleted** macro to poll for completion.

3. Digitized image is written into the buffer defined in **SRvclFrame** structure. The process of buffer filling is performed in busmaster DMA mode without usage of central processor(s).

4. When the pending video capture operation has been completed, the system sets the **hEvent** event object from **OVERLAPPED** structure to the signaled state.

5. The application calls **RvclGetGrabFrameResult**. If image capturing was successful **RvclGetGrabFrameResult** returns TRUE.

   If asynchronous operation of capturing is not finished yet **RvclGetGrabFrameResult** function returns FALSE and **GetLastError** function returns ERROR_IO_INCOMPLETE.

   If RVCL_GRAB_NOSIGNAL flag was not set for capturing process request and there is no signal on video input, **RvclGetGrabFrameResult** function returns FALSE and **GetLastError** function returns ERROR_NOT_READY. If RVCL_GRAB_NOSIGNAL flag was set for capturing process request, the absents of video on device input will not be recognized as an error. Buffer will be filled by blue color (for color formats) or by gray color (for grayscale format).

   If all pending capture operations for defined device were cancelled by **RvclCancelIo** function, **RvclGetGrabFrameResult** function returns FALSE and **GetLastError** function returns ERROR_OPERATION_ABORTED.

   For detailed information about overlapped input refer to "Synchronization and Overlapped Input and Output" chapter in Win32 API documentation.

6. To cancel all pending video capture operations for specified device the application calls **RvclCancelVideoCapture**. The **RvclCancelVideoCapture** function cancels all pending video capture operations that were issued by the calling thread for the specified device. The function does not cancel video capture operations issued for the device by other threads. All capture operations that are canceled will complete with the error ERROR_OPERATION_ABORTED. All completion notifications for the capture operations will occur normally. To cancel both video and audio capture operations, call **RvclCancelIo** function.

6.1.7 Audio Capture

Audio capture in RVCL library is performed in asynchronous mode, similarly to video capture.

1. The application prepares **SRvclAudioBlock** and **OVERLAPPED** structures and buffer for digitized audio data.

   The **SRvclAudioBlock** structure must contain following parameters of capturing block:
   - Number of bits per sample – 8, 16 or 32.
   - Number of samples.
   - Digitized data buffer address and size. Minimal size of buffer has to = (number of samples) * (bits per sample) / 8.

   Cx2388x can work in two modes: 1. Capturing Stereo channel. 2. Capturing two mono channels. (See p.65).

   The **OVERLAPPED** structure must contain a handle to a manual-reset event object created by a call to the **CreateEvent** function. The system sets the state of the event object to nonsignaled when a call to the **RvclGrabFrame** function returns before the operation has been completed. The system sets the state of the event object to signaled when the operation has been completed. For detailed information about **OVERLAPPED** structure refer to Win32 API documentation.

2. The application calls **RvclCaptureAudioBlock** function. If **RvclCaptureAudioBlock** function returns FALSE and **GetLastError** function returns ERROR_IO_PENDING (the most common case), the capture operation is performed as an overlapped (asynchronous) operation.

   A thread can manage overlapped operations by either of two methods:
   - Use the **RvclGetAudioCaptureResult** function to wait for the overlapped operation to be completed.
   - Specify a handle to the **OVERLAPPED** structure's manual-reset event object in one of the wait functions (**WaitForSingleObject**, **WaitForMultipleObjects**, etc.) and then call **RvclGetAudioCaptureResult** after the wait function returns. The **RvclGetAudioCaptureResult** function returns the results of the completed overlapped operation, and for functions in which such information is appropriate, it reports the actual number of bytes that were transferred.

   You can also use the **HasOverlappedIoCompleted** macro to poll for completion.

3. Digitized audio data is written into the buffer defined in **SRvclAudioBlock** structure. The process of buffer filling is performed in busmaster DMA mode without usage of central processor(s).

   Due to Bt8xx hardware features the audio capture is performed with fixed frequency RVCL_BT8XX_AUDIO_FREQUENCY, which is equal to 119318 Hz. Such frequency is too high for standard audio output devices and audio compression codecs. Therefore it is nessesary to reduce audio samples up to standard frequency (8000, 11025, 22050, 32000, 44100 or 48000 Hz) with **RvclScaleAudioBlock** function after block has been captured.

   Due to Cx2388x hardware features the audio capture is performed with fixed frequency RVCL_CX2388X_AUDIO_FREQUENCY, which is equal to 48000 Hz.

4. When the pending capture operation has been completed, the system sets the **hEvent** event object from **OVERLAPPED** structure to the signaled state.

5. The application calls **RvclGetAudioCaptureResult**. If image capturing was successful **RvclGetAudioCaptureResult** returns TRUE.

   If asynchronous operation of capturing is not finished yet **RvclGetAudioCaptureResult** function returns FALSE and **GetLastError** function returns ERROR_IO_INCOMPLETE.

   If all pending capture operations for defined device were cancelled by **RvclCancello** or **RvclCancelAudioCapture** function, **RvclGetAudioCaptureResult** function returns FALSE and **GetLastError** function returns ERROR_OPERATION_ABORTED.

For detailed information about overlapped input refer to "Synchronization and Overlapped Input and Output" chapter in Win32 API documentation.

6. To cancel all pending audio capture operations for specified device the application calls **RvclCancelAudioCapture**. The **RvclCancelAudioCapture** function cancels all pending audio capture operations that were issued by the calling thread for the specified device. The function does not cancel audio capture operations issued for the device by other threads. All capture operations that are canceled will complete with the error ERROR_OPERATION_ABORTED. All completion notifications for the capture operations will occur normally. To cancel both video and audio capture operations, call **RvclCancelIo** function.

7. To scale audio data frequency to standard value the application calls **RvclScaleAudioBlock** function.

To get and set audio capture parameters, such as input signal gain, call **RvclAudioControl** function.

6.1.8 General Purpose I/O (GPIO) Port

The Bt8xx provides 24-bit wide general-purpose I/O port. Each of the general-purpose I/O pins can be programmed individually. An internal ***output enable*** register can be programmed to enable the output buffers of the pins selected as outputs. The contents of the ***data*** register are put on the enabled GPIO output pins. In case the GPIO pins are used as general purpose input pins, the contents of the GPIO data register are ignored and the signals on the GPIO bus pins are read through a separate register.

To read and modify ***output enable*** and ***data*** registers call **RvcIGpioControl** function.

6.1.9 PowerVN4/Pro/Pro2/Pro3, Tiny VN4/Pro/Pro2/Pro3, TitanVN8/Pro/ and Watchdog Timer

The **Power**, **Titan** and **Tiny** cards provide one watchdog timer connected to the **Bt8xx/Cx2388x** device **D** (#4). The watchdog timer circuit may be used to monitor the activity of the microprocessor in order to check that it is not stalled in an indefinite loop or in an unserved IRQ.

To activate or deactivate watchdog timer call **RvclActivateWatchdog** function. The **nDevice** parameter should point to the Bt8xx/Cx2388x device **D** on **Power/Tiny/Titan** card. For example, set **nDevice** to 3 if the only one **Power/Titan** card is installed (assuming the numeration to count from zero).

To perform hardware computer's work monitoring:
1. Connect one of "Reset" pins on the **Power/Titan/Tiny** board with "Reset" pin on the motherboard (see *Appendix A*).
2. Activate **Power/Titan/Tiny** watchdog timer.

6.1.10 HASP Key User Data Area

Every HASP key, supplied by PENTACON CORPORATION for working with Developer Kit, contains the *user data area*. Software developers can use this memory at their discretion. See chapter 4 for details.

To get user data area size call **RvcIInitializeEx** function. To read HASP key user data call **RvcIReadKeyData** function.

6.2 RVCL Reference

Functions

Call Type
All RVCL function has **__stdcall** call type.

Parameters
For all functions of RVCL library, where address of SRvclXXX structure is used as a parameter, it is necessary to define size of structure in bytes in **dwSize** member before function call.

Return Values
All RVCL function returns BOOL. If the function succeeds, the return value is nonzero. If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RvclInitializeEx

The **RvclInitializeEx** function provides start initialization of RVCL library.

```
BOOL RvclInitialize(SRvclInitialize* pInit)
```

Parameters

pInit

[in,out] Pointer to **SRvclInitialize** structure. It is necessary to define size of structure in bytes in **dwSize** member before function call.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

RvclInitialize function must be called once prior to call of any other function of RVCL library.

RvclInitialize

The **RvclInitialize** function is obsolete and always returns error code.

```
BOOL RvclInitialize()
```

RvclGetVersion

The **RvclGetVersion** function returns RVCL library version description.

```
BOOL RvclGetVersion(SRvclVersionInfo* pVersionInfo)
```

Parameters

pVersionInfo

[out] Pointer to **SRvclVersionInfo** structure, which must be filled by library version information. It is necessary to define size of structure in bytes in **dwSize** member before function call.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RvclGetDevicesCount

The **RvclGetDevicesCount** function returns a number of video capture devices available.

```
BOOL RvclGetDevicesCount(int* pnDevicesCount)
```

Parameters

pnDevicesCount

[out] Pointer to variable, that the function fills with number of video captures devices available.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RvclGetDeviceInfo

The **RvclGetDeviceInfo** function returns video capture devices static information.

```
BOOL RvclGetDeviceInfo(
            int nDevice,
            SRvclDeviceInfo* pDeviceInfo)
```

Parameters

nDevice

[in] Zero-based device index.

pDeviceInfo

[out] Pointer to **SRvclDeviceInfo** structure, which has to be filled by device information. It is necessary to define size of structure in bytes in **dwSize** member before function call.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RvclSetVideoSource

The **RvclSetVideoSource** function performs built-in multiplexer switching to specified input.

```
BOOL RvclSetVideoSource(
            int nDevice,
            DWORD dwVideoSource)
```

Parameters

nDevice

[in] Zero-based device index.

dwVideoSource

[in] Zero-based video input number. Value can be in range from 0 to 3.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

Bt8xx type capture device needs time interval to tune for video signal of input selected:

- Minimum 2 fields to define frame synchronization of new video signal.
- Minimum 8 fields to define polarity of new video signal.

It is recommended to set up 2 fields time delay prior to frame grabbing. Bt8xx type capture device can define signal polarity incorrectly (to mix odd and even fields). While fast multiplexer switching this can cause image flattering. To avoid this effect it is necessary to set-up a delay at least for 8 fields.

RvclSetVideoFormat

The **RvclSetVideoFormat** function sets video signal format.

```
BOOL RvclSetVideoFormat(
            int nDevice,
            DWORD dwVideoFormat)
```

Parameters

nDevice

[in] Zero-based device index.

dwVideoFormat

[in] Format code:

Table 6-3 RVCL Video Signal Format Codes

Code	Video Signal Format
RVCL_VF_NTSCM	NTSC (M)
RVCL_VF_NTSCJ	NTSC (Japan)
RVCL_VF_PALBDGHI	PAL (B,D,G,H,I)
RVCL_VF_PALM	PAL (M)
RVCL_VF_PALN	PAL (N)
RVCL_VF_SECAM	SECAM

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RvclGetDeviceStatus

The **RvclGetDeviceStatus** function returns current device state.

```
BOOL RvclGetDeviceStatus(
            int nDevice,
            SRvclDeviceStatus* pDeviceStatus)
```

Parameters

nDevice

[in] Zero-based device index.

pDeviceStatus

[in,out] Pointer to **SRvclDeviceStatus** structure, which has to be filled by current device state information. Before function call it is necessary to define size of structure in bytes in **dwSize** member and flags mask defining what members of structure will be filled in **dwMask** member.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RvclSetVideoAdjustments

The **RvclSetVideoAdjustments** function sets digitizing parameters.

```
BOOL RvclSetVideoAdjustments(
            int nDevice,
            const SRvclVideoAdjustments* pVideoAdjustments)
```

Parameters

nDevice

[in] Zero-based device index.

pVideoAdjustments

[in] Pointer to **SRvclVideoAdjustments** structure, where digitizing parameters are set. Before function call it is necessary to define size of structure in bytes in **dwSize** member and flags mask defining what members of structure are valid in **dwMask** member.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RvclGetVideoAdjustments

The **RvclGetVideoAdjustments** function reads current digitizing parameters

```
BOOL RvclGetVideoAdjustments(
            int nDevice,
            SRvclVideoAdjustments* pVideoAdjustments)
```

Parameters

nDevice

[in] Zero-based device index.

pVideoAdjustments

[in,out] Pointer to **SRvclVideoAdjustments** structure, which has to be filled by current digitizing parameter information. Before function call it is necessary to define size of structure in bytes in **dwSize** member and flags mask defining what members of structure must be filled in **dwMask** member.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RvclGrabFrame

The **RvclGrabFrame** function sets a request for frame capture asynchronous operation.

```
BOOL RvclGrabFrame(
            int nDevice,
            const SRvclFrame* pFrame,
            OVERLAPPED* pOverlapped)
```

Parameters

nDevice

[in] Zero-based device index.

pFrame

[in] Pointer to **SRvclFrame** structure where capturing parameters are set. Before function call it is necessary to define size of structure in bytes in **dwSize** member.

pOverlapped

[in,out] Pointer to **OVERLAPPED** structure, which must contain a handle to a manual-reset event object created by a call to the **CreateEvent** function. For detailed information about **OVERLAPPED** structure refer to Win32 API documentation.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

If **RvclGrabFrame** returns FALSE and **GetLastError** function returns ERROR_IO_PENDING, the capture operation is performed as an overlapped (asynchronous) operation.

RvclGetGrabFrameResult

The **RvclGetGrabFrameResult** function returns the results of an overlapped capture operation on the specified device.

```
BOOL RvclGetGrabFrameResult(
            int nDevice,
            OVERLAPPED* pOverlapped,
            DWORD* pdwBytesWritten,
            BOOL bWait)
```

Parameters

nDevice

[in] Zero-based device index.

pOverlapped

[in] Pointer to an **OVERLAPPED** structure that was specified when the overlapped capture operation was started.

pdwBytesWritten

[out] Pointer to a variable that receives the number of bytes that were actually transferred by a capture operation.

bWait

[in] Specifies whether the function should wait for the pending overlapped operation to be completed. If TRUE, the function does not return until the operation has been completed. If FALSE and the operation is still pending, the function returns FALSE and the **GetLastError** function returns ERROR_IO_INCOMPLETE.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

The results reported by the **RvclGetGrabFrameResult** function are those of the specified device's last overlapped operation to which the specified **OVERLAPPED** structure was provided, and for which the operation's results were pending. A pending operation is indicated when the **RvclGrabFrame** function that started the operation returns FALSE, and the **GetLastError** function returns ERROR_IO_PENDING. When a capture operation is pending, the **RvclGrabFrame** function resets the **hEvent** member of the **OVERLAPPED** structure to the nonsignaled state. Then when the pending operation has been completed, the system sets the event object to the signaled state.

Specify a manual-reset event object in the **OVERLAPPED** structure. If an auto-reset event object is used, the event handle must not be specified in any other wait operation in the interval between starting the overlapped operation and the call to **RvclGetGrabFrameResult**. For example, the event object is sometimes specified in one of the wait functions to wait for the operation's completion. When the wait function returns, the system sets an auto-reset event's state to nonsignaled, and a subsequent call to **RvclGetGrabFrameResult** with the *bWait* parameter set to TRUE causes the function to be blocked indefinitely.

If the *bWait* parameter is TRUE, **RvclGetGrabFrameResult** determines whether the pending operation has been completed by waiting for the event object to be in the signaled state.

If RVCL_GRAB_NOSIGNAL flag was not set and video signal is absent on device input **RvclGetGrabFrameResult** returns FALSE and **GetLastError** function returns ERROR_NOT_READY. If RVCL_GRAB_NOSIGNAL flag is set in **dwFlags** member, the absents of video on device input will not be recognized as an error. Buffer will be filled by blue color (for color formats) or by gray color (for Black&White formats).

If all pending capture operations were cancelled by **RvclCancelIo** function call, **RvclGetGrabFrameResult** returns FALSE and **GetLastError** function returns ERROR_OPERATION_ABORTED.

RvclCancelVideoCapture

The **RvclCancelVideoCapture** function cancels all pending video capture operations that were issued by the calling thread for the specified device. The function does not cancel video capture operations issued for the device by other threads or audio capture operations.

```
BOOL RvclCancelVideoCapture(
            int nDevice)
```

Parameters

nDevice

[in] Zero-based device index.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

The **RvclCancelVideoCapture** function cancels all pending video capture operations that were issued by the calling thread for the specified device. The function does not cancel video capture operations issued for the device by other threads or audio capture operations. All capture operations that are canceled will complete with the error ERROR_OPERATION_ABORTED. All completion notifications for the capture operations will occur normally.

RvclAudioControl

The **RvclAudioControl** function sets and(or) gets device audio capture parameters.

```
BOOL RvclAudioControl(
            int nDevice,
            const SRvclAudioControl* pAudioCtrl,
            OVERLAPPED* pOverlapped)
```

Parameters

nDevice

[in] Zero-based device index.

pAudioCtrl

[in,out] Pointer to **SRvclAudioControl** structure where audio capturing parameters are set. Before function call it is necessary to define size of structure in bytes in **dwSize** member.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RvclCaptureAudioBlock

The **RvclCaptureAudioBlock** function sets a request for audio block capture asynchronous operation.

```
BOOL RvclCaptureAudioBlock(
            int nDevice,
            const SRvclAudioBlock* pAudioBlock,
            OVERLAPPED* pOverlapped)
```

Parameters

nDevice

[in] Zero-based device index.

pFrame

[in] Pointer to **SRvclAudioBlock** structure where capturing parameters are set. Before function call it is necessary to define size of structure in bytes in **dwSize** member.

pOverlapped

[in,out] Pointer to **OVERLAPPED** structure, which must contain a handle to a manual-reset event object created by a call to the **CreateEvent** function. For detailed information about **OVERLAPPED** structure refer to Win32 API documentation.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

If **RvclCaptureAudioBlock** returns FALSE and **GetLastError** function returns ERROR_IO_PENDING, the capture operation is performed as an overlapped (asynchronous) operation.

RvclGetAudioCaptureResult

The **RvclGetAudioCaptureResult** function returns the results of an overlapped audio capture operation on the specified device. It has the same behaviour as **RvclGetGrabFrameResult** function.

```
BOOL RvclAudioCaptureResult(
            int nDevice,
            OVERLAPPED* pOverlapped,
            DWORD* pdwBytesWritten,
            BOOL bWait)
```

Parameters

nDevice

[in] Zero-based device index.

pOverlapped

[in] Pointer to an **OVERLAPPED** structure that was specified when the overlapped audio capture operation was started.

pdwBytesWritten

[out] Pointer to a variable that receives the number of bytes that were actually transferred by a capture operation.

bWait

[in] Specifies whether the function should wait for the pending overlapped operation to be completed. If TRUE, the function does not return until the operation has been completed. If FALSE and the operation is still pending, the function returns FALSE and the **GetLastError** function returns ERROR_IO_INCOMPLETE.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

See **RvclGetGrabFrameResult** for more information.

RvclCancelAudioCapture

The **RvclCancelAudioCapture** function cancels all pending audio capture operations that were issued by the calling thread for the specified device. The function does not cancel audio capture operations issued for the device by other threads or video capture operations.

```
BOOL RvclCancelAudioCapture(
            int nDevice)
```

Parameters

nDevice

[in] Zero-based device index.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

The **RvclCancelAudioCapture** function cancels all pending audio capture operations that were issued by the calling thread for the specified device. The function does not cancel audio capture operations issued for the device by other threads or video capture operations. All capture operations that are canceled will complete with the error ERROR_OPERATION_ABORTED. All completion notifications for the capture operations will occur normally.

RvclScaleAudioBlock

The **RvclScaleAudioBlock** function reduces audio samples and lowers audio frequency to desired value.

```
BOOL RvclCancelAudioCapture
            void* pUnscaledDataPtr,
            void* pScaledDataPtr,
            DWORD dwUnscaledSamples,
            DWORD dwScaledSamples,
            DWORD dwBitsPerSample)
```

Parameters

pUnscaledDataPtr

[in] Pointer to a buffer containing source audio block.

pScaledDataPtr

[out] Pointer to destination buffer where scaled (reduced) audio data will be placed. Can be equal to **pUnscaledDataPtr**.

dwUnscaledSamples

[in] Initial number of audio samples in the source buffer. Must be divisible by 4.

dwScaledSamples

[in] Number of scaled samples in destination buffer. Must be less than **dwUnscaledSamples**. Must be divisible by 4.

dwBitsPerSample

[in] Number of bits per sample (8 or 16).

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

Use the following formula to calculate scaled (destination) samples value:

ScaledSamples =
((UnscaledSamples * ScaledFrequency) / UnscaledFrequency) & 0xFFFFFFFC

For example, if buffer size is 40 Kb, one sample is 16 bit, desired audio frequency is 44100 Hz, than:

UnscaledSamples = 40960 / 2 = 20480
ScaledSamples = ((20480 * 44100) / RVCL_BASE_AUDIO_FREQUENCY) & 0xFFFFFFFC = 7568

RvclCancelIo

The **RvclCancelIo** function cancels all pending capture operations (video and audio) that were issued by the calling thread for the specified device. The function does not cancel capture operations issued for the device by other threads.

```
BOOL RvclCancelIo(
            int nDevice)
```

Parameters

nDevice

[in] Zero-based device index.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

The **RvclCancelIo** function cancels all pending capture operations that were issued by the calling thread for the specified device. The function does not cancel capture operations issued for the device by other threads. All capture operations that are canceled will complete with the error ERROR_OPERATION_ABORTED. All completion notifications for the capture operations will occur normally.

RvclGpioControl

The **RvclGpioControl** function manipulates with Bt8xx GPIO port.

```
BOOL RvclGpioControl(
            int nDevice,
            SRvclGpioControl* pGpioControl)
```

Parameters

nDevice

[in] Zero-based device index.

pGpioControl

[in,out] Pointer to **SRvclGpioControl** structure where GPIO parameters are set. It is necessary to define size of structure in bytes in **dwSize** member before function calling.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RvclActivateWatchdog

The **RvclActivateWatchdog** function activates or deactivates **Power/Titan/Tiny** watchdog timer.

```
BOOL RvclActivateWatchdog(
            int nDevice,
            BOOL fActivate)
```

Parameters

nDevice

[in] Zero-based device index. Must point to the Bt8xx device **D** on **Power/Titan/Tiny** card.

fActivate

[in] TRUE to activate watchdog timer, FALSE to deactivate it.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RvclReadKeyData

The **RvclReadKeyData** function reads data from the HASP key user data area.

```
BOOL RvclReadKeyData(const SRvclKeyData* pKeyData)
```

Parameters

pKeyData

[in] Pointer to **SRvclKeyData** structure where reading parameters are set. It is necessary to define size of structure in bytes in **dwSize** member before function calling.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RvclDeinterlaceFilter

The **RvclDeinterlaceFilter** function returns a number of video capture devices available. The filter was designed to remove interlace from video (effect, that appear at joining of two half-frames into one). Attention. For working with RvclDeinterlaceFilter it is necessary to obtain the corresponding HASP key license.

```
BOOL RvclDeinterlaceFilter(VOID* pDataPtr, SIZE sizeRes, DWORD dwColorFormat)
```

Parameters

pDataPtr

[in,out] Pointer to buffer, where image was placed.

sizeRes

[in] Size of capturing image. To get maximum and minimum capture image resolution and horizontal and vertical alignment call **RvclGetDeviceStatus** function.

dwColorFormat

[in] Digitized image color format code.

Code	Format	Bits per pixel
RVCL_CF_RGB32	RGB 32	32
RVCL_CF_RGB24	RGB 24	24
RVCL_CF_Y8 RVCL_CF_GRAYSCALE	Grayscale (Y8, Black & White)	8
RVCL_CF_YUV422_PACKED	YUV 4:2:2 Packed	16
RVCL_CF_YUV411_PACKED	YUV 4:1:1 Packed	12
RVCL_CF_YUV422_PLANAR	YUV 4:2:2 Planar	16
RVCL_CF_YUV411_PLANAR	YUV 4:1:1 Planar	12

Attention! RGB 16, RGB 15 formats not supported by this function.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Notes

About Deinterlace. You possibly noticed that in many movies - especially shot with home camcorders - in episodes where fast movement is recorded, the image looks blurred and stripped. This is because many cameras divide each frame into even and odd lines: first the even lines are recorded, then the odd ones. Hence, each frame you observe on your monitor is actually comprised of two half-frames - "even" and "odd" ones. Half-frames are connected with each other line by line - that is, the first line of the first half-frame - the first line of the second half-frame; the second line of the first half-frame - the second line of the second half-frame; etc. As a result, if the frame contains fast enough movement, then on the second half-frame the moving object has already moved to slightly different position compared to the first half-frame, and at the connection points the "stripes" effect occurs.
Deinterlace is a process which allows to avoid the undesirable effects while connecting the two half-frames.

Structures

SRvclInitialize

The **SRvclInitialize** structure contains RVCL initialization information.

```
typedef struct _SRvclInitialize
{
      DWORD dwSize;
      DWORD pdwCustomerPassword[4];
      DWORD dwUserDataSize;
} SRvclInitialize;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure is used.

pdwCustomerPasswords

Customer passwords.

dwUserDataSize

HASP key user data area size (in bytes). This member is filled in during **RvclInitializeEx** call.

SRvclVersionInfo

The **SRvclVersionInfo** structure contains RVCL current version information.

```
typedef struct _SRvclVersionInfo
{
      DWORD dwSize;
      WORD  wMajor;
      WORD  wMinor;
      WORD  wBuild;
} SRvclVersionInfo;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure is used.

wMajor

Major library version number.

wMinor

Minor library version number.

wBuild

Build number.

SRvclDeviceInfo

The **SRvclDeviceInfo** structure contains video capture device static information.

```
typedef struct _SRvclDeviceInfo
{
      DWORD dwSize;
      WORD  wPciDeviceID;
      DWORD dwPciBusNumber;
      DWORD dwPciSlotNumber;
      DWORD dwDeviceType;
      CHAR  szDeviceDesc[16];
} SRvclDeviceInfo;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure is used.

wPciDeviceID

PCI device ID. Defined by device manufacturer.

Table 6-4 Device IDs

Value	Device
848	Bt848
849	Bt849
878	Bt878
879	Bt879
34816	Cx2388x

dwPciBusNumber

Number of PCI-bus where device is installed.

dwPciSlotNumber

Number of PCI-slot where device is installed.

dwDeviceType

Returns the type of device, described by this enum:

```
enum EDeviceType
{
      E_DEVICE_BT8XX   = 0,
      E_DEVICE_CX2388X = 1,
};
```

szDeviceDesc

Text representation of device type value. Can be “Bt8xx” or “Cx2388x”.

SRvclDeviceStatus

The **SRvclDeviceStatus** structure contains video capture device current state information.

```
typedef struct _SRvclDeviceStatus
{
      DWORD dwSize;
      DWORD dwMask;
      BOOL  fVideoPresent;
      DWORD dwVideoSource;
      DWORD dwVideoFormat;
      SIZE  sizeMaxResolution;
      SIZE  sizeMinResolution;
      SIZE  sizeResolutionAlignment;
} SRvclDeviceStatus;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure is used.

dwMask

Set of flags that specify which members of this structure are being requested.

RVCL_STATUS_VIDEOPRESENT	The **fVideoPresent** member must be filled in.
RVCL_STATUS_VIDEOSOURCE	The **dwVideoSource** member must be filled in.
RVCL_STATUS_VIDEOFORMAT	The **dwVideoFormat** member must be filled in.
RVCL_STATUS_RESOLUTION	The **sizeMaxResolution**, **sizeMinResolution**, **sizeResolutionAlignment** members must be filled in.

fVideoPresent

Flag shows the presents of signal on Bt8xx device video input. TRUE if video signal is present.

dwVideoSource

Number of Bt8xx device video signal input selected by in-built multiplexer. Value can be in range from 0 to 3.

dwVideoFormat

Current video signal format. For list of formats available refer to **RvclSetVideoFormat** function description.

sizeMaxResolution

Maximum size of the image, which can be captured using video signal format selected.

sizeMinResolution

Minimum size of the image, which can be captured using video signal format selected.

sizeResolutionAlignment

Capturing image size alignment is required for video signal format selected.

Remarks

For every video signal format maximum and minimum capture image resolution is set. Besides, it is set horizontal and vertical alignment – the grade image size modification.

Table 6-5 Image Resolutions

Format	Maximum { horizontal, vertical }	Minimum { horizontal, vertical }	Alignment { horizontal, vertical }
NTSC	{ 640, 480 }	{ 40, 30 }	{ 4, 1 }
PAL	{ 768, 576 }	{ 48, 36 }	{ 4, 1 }
SECAM	{ 768, 576 }	{ 48, 36 }	{ 4, 1 }

SRvclVideoAdjustments

The **SRvclVideoAdjustments** structure contains video signal digitizing parameters.

```
typedef struct _SRvclVideoAdjustments
{
	DWORD dwSize;
	DWORD dwMask;
	DWORD dwBrightness;
	DWORD dwContrast;
	DWORD dwSaturation;
	DWORD dwHue;
	DWORD dwCombRange;
} SRvclVideoAdjustments;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure is used.

dwMask

Set of flags that specify which members of this structure are valid or which members are being requested.

RVCL_VADJUST_BRIGHTNESS	The **dwBrightness** member is valid or must be filled in.
RVCL_VADJUST_CONTRAST	The **dwContrast** member is valid or must be filled in.
RVCL_VADJUST_SATURATION	The **dwSaturation** member is valid or must be filled in.
RVCL_VADJUST_HUE	The **dwHue** member is valid or must be filled in.
RVCL_VADJUST_COMBRANGE	The **dwCombRange** member is valid or must be filled in.

dwBrightness

Brightness. Value can be in the rage from 0 to RVCL_MAX_BRIGHTNESS. Default value is RVCL_DEFAULT_BRIGHTNESS.

dwContrast

Contrast. Value can be in the rage from 0 to RVCL_MAX_CONTRAST. Default value is RVCL_DEFAULT_CONTRAST.

dwSaturation

Saturation. Value can be in the rage from 0 to RVCL_MAX_SATURATION. Default value is RVCL_DEFAULT_SATURATION.

dwHue

Hue. Value can be in the rage from 0 to RVCL_MAX_HUE. Default value is RVCL_DEFAULT_HUE.

dwCombRange

COMP filter. Value can be in the rage from 0 to RVCL_MAX_COMBRANGE. Default value is RVCL_DEFAULT_COMBRANGE.
COMB filter is intended for elimination of artifacts, which appear on video as a result of analog-to-digital data signal conversion. An advantage of COMB filter is that it represents a hardware filtering by itself. By the moment of digitization a chip has the most information about video signal, that's why the hardware filtering during digitization leads to the better results than application of filters to already digitized signal.

SRvclFrame

The **SRvclFrame** structure contains video capture parameters information.

```
typedef struct _SRvclFrame
{
	DWORD dwSize;
	DWORD dwFlags;
	SIZE  sizeResolution;
	DWORD dwColorFormat;
	void* pBufferPtr;
	DWORD dwBufferSize;
	DWORD dwVideoSource;
} SRvclFrame;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure is used.

dwFlags

Flags:

RVCL_GRAB_FIELD_ODD	If vertical size of image **sizeResolution.cy** is smaller or equal to number of lines in one field for current signal format, odd field will be captured.
RVCL_GRAB_FIELD_EVEN	If vertical size of image **sizeResolution.cy** is smaller or equal to number of lines in one field for current signal format, even field will be captured.
RVCL_GRAB_FIELD_NEAREST	If vertical size of image **sizeResolution.cy** is smaller or equal to number of lines in one field for current signal format, nearest field will be captured.
RVCL_GRAB_NOSIGNAL	Produce frame capturing independently on signal on Bt8xx device video input presents. If this flag is set to 1, buffer will be filled by blue color (for color formats) or by gray color (for grayscale format) and capturing operation will be performed successfully. If this flag is not set capturing operation will be terminated with ERROR_NOT_READY error if video signal is not present.
RVCL_GRAB_BOTTOM_UP	Produce bottom-up image capturing. Bottom-up means that the first line in the frame is the last line to be stored in the buffer.
RVCL_GRAB_AFTER_SWITCH	Produces image capturing from **dwVideoSource** video input. Perform built-in multiplexer switching if necessary. If this flag is not set **dwVideoSource** member is ignored.
RVCL_GRAB_SVIDEO	Captures S-Video (for Cx2388x only).

sizeResolution

Size of capturing image. To get maximum and minimum capture image resolution and horizontal and vertical alignment call **RvclGetDeviceStatus** function.

dwColorFormat

Digitized image color format code.

Code	Format	Bits per pixel
RVCL_CF_RGB32	RGB 32	32
RVCL_CF_RGB24	RGB 24	24
RVCL_CF_RGB16	RGB 16	16 in format 5:6:5
RVCL_CF_RGB15	RGB 15	16 in format 5:5:5
RVCL_CF_Y8 RVCL_CF_GRAYSCALE	Grayscale (Y8, Black & White)	8
RVCL_CF_YUV422_PACKED	YUV 4:2:2 Packed	16
RVCL_CF_YUV411_PACKED	YUV 4:1:1 Packed	12
RVCL_CF_YUV422_PLANAR	YUV 4:2:2 Planar	16
RVCL_CF_YUV411_PLANAR	YUV 4:1:1 Planar	12

See

Color Formats for more information.

pBufferPtr

Pointer to buffer where image will be placed.

dwBufferSize

Size of buffer where image will be placed. Size of buffer has to be equal at least to (sizeResolution.cx) * (sizeResolution.cy) * (Number of bits per pixel) / 8.

dwVideoSource

Zero-based video input number. Value can be in range from 0 to 3. Set RVCL_GRAB_AFTER_SWITCH flag to take effect.

Remarks

If required image vertical size **sizeResolution.cy** is smaller or equal to number of lines of one field for current signal format, capturing of only of one field will be performed. For PAL and SECAM formats field contains 288 lines and for NTSC format field contains 240 lines. RVCL_GRAB_FIELD_ODD, RVCL_GRAB_FIELD_EVEN and RVCL_GRAB_FIELD_NEAREST flags allow to define what field is it necessary to capture in this case – odd, even or nearest one. This flags are mutually exclusive, only one of them can be set same time.

SRvclAudioControl

The **SRvclAudioControl** structure contains audio capture parameters information.

```
typedef struct _SRvclAudioControl
{
      DWORD dwSize;
      DWORD dwMask;
      DWORD dwFlags;
      DWORD dwGain;
} SRvclAudioControl;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure using.

dwMask

Set of flags that specify which members of this structure contain data to be set or which members are being requested. This member can have one or more of the following flags set:

RVCL_AUDIO_SET_FLAGS	The **dwFlags** member is valid and contain flags to set.
RVCL_AUDIO_SET_GAIN	The **dwGain** member is valid and gain value to set.
RVCL_AUDIO_GET_FLAGS	The **dwFlags** member must be filled in with flags.
RVCL_AUDIO_GET_GAIN	The **dwGain** member must be filled in with gain value.

dwFlags

Flags:

RVCL_AUDIO_GAINBOOST	Adds a +6 dB gain in Bt8xx audio pre-amp. The 6 dB boost is useful for very small input signals.

dwGain

Input signal gain code:

Code	Input Gain	dB	Nominal Input V_{rms}	V_{p-p}
0	0.500	–6.02	1.000	2.828
1	0.667	–3.52	0.750	2.121
2	0.833	–1.58	0.600	1.697
3	1.000	0.00	0.500	1.414
4	1.167	1.34	0.429	1.212
5	1.333	2.50	0.375	1.061
6	1.500	3.52	0.333	0.943
7	1.667	4.44	0.300	0.849
8	1.833	5.26	0.273	0.771
9	2.000	6.02	0.250	0.707
10	2.167	6.72	0.231	0.653
11	2.333	7.36	0.214	0.606
12	2.500	7.96	0.200	0.566
13	2.667	8.52	0.188	0.530
14	2.833	9.05	0.176	0.499
15	3.000	9.54	0.167	0.471

SRvclAudioBlock

The **SRvclAudioBlock** structure contains audio capture parameters information.

```
typedef struct _SRvclAudioBlock
{
	DWORD dwSize;
	DWORD dwFlags;
	DWORD dwBitsPerSample;
	DWORD dwSamples;
	void* pBufferPtr;
	DWORD dwBufferSize;
} SRvclAudioBlock;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure is used.

dwFlags

Flags.

RVCL_CX2388X_AUDIO_STEREO	Grab audio as stereo (left and right channels consecutively)
RVCL_CX2388X_AUDIO_RIGHT	Grab audio as two mono buffers. Shows, that it is necessary to grab right channel.
RVCL_CX2388X_AUDIO_LEFT	Grab audio as two mono buffers. Shows, that it is necessary to grab left channel.

In case, RVCL_CX2388X_AUDIO_RIGHT | RVCL_CX2388X_AUDIO_LEFT are set simultaneously, then to the first half of a buffer the left channel will be captured and to the second half – the right.

dwBitsPerSample

Number of bits per sample: 8, 16 or 32.

dwSamples

Number of samples. This value must be divisible by 4.

pBufferPtr

Pointer to buffer where audio block will be placed.

dwBufferSize

Size of buffer where audio block will be placed. Size of buffer has to be equal at least to (Samples) * (Number of bits per sample) / 8.

Remark:

For **Cx2388x** are possible the following combinations of values of this structure:

Description	*dwFlags*	*dwBitsPerSample*	*dwSamples*	*dwBufferSize*
Will grab 51200 bytes of stereo sound.	RVCL_CX2388X_AUDIO_STEREO	32	12800	51200
Will grab 25600 bytes of mono sound.	RVCL_CX2388X_AUDIO_RIGHT	16	25600	51200
Will grab 25600 bytes of mono sound.	RVCL_CX2388X_AUDIO_LEFT	16	25600	51200
Will grab 51200 bytes of mono sound.	RVCL_CX2388X_AUDIO_RIGHT| RVCL_CX2388X_AUDIO_LEFT	16	25600	51200

SRvclGpioControl

The **SRvclGpioControl** structure contains GPIO port parameters information.

```
typedef struct _SRvclGpioControl
{
	DWORD dwSize;
	DWORD dwMask;
	DWORD dwOutputEnableMask;
	DWORD dwOutputEnable;
	DWORD dwDataMask;
	DWORD dwData;
} SRvclGpioControl;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure using.

dwMask

Set of flags that specify which members of this structure contain data to be set or which members are being requested. This member can have one or more of the following flags set:

RVCL_GPIO_SET_OUTPUT_ENABLE	The **dwOutputEnableMask** and **dwOutputEnable** members are valid and contain GPIO ***output enable*** register to set.
RVCL_GPIO_SET_DATA	The **dwDataMask** and **dwData** members are valid and contain GPIO ***data*** register to set.
RVCL_GPIO_GET_OUTPUT_ENABLE	The **dwOutputEnable** member must be filled in with GPIO ***output enable*** register.
RVCL_GPIO_GET_DATA	The **dwData** member must be filled in with GPIO ***data*** register.

dwOutputEnableMask

Specifies GPIO ***output enable*** mask when RVCL_GPIO_SET_OUTPUT_ENABLE flag is set. Only lower 24 bits are significant.

dwOutputEnable

Specifies GPIO ***output enable*** value when RVCL_GPIO_SET_OUTPUT_ENABLE flag is set. Filled in with current GPIO ***output enable*** when RVCL_GPIO_GET_OUTPUT_ENABLE flag is set. Only lower 24 bits are significant.

dwDataMask

Specifies GPIO output ***data*** mask when RVCL_GPIO_SET_DATA flag is set. Only lower 24 bits are significant.

dwData

Specifies GPIO output ***data*** value when RVCL_GPIO_SET_DATA flag is set. Contains current GPIO ***data*** when RVCL_GPIO_GET_DATA flag is set. Only lower 24 bits are significant.

SRvclKeyData

The **SRvclKeyData** structure contains HASP key user data area parameters information.

```
typedef struct _SRvclKeyData
{
	DWORD dwSize;
	DWORD dwDataOffset;
	DWORD dwDataSize;
	BYTE* pDataPtr;
} SRvclKeyData;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure using.

dwDataOffset

Offset from the beginning of the user data area (in bytes).

dwDataSize

Specifies the number of bytes to be read from the user data area.

pDataPtr

Pointer to the buffer that receives the data read from the user data area.

6.3 RVCL Examples

All PENTACON Developer Kit samples use *demo passwords* for initializing the RVCL library. Every developer can modify samples, by entering the passwords, obtained from PENTACON CORPORATION technical support service for working with HASP keys.

RvcViewer

Illustrates basic RVCL functionality: asynchronous frame capturing (for one Bt8xx device at a time), video formats control, color formats control, General Purpose I/O (GPIO) Port control, Deinterlacing.

MultiScreen

Illustrates advanced RVCL functionality: asynchronous frame capturing for all Bt8xx devices, stable video inputs switching, YUV formats support.

7 DVPack™ Library

DVPack™ Library (RDVP) is attended for video streams compressing and decompressing in the time-critical systems.
DVPack™ codec is developed for extremely fast middle-motion video streams processing. It uses all Intel® processor features available such as MMX (multimedia extension), SSE (streaming SIMD extension) and SSE2 technologies.

7.1 System requirements

The DVPack™ Library runs on personal computers that are based on Intel Architecture processors and running Microsoft Windows 95/98/2K/Xp, or Windows NT SP5 operating system.
Minimum requirements:
Intel® Pentium® MMX processor based system with and 32 MB RAM installed. Intel® Image Processing Library is required (included in package).
The DVPack™ Library does not run without MMX support. It hasn't been tested on AMD™ based systems.

7.2 RDVP Design Guide

7.2.1 Library Initialization

To initialize RDVP library call **RdvpInitialize()** function. If the function returns FALSE value further operation are impossible. To get more information call **GetLastError.**

To get library current version information call **RdvpGetVersion** function. This function returns version and built numbers.

Example 1. Getting Library Version.

```
// This example shows how to properly get library version
#include <stdio.h>

SRdvpVersionInfo svi;
memset(&svi, 0, sizeof(svi));
svi.dwSize = SRdvpVersionInfo_V1;

if(RdvpGetVersion(&svi))
{
    printf("Rdvp Library version %i.%i built %i is installed.\n",
            svi.dwMajor, svi.dwMinor, svi.dwBuild );
}
else
{
    // to get error information use GetLastError()
}
```

Result.

```
Rdvp Library version 1.0 built 20 is installed.
```

7.2.2 Available Codecs Enumeration

Each codec supported by RDVP Library has its own identifier which can be obtained by calling **RdvpEnumCodecs** function. This function returns array of codec identifiers. Identifier should be used while creating video stream.
See more detailed information in **Getting Codec Capabilities** and **Creating New Video Stream**.

7.2.3 Supported Video Stream Formats

DVPack™ codec supports following video frame color formats: Grayscale, YUV 4:1:1 Packed, RGB24, RGB32. See ***Table 6-2 Color Formats*** (chapter 6.1.5) for more information.

Minimum and maximum frame resolution and other necessary frame format information can be obtained by calling **RdvpGetCodecCapabilities** function.

Each video frame should be width and height aligned. If width is not aligned compressor returns an error. And if height is not aligned – least image lines would be cut off.

Current version of DVPack™ codec supports no frame rate control and developer should take care of the video frame time stamps. So there is no time restrictions in video frame sequence. The only thing is to fit input and output frame formats into codec capabilities. Each frame in video stream is not required to be the same resolution and color format as previous one.

DVPack™ codec supports color format conversion and image resizing. Color formats conversion is performed accordingly to “color depth reduction” consideration. So, for example, it couldn’t be performed conversion from grayscale to YUV 4:1:1 color format, but from YUV 4:1:1 to grayscale.

Resultant (compressed) frame resolution is a combination of minimums of passed dimensions. For example, if you specify source frame resolution to be 768x288 pixels and compressed frame resolution to be 512x384 pixels, resultant resolution is going to be 512x288 pixels. No image enlarging is ever performed.

See ***7.2.6 Video Stream Processing*** for detailed information.

7.2.4 Getting Codec Capabilities

Description and capabilities of each codec provided by RDVP Library can be obtained by calling **RdvpGetCodecCaps** function. Description contains codec name and version. Codecs have such capabilities like possible input, output and compressed frame color formats, minimum and maximum image resolution, availability and rules of image resizing.

Example 2. Getting Codec Capabilities.

```
// This example shows how to properly enumerate available codecs
// and get their capabilities

CADWORD cadwCodecs;                                     // codec identifiers

SRdvpCodecCaps scc;                                     // codec caps
memset(&scc, 0, sizeof(scc));
scc.dwSize = SRdvpCodecCaps_V1;
// query for all available information
scc.dwCapsMask = RDVP_CODEC_CAPSMASK_ALLINFO;

if(!RdvpEnumCodecs(&cadwCodecs))
{
    // handle error
}

// proceed every codec in library
for(DWORD ix = 0; ix < cadwCodecs.cElem; ix++)
{
    if(!RdvpGetCodecCapabilities(cadwCodecs.pElems[i],&scc)) continue;

    printf("\n%s:\nVersion %i.%i\n", CString(scc.bstrName), scc.dwVersionMajor,
            scc.dwVersionMinor);
    printf("Minimum resolution: %i x %i pixels\n",
            scc.sizeMinResolution.cx, scc.sizeMinResolution.cy);
    printf("Maximum resolution: %i x %i pixels\n",
```

```
            scc.sizeMaxResolution.cx, scc.sizeMaxResolution.cy);
    printf("Horizontal alignment: %i pixels\n", scc.dwHAlign);
    printf("Vertical alignment: %i pixels\n", scc.dwVAlign);
    printf("Input color format mask: %#06x\n", scc.dwInputCFMask);
    printf("Compressed frame color format mask: %#06x\n",
            scc.dwCompressedCFMask);
    printf("Output color format mask: %#06x\n", scc.dwOutputCFMask);

    // free bstr
    ::SysFreeString(scc.bstrName);
}

// free memory by calling CoTaskMemFree() function.
CoTaskMemFree(cadwCodecs.pElems);
```

Result.

```
DVPack:
Version 1.41
Minimum resolution: 16 x 16 pixels
Maximum resolution: 768 x 576 pixels
Horizontal alignment: 16 pixels
Vertical alignment: 16 pixels
Input color format mask: 0x0292
Compressed frame color format mask: 0x0210
Output color format mask: 0x0293
```

7.2.5 Video Streams Management

To get advantage of simultaneous video streams processing there are basic video streams management functions provided.

To process separate video frame sequence you should create a new video stream by calling **RdvpCreateVideoStream** function. Each time you create a new stream you should specify identifier (ID) of codec which will be used in video processing. New stream ID would be returned. It should be used whenever you call functions concerned with video streams utilization.

If you need existing video stream no more then occupied resources should be freed by calling **RdvpFreeVideoStream** function and specifying proper video stream ID. To enumerate all existing video streams call **RdvpEnumVideoStreams** function. To get stream properties and current settings call **RdvpGetStreamInfo** function.

All video stream management and processing functions are multithread-safe.

Example 3. Managing Video Streams.

```
// this example shows how to manage video streams
// assume that we've already enumerated existing codecs
CADWORD cadwCodecs;

// let's create for example 2 streams of each codec
for(DWORD ix = 0; ix < 2*cadwCodecs.cElem; ix++)
{
    DWORD dwStreamID;

    if(!RdvpCreateVideoStream(cadwCodecs.pElems[ix/2], &dwStreamID))
        // handle error
        continue;

    // store dwStreamID for required operations
    ...
}
```

```
// here we do some stream processing
...

// let's enumerate all streams and print their information
CADWORD cadwStreams;                              // streams existing in system

SRdvpStreamInfo ssi;                             // codec caps
memset(&ssi, 0, sizeof(ssi));
ssi.dwSize = SRdvpStreamInfo_V1;
scc.dwCapsMask = RDVP_STREAM_INFOMASK_ALL;    // query for all available
information

SRdvpCodecCaps scc;                              // codec caps
memset(&scc, 0, sizeof(scc));
scc.dwSize = SRdvpCodecCaps_V1;
scc.dwCapsMask = RDVP_CODEC_CAPSMASK_NAME | RDVP_CODEC_CAPSMASK_VERSION; //
codec name and version

if(!RdvpEnumVideoStreams(&cadwStreams))
{
    // handle error
}

// get information and codecs name and print all stream settings
for(ix = 0; ix < cadwStreams.cElem; ix++)
{
    if(!RdvpGetStreamInfo(&ssi))
        // handle error
        continue;

    if(!RdvpGetCodecCaps(ssi.dwCodecID, &ssc))
        // handle error
        continue;

    printf("\nStream ID = %#04x\n", cadwStreams.pElems[ix]);
    printf("Used codec: %s %i.%i\n", CString(scc.bstrName), scc.dwVersionMajor,
scc.dwVersionMinor);
    printf("Stream mode: %#x\n", ssi.dwStreamMode);
    printf("Inter frames count: %i\n", ssi.dwInterFramesCount);

    // free bstr
    ::SysFreeString(scc.bstrName);
}

// free resources
for(ix = 0; ix < cadwStreams.cElem; ix++)
    RdvpFreeVideoStream(cadwStreams.pElems[ix]);

::CoTaskMemFree(cadwStreams.pElems);
```

Result

```
Stream ID = 0x01
Used codec: DVPack 1.41
Stream mode: 0x0
Inter frames count: 34

Stream ID = 0x02
Used codec: DVPack 1.41
Stream mode: 0x1
Inter frames count: 0
```

7.2.6 Video Stream Processing

With DVPack™ Library you can compress and decompress video streams with low time and optimal rate of quality and bitrate. Firstly, a video stream has to be created (see ***7.2.5 Video Streams Management*** for details). Each video stream can be used as coder and decoder stream as well.

Important:
Use one video stream for compressing and decompressing carefully because of each video stream object shares the same data in coder or decoder mode.

Simply, internal pseudoalgorithm of changing video stream mode is:

```
if PreviousMode == CoderMode AND CurrentMode == DecoderMode
then
     ResetAllInternalDataBuffers();
     SetDecoderMode();
     WaitForIntraFrame(); return ERROR otherwise;
```

and conversely

```
if PreviousMode == DecoderMode AND CurrentMode == CoderMode
then
     ResetAllInternalDataBuffers();
     SetCoderMode();
     PrepareToMakeIntraFrame(); return ERROR otherwise;
```

Thus, switching the stream between coder and decoder modes within one time interval is not efficiently. On the other hand, it is recommended to create and use single video stream if you expect the video streams for processing in series.

Read chapters below carefully to get full information.

7.2.7 Compressing Video Stream

This chapter gives you an information about compressing video streams using DVPack™ Library.
To learn necessary steps to compress the video stream see **Stream Compression Basics** chapter.
Frame Format Conversion section explains DVPack™ codec possibilities and rules of internal frame format conversion.

The differences between “difference” and “reference” frame, and basic principles of stream processing are explained in **Coding Technique** section.

7.2.7.1 Coding Technique

DVPack™ codecs use MPEG-similar coding algorithms. Main idea is to compare current and previous frames to determine “non-static” areas, which should be coded using DCT (discrete cosine transform) and arithmetic coding. So, the frame called “difference” will be coded.
To decompress video stream you need the “reference” frame in start of sequence. The full frame area is taken as “non-static” in such frame, so lately only differences can be coded.
Thus, you can’t get immediate access to arbitrary frame in sequence. You should find nearest earlier “reference” frame and move forward to desired frame. So, it’s reasonable to insert “reference” frame within rational distance. The distance depends on such characteristics as computer performance, video stream parameters and necessity of fast access to random frame.
If requirements are not too hard then “reference” frame can be inserted every 200^{th} (or even times more) frame.

7.2.7.2 Video Stream Quality Understanding

The DVPack™ codec supports two quality parameters and below is detailed descriptions of what they are meaning:

- JPEG Quality.
  This parameter varies from 1 to 100 (in percents). It shows how accurate DCT-coefficients quantization is performed. The lower value – the higher data looses during compression, lower result image quality (small details quality, artifacts and so on).

- Sensitiveness.
  This parameter is necessary to decide an area is “non-static”. The lower value, the higher threshold – the more areas of frame are taken as “static” and the lower amount of data is being coded. Be careful – if it is low motion video stream then too low data will be sent and user will see some artifacts being on the screen after decompression.

The goal is to find optimal correlation between these two parameters to get suitable quality and bitrate. Currently, no direct bitrate control is supported. But it can vary from 200 to 6000 bytes per frame depends on specified quality parameters, video stream format and motion activity.

7.2.7.3 Stream Compression Basics

To compress the video stream follow these steps:

1. Choose codec suitable for your requirements.
2. Create a video stream using desired codec.
3. Get source frame with format that fits in selected codec capabilities.
4. Choose compressed frame resolution and color format.
   Read carefully about conversion in **Frame Format Conversions** section.
   Note: It is recommended but not necessary to specify identical characteristics for each compressed frame within the whole stream, because on resolution or color format changes new reference frame will be generated. On the other hand, source frame characteristics may vary.
5. Reserve buffer for compressed frame according to reasons:
   - if you don’t limited with usage of memory reserve **dwDestImageSize = szDestResolution.cx* szDestResolution.cy*(Num Bits of dwDestColorFormat)/8** bytes.
   - otherwise, if you expect a reference frame reserve buffer about 10^{th} part of size calculated above. Difference frame size may vary from $1/200^{th}$ to $1/10^{th}$ of **dwDestImageSize**.
6. Fill **SRdvpCompress** structure with selected parameters and call **RdvpCompress** function.
7. Analyze return code – if it’s **FALSE** then call **GetLastError()** function to get extended error information.
   An **ERROR_INSUFFISIENT_BUFFER** error means, that specified compressed frame size (**dwDestBufferSize**) was exceeded, results of compression are stored in temporary buffer and ***pdwBytesWritten** value is filled with size of compressed data. All you need to do is allocate buffer with size at least equal to ***pdwBytesWritten** value and recall **RdvpCompress** function with new **pDestBuffer** and **dwDestBufferSize**.
   Same error will be returned until you specify proper **dwDestBufferSize**. Then temporary buffer will be flashed into specified compressed frame buffer and stream will be available for next frame processing.
8. Perform these steps for each frame. To get reference frame call **Reset** function before calling **RdvpCompress()** function.
9. Finish working with the stream object by calling **RdvpFreeStream** function.

7.2.7.4 Frame Format Conversion Considerations

While compressing following format conversions could be performed:

- Resolution conversion:
  The resulting resolution of compressed frame is calculated by algorithm:
  szCompressedResolution.cx = min(szSrcResolution.cx, szDestResolution.cx);
  szCompressedResolution.cy = min(szSrcResolution.cy, szDestResolution.cy);
  Thus, no image enlarging is ever performed during compression.
  Be sure to hold specified resolutions within interval **[szMinResolution, szMaxResolution]**.
- Color format conversion:
  Source and destination color formats should fit into corresponding color format masks returned by **RdvpGetCodecCaps** function.
  If destination (compressed frame) color format has lower **bits per pixel** value than source color format, then conversion will be performed. Otherwise, an error **ERROR_INVALID_PARAMETER** will be returned.

7.2.8 Decompressing Video Stream

This chapter gives you an information about decompressing video streams using DVPack™ Library.
To learn necessary steps to decompress the video stream see **Stream Decompression Basics** chapter.
Frame Format Conversions chapter explains DVPack™ codec possibilities and rules of internal frame format conversion.
See **No Destination Mode** chapter to find out about this special stream mode usage.

7.2.8.1 Stream Decompression Basics

To decompress the video stream follow these steps:

1. Call **RdvpGetFrameInfo** function to get compressed frame format information and used codec.
2. Create stream, using specified codec identifier.
3. Choose uncompressed frame color format and allocate destination buffer at least equal to **dwImageSize = sizeResolution.cx*sizeResolution.cy*Number of pixels of selected color format**.
4. Be sure that first frame in the stream is reference one. Otherwise, an error will be returned.
5. Specify decompression mode:
   - if no result is required (for example, positioning to arbitrary frame), specify **RDVP_DECOMPRESS_NODESTINATION** flag to update internal stream buffers only.
6. Call **RdvpDecompress** function. If return value is **TRUE**, then in destination buffer correct decompressed frame is stored. Otherwise, call **GetLastError** to get error code.
7. If the error is **ERROR_INVALID_PARAMETER**, then check input parameters (like fitting desired color format into codec capabilities and compliance of frame codec identifier with streams one) and recall **RdvpDecompress** function.
   If an error has been occurred within the stream – wind stream to the next reference frame. If you don't wind and continue with next difference frame – no decompressed data adequacy is guarantied.
8. Continue until stream is finished. Then free stream by calling **RdvpFreeStream** function.

7.2.8.2 Frame Format Conversion Considerations

While decompressing color format only conversions could be performed. Any destination (uncompressed) color format, that fits into corresponding color format mask returned by **RdvpGetCodecCaps** function, could be specified.

7.2.8.3 No Destination Mode

This mode is used for fast update of stream internal buffers while decompressing. You don't need specify destination buffer and its size, just set **RDVP_DECOMPRESS_NODESTINATION** flag in dwFlags member. To get resulting uncompressed frame you should clear this flag before needed frame decompressing and call **RdvpDecompress**. Internal buffers will be updated with last frame data and then flushed to output buffer.

7.3 RDVP Reference

Functions

Call Type
All RDVP function has **__stdcall** call type.

Parameters
For all functions of RDVP library, where address of SRdvpXXX structure is used as a parameter, it is necessary to define size of structure in bytes in **dwSize** member before function call.

Return Values
All RDVP function returns BOOL. If the function succeeds, the return value is nonzero. If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RdvpInitialize

The **RdvpInitialize** function provides start initialization of RDVP Library.

```
BOOL RdvpInitialize(SRdvpInitialize* pInit)
```

Parameters

pInit

[in,out] Pointer to **SRdvpInitialize** structure. It is necessary to define size of structure in bytes in **dwSize** member before function call.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

RdvpInitialize function must be called once prior to call of any other function of RDVP library.

RdvpGetVersion

The **RdvpVersion** function returns RDVP library version description.

```
BOOL RdvpGetVersion(SRdvpVersionInfo* pVersionInfo)
```

Parameters

pVersionInfo

[out] Pointer to **SRdvpVersionInfo** structure, which will be filled with library version information. It is necessary to define size of structure in bytes in **dwSize** member before function call.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RdvpEnumCodecs

The **RdvpEnumCodecs** function returns array of available codec identifiers.

```
BOOL RdvpEnumCodecs(CADWORD *pcadwCodecs)
```

Parameters

pcadwCodecs

[out] Pointer to **CADWORD** structure, which will be filled with amount and identifiers of available codecs.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

Array of identifiers will be represented in this way:

- **pcadwCodecs->cElem** is filled with amount of available codecs.
- **pcadwCodecs->pElems** points to internally allocated memory with **DWORD** identifiers.

It is necessary to call **::CoTaskMemFree(pcadwCodecs->pElems)** after using array.

RdvpGetCodecCaps

The **RdvpGetCodecCaps** function returns specified codec capabilities.

```
BOOL RdvpGetCodecCaps(
            DWORD dwCodecID,
            SRdvpCodecCaps* pCodecCaps)
```

Parameters

dwCodecID

[in] Codec identifier.

pCodecCaps

[out] Pointer to **SRdvpCodecCaps** structure which will be filled with codec capabilities. Before function call it is necessary to define size of structure in bytes in **dwSize** member and flags of information to be returned in **dwMask** member.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

It is necessary to call **::SysFreeString(pCodecCaps->bstrName)** if **RDVP_CODEC_CAPS_NAME** flag was specified.

RdvpCreateVideoStream

The **RdvpCreateVideoStream** function creates a new instance of video stream.

```
BOOL RdvpCreateVideoStream(
            DWORD dwCodecID,
            DWORD* pdwStreamID)
```

Parameters

dwCodecID

[in] Codec identifier.

pdwStreamID

[out] Pointer to **DWORD** value, which will be filled with created stream identifier.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RdvpFreeVideoStream

The **RdvpFreeVideoStream** function deletes instance of specified stream.

```
BOOL RdvpFreeVideoStream (DWORD dwStreamID)
```

Parameters

dwStreamID

[in] Stream identifier.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RdvpEnumVideoStreams

The **RdvpEnumVideoStreams** function returns an array of existing video streams identifiers.

```
BOOL RdvpEnumVideoStreams(CADWORD *pcadwStreams)
```

Parameters

pcadwStreams

[out] Pointer to **CADWORD** structure, which will be filled with amount and identifiers of existing streams.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

Array of identifiers will be represented in this way:

- **pcadwStreams->cElem** is filled with amount of existing streams.
- **pcadwStreams->pElems** points to internally allocated memory with **DWORD** identifiers.

It is necessary to call **::CoTaskMemFree(pcadwStreams->pElems)** after using array.

RdvpGetStreamInfo

The **RdvpGetStreamInfo** function returns specified stream information.

```
BOOL RdvpGetStreamInfo(
            DWORD dwStreamID,
            SRdvpStreamInfo* pStreamInfo)
```

Parameters

dwStreamID

[in] Stream identifier.

pStreamInfo

[out] Pointer to **SRdvpStreamInfo** structure which will be filled with stream information. Before function call it is necessary to define size of structure in bytes in **dwSize** member and flags of information to be returned in **dwMask** member.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RdvpResetStream

The **RdvpResetStream** function resets state of specified stream.

```
BOOL RdvpResetStream(DWORD dwStreamID)
```

Parameters

dwStreamID

[in] Stream identifier.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

The reference frame will be generated by following **RdvpCompressFrame** function call if stream works in coder mode.
Resetting stream in decoder mode is not useful. But consider that if you decide to do it then reference frame will be requested by following **RdvpDecompressFrame** function call. An error will be returned in case of difference frame.

RdvpCompressFrame

The **RdvpCompressFrame** function performs compression of one frame of specified video stream.

```
BOOL RdvpCompressFrame(
            DWORD dwStreamID,
            SRdvpCompress* pCompress,
            DWORD* pdwBytesWritten)
```

Parameters

dwStreamID

[in] Stream identifier.

pCompress

[in] Pointer to **SRdvpCompress** structure which has to be filled with necessary information required to frame compression.

pdwBytesWritten

[out] Pointer to DWORD value, which will be filled with actual size in bytes of compressed frame.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

Always check an error returned by function.
ERROR_INSUFFICIENT_BUFFER error means that it was size of destination buffer exceeded and **pdwBytesWritten** value is valid. You must specify the destination buffer with proper size (larger or equal to **pdwBytesWritten** value) and call **RdvpCompressFrame** function again to place compressed frame in sufficient destination buffer. Until you specify proper destination function continue to return **ERROR_INSUFFICIENT_BUFFER** error.
Be attentive, no new frame compression will be performed until you take previous compressed frame.

RdvpDecompressFrame

The **RdvpDecompressFrame** function performs compression of one frame of specified video stream.

```
BOOL RdvpDecompressFrame(
            DWORD dwStreamID,
            SRdvpDecompress* pDecompress)
```

Parameters

dwStreamID

[in] Stream identifier.

pDecompress

[in] Pointer to **SRdvpDecompress** structure which has to be filled with necessary information required to frame decompression.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RdvpGetFrameInfo

The **RdvpGetFrameInfo** function returns an information on specified compressed frame.

```
BOOL RdvpGetFrameInfo(
            LPVOID pCompressedFrame,
            DWORD dwCompressedFrameSize,
            SRdvpFrameInfo *pFrameInfo)
```

Parameters

pCompressedFrame

[in] Pointer to start of compressed frame data.

dwCompressedFrameSize

[in] Size in bytes of compressed frame data.

pFrameInfo

[out] Pointer to **SRdvpFrameInfo** structure which will be filled with frame information. It is necessary to define size of structure in bytes in **dwSize** member before function call.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

RdvpReadKeyData

The **RdvpReadKeyData** function reads data from the HASP key user data area.

```
BOOL RdvpReadKeyData(const SRdvpKeyData* pKeyData)
```

Parameters

pKeyData

[in] Pointer to **SRdvpKeyData** structure where reading parameters are set. It is necessary to define size of structure in bytes in **dwSize** member before function calling.

Return Values

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Structures

SRdvpInitialize

The **SRdvpInitialize** structure contains RDVP initialization information.

```
typedef struct _SRdvpInitialize
{
	DWORD dwSize;
	DWORD pdwCustomerPassword[4];
	DWORD dwUserDataSize;
} SRdvpInitialize;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure is used.

pdwCustomerPasswords

Customer passwords.

dwUserDataSize

HASP key user data area size (in bytes). This member is filled in during **RdvpInitialize** call.

SRdvpVersionInfo

The **SRdvpVersionInfo** structure provides information about library version.

```
typedef struct _SRdvpVersionInfo
{
	DWORD dwSize;
	DWORD dwMajor;
	DWORD dwMinor;
	DWORD dwBuild;
} SRdvpCodecCaps;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure is used.

dwMajor

Major version namber.

dwMinor

Minor version number.

dwBuild

Build number.

SRdvpCodecCaps

The **SRdvpCodecCaps** structure contains video codec properties and capabilities.

```
typedef struct _SRdvpCodecCaps
{
	DWORD dwSize;
	DWORD dwMask;
	BSTR  bstrName;
	DWORD dwVersionMajor;
	DWORD dwVersionMinor;
	SIZE  sizeMinResolution;
	SIZE  sizeMaxResolution;
	SIZE  sizeResolutionAlignment;
	DWORD dwInputCFMask;
	DWORD dwCompressedCFMask;
	DWORD dwOutputCFMask;
} SRdvpCodecCaps;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure is used.

dwMask

Set of flags that specify which members of this structure are being requested.

RDVP_CODEC_CAPS_NAME	The **bstrName** member must be filled in.
RDVP_CODEC_CAPS_VERSION	The **dwVersionMajor, dwVersionMinor** members must be filled in.
RDVP_CODEC_CAPS_RESOLUTION	The **sizeMaxResolution**, **sizeMinResolution**, **sizeResolutionAlignment** members must be filled in.
RDVP_CODEC_CAPS_COLORFORMAT	The **dwInputCFMask, dwCompressedCFMask, dwOutputCFMask** members must be filled in.
RDVP_CODEC_CAPS_ALLINFO	All structure members must be filled in.

bstrName

The name of codec. It must be freed with **::SysFreeString** function after using.

dwVersionMajor

Major version namber.

dwVersionMinor

Minor version number.

sizeMinResolution

Minimum size of the image, which can be proceeded.

sizeMaxResolution

Maximum size of the image, which can be proceeded.

sizeResolutionAlignment

Compressing image size alignment.

dwInputCFMask

Set of codes specifying possible color formats of source (before compression) frame.

dwCompressedCFMask

Set of codes specifying possible color formats of compressed frame.

dwOutputCFMask

Set of codes specifying possible color formats of output (after decompression) frame.

Remarks

See **Supported Video Stream Formats** to get information about color format codes.

SRdvpStreamInfo

The **SRdvpStreamInfo** structure contains the video stream properties information.

```
typedef struct _SRdvpStreamInfo
{
	DWORD dwSize;
	DWORD dwCodecID;
	DWORD dwInterFrameCount;
} SRdvpStreamInfo;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure is used.

dwCodecID

Used codec identifier.

dwInterFrameCount

Difference frames count (since last reference frame).

SRdvpFrameInfo

The **SRdvpFrameInfo** structure contains the compressed frame information.

```
typedef struct _SRdvpFrameInfo
{
      DWORD dwSize;
      DWORD dwCodecID;
      DWORD dwFrameType;
      SIZE  sizeResolution;
      DWORD dwColorFormat;
} SRdvpFrameInfo;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure is used.

dwCodecID

Identifier of codec used to compress frame.

dwFrameType

Type of compressed frame.

RDVP_FRAME_REFERENCE	Reference frame contains full set of data required to decompress frame. The stream must be started with reference frame. And it can be used within stream to speedup stream positioning.
RDVP_FRAME_DIFFERENCE	Difference frame contains only data changes since previous frame. It can be never used in beginning of stream.

szResolution

Compressed frame resolution.

dwColorFormat

Compressed frame color format code.

See **Supported Video Stream Formats** to get information about color format codes.

SRdvpCompress

The **SRdvpCompress** structure contains information required to compress frame with **RdvpCompressFrame** function.

```
typedef struct _SRdvpCompress
{
	DWORD  dwSize;
	SIZE   szSrcResolution;
	SIZE   szDestResolution;
	DWORD  dwSrcColorFormat;
	DWORD  dwDestColorFormat;
	LPVOID pSrcBuffer;
	DWORD  dwSrcBufferSize;
	LPVOID pDestBuffer;
	DWORD  dwDestBufferSize;
	SRdvpQualityParam* pQuality;
} SRdvpCompress;
```

Members

dwSize
Size of the structure, in bytes. This member must be initialized before the structure is used.

szSrcResolution
Resolution of source frame.

szDestResolution
Desired compressed frame resolution.

dwSrcColorFormat
Source frame color format code.
The value has to fit in **SRdvpCodecCaps.dwInputCFMask** (bitwise AND produces true).

dwDestColorFormat
Compressed frame color format code.
The value has to fit in **SRdvpCodecCaps.dwCompressedCFMask** (bitwise AND produces true).

pSrcBuffer
Pointer to source frame data buffer.

dwSrcBufferSize
Source frame data buffer size. Size of buffer has to be equal at least to (sizeSrcResolution.cx) * (sizeSrcResolution.cy) * (Number of bits per pixel of source color format) / 8.

pDestBuffer
Pointer to buffer where compressed frame data will be placed.

dwDestBufferSize
Compressed frame data buffer size. See **Stream Compression Basics** for more information.

pQuality
Pointer to **SRdvpQualityParam** structure which has to be filled with quality parameters.
NULL causes frame to be compressed with built-in quality parameters.

Remarks

See **Supported Video Stream Formats** to get information about color format codes.
See **Stream Compression Basics** for more information about parameters usage technique.

SRdvpQualityParam

The **SRdvpQualityParam** structure contains compression quality parameters.

```
typedef struct _SRdvpQualityParam
{
	DWORD dwSize;
	DWORD dwQuality;
	DWORD dwSense;
} SRdvpQualityParam;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure is used.

dwQuality

JPEG-like quality value. Affects image clarity and smoothness. Varies from 1 to 100.

dwSense

Sensitiveness value. Affects amount of data to be decided as “non-static”. Varies from 1 to 10.

Remarks

See **Stream Compression Basics** for more information about parameters usage technique.

SRdvpDecompress

The **SRdvpDecompress** structure contains information required to decompress frame with **RdvpDecompressFrame** function.

```
typedef struct _SRdvpDecompress
{
	DWORD  dwSize;
	LPVOID pSrcBuffer;
	DWORD  dwSrcBufferSize;
	LPVOID pDestBuffer;
	DWORD  dwDestBufferSize;
	DWORD  dwColorFormat;
	DWORD  dwFlags;
} SRdvpDecompress;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure is used.

pSrcBuffer

Pointer to compressed frame data buffer.

dwSrcBufferSize

Compressed frame data buffer size. See **Stream Compression Basics** for more information.

pDestBuffer

Pointer to buffer, where decompressed frame data will be placed.
Not required if **RDVP_DECOMPRESS_NODEST** flag specified.

dwDestBufferSize

Decompressed frame data buffer size.
Size of buffer has to be equal at least to (sizeResolution.cx) * (sizeResolution.cy) * (Number of bits per pixel of uncompressed frame color format) / 8.
Not required if **RDVP_DECOMPRESS_NODEST** flag specified.

dwColorFormat

Uncompressed frame color format code.
The value has to fit in **SRdvpCodecCaps.dwOutputCFMask** (bitwise AND produces true).
Not required if **RDVP_DECOMPRESS_NODEST** flag specified.

dwFlags

Additional flags specifying decompressor work modes.

RDVP_DECOMPRESS_NORMAL	Normal mode
RDVP_DECOMPRESS_NODEST	Update internal buffers only

Remarks

See **Supported Video Stream Formats** to get information about color format codes.
See **Stream Decompression Basics** for more information about parameters usage technique.

SRdvpKeyData

The **SRdvpKeyData** structure contains HASP key user data area parameters information.

```
typedef struct _SRdvpKeyData
{
	DWORD dwSize;
	DWORD dwDataOffset;
	DWORD dwDataSize;
	BYTE* pDataPtr;
} SRdvpKeyData;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure using.

dwDataOffset

Offset from the beginning of the user data area (in bytes).

dwDataSize

Specifies the number of bytes to be read from the user data area.

pDataPtr

Pointer to the buffer that receives the data read from the user data area.

7.4 RDVP Examples

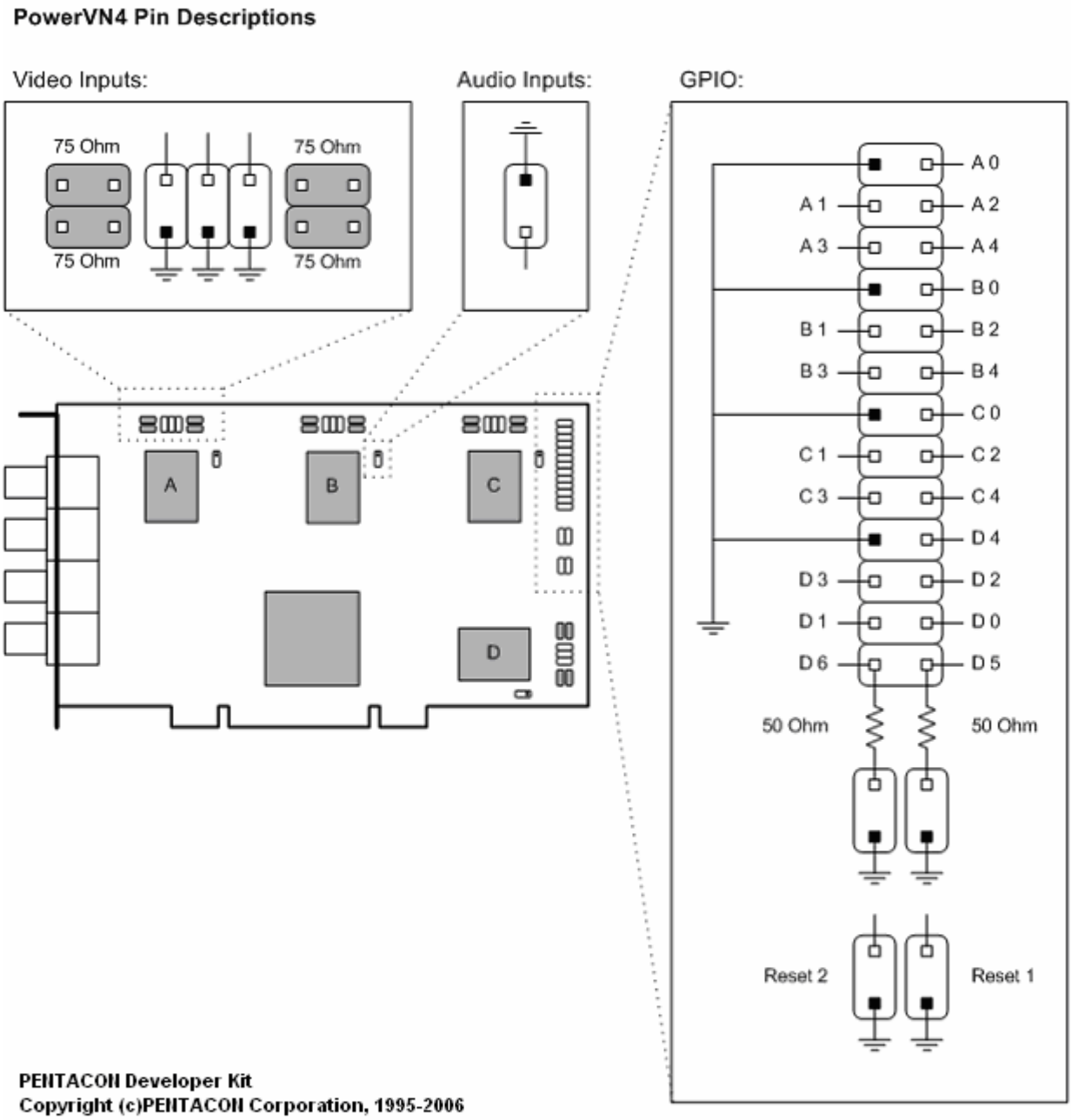
All PENTACON Developer Kit samples use *demo passwords* for initializing the RDVP library. Every developer can modify samples, by entering the passwords, obtained from PENTACON CORPORATION technical support service for working with HASP keys.

RdvpViewer

Illustrates RDVP functionality: compressing and decompressing streams with different frame and quality parameters. Provides to ways of data input: uses RVCL library (if installed) to grab live video or reads video stream from AVI file.

8 Appendix A: Videoblasters Pin Descriptions

Figure 8-1 PowerVN4 Pin Descriptions



Parameter	Min (V)	Max (V)	Sensor State / Output Command *
Input High Voltage (1)	2.4	5.5	Opened
Input Low Voltage (0)	-0.5	0.8	Closed
Output High Voltage (1)	2.4	5	Close
Output Low Voltage (0)		0.4	Open

* Recommended term

9 Appendix B: PowerVN4 Code Samples

Sample 1

This sample illustrates how to determine **PowerVN4** capture boards installed in your computer. It uses the following suppositions:

- Bt8xx devices placed on **PowerVN4** board always have bus number greater or equal to 2.
- Bt8xx devices placed on **PowerVN4** board always have 4, 5, 6 and 7 slot numbers. None other slot numbers are permitted.

The **GetPowerBuses** function returns array of PCI buses which recognized as **PowerVN4** boards.

Parameters:

pBt8xxDevices
[in] Bt8xx devices array.

nBt8xxDevicesCount
[in] Bt8xx devices array size.

pdwPowerBuses
[out] **PowerVN4** PCI buses array.

pnPowerBusesCount
[out] **PowerVN4** PCI buses array size.

```
void GetPowerBuses(
      const SRvclDeviceInfo*  pBt8xxDevices,
      int                     nBt8xxDevicesCount,
      DWORD                   pdwPowerBuses[16],
      int*                    pnPowerBusesCount)
{
      *pnPowerBusesCount = 0;

      // Legal buses map. Key = bus number, value = devices count.
      CMap<DWORD,DWORD,DWORD,DWORD> mapLegalBuses;

      // Illegal buses map. Key = bus number.
      CMap<DWORD,DWORD,DWORD,DWORD> mapIllegalBuses;

      DWORD dwBus,dwDevices,
                  dwMinLegalBus = 0xffff,       // minimum legal bus number
                  dwMaxLegalBus = 0;            // maximum legal bus number


      for(int i=0; i<nBt8xxDevicesCount; i++)
      {
            dwBus = pBt8xxDevices[i].dwPciBusNumber;

            if(mapIllegalBuses.Lookup(dwBus,dwDevices))
                  continue;

            // Bus number m.b. >= 2, slot number - 4..7
            if(   dwBus >= 2 &&
                  pBt8xxDevices[i].dwPciSlotNumber >= 4 &&
                  pBt8xxDevices[i].dwPciSlotNumber <= 7 )
            {
                  // This device can be a part of PowerVN4
                  if(!mapLegalBuses.Lookup(dwBus,dwDevices))
                        dwDevices = 0;

                  mapLegalBuses[dwBus] = dwDevices + 1;

                  dwMinLegalBus = min(dwBus,dwMinLegalBus);
                  dwMaxLegalBus = max(dwBus,dwMaxLegalBus);
```

```
            }
            else
            {
                  // This is illegal bus
                  mapLegalBuses.RemoveKey(dwBus);
                  mapIllegalBuses[dwBus] = 1;
            }
      }

      // Get all PowerVN4 buses (ascending)
      for(dwBus=dwMinLegalBus; dwBus<=dwMaxLegalBus; dwBus++)
      {
            if(mapLegalBuses.Lookup(dwBus,dwDevices) && dwDevices == 4)
            {
                  pdwPowerBuses[*pnPowerBusesCount] = dwBus;
                  (*pnPowerBusesCount)++;
            }
      }
}
```

10 Appendix C: Video Capture Cards Types

Parameters	TinyVN4 family				TitanVN8	
	VN4	VN4 Pro	VN4 Pro2	VN4 Pro3	VN8	VN8 Pro
Chip	Bt8xx	Bt8xx	Cx23880	Cx23880	Bt8xx	Cx23880
Number of chips	1	1	1	1	8	8
Motherboard slot	PCI 32/33	PCI 32/33	PCI 32/33	PCI 32/33	PCI 64/66	PCI Express x1/x4/x8/x16
Video inputs in a real-time mode	1	1	1	1	8	8
Switchable video inputs	4	4	4	4	32	32
S-Video inputs	–	–	1	1	–	8
Audio inputs	1	1	2	2	8	16
Alarm sensors	–	16	–	16	32	32
Relay outputs	–	4	–	4	8	8
LEDs	–	+	–	+	+	+
Watchdogtimer	–	+	–	+	+	+
MB-RIO(4/16)/ MB-RIO2 (4/16) connection	–	+	–	+	+	+
MB-BNC4, MB-RCA4 or MB-DB25 connection	+	+	+	+	+	+
Panel connection for working with *S-Video*	–	–	+	+	–	+

Parameters	PowerVN4 family				
	VN4	VN4 Pro	VN4 Pro2	VN4 Pro3	VN4 Pro4
Chip	Bt8xx	Bt8xx	Cx23880	Cx23880	Cx23880
Number of chips	4	4	4	4	4
Motherboard slot	PCI 32/33	PCI 32/66 (*)	PCI 64/133	PCI Express x1/x4/x8/x16	PCI 32/66 (*)
Video inputs in a real-time mode	4	4	4	4	4
Switchable video inputs	16	16	16	16	16
S-Video inputs	–	–	4	4	4
Audio inputs	4	4	8	8	8
Alarm sensors	16	16	16	16	16
Relay output	4	4	4	4	4
LEDs	+	+	+	+	+
Watchdogtimer	+	+	+	+	+
MB-RIO (4/16)/ MB-RIO2 (4/16) connection	+	+	+	+	+
MB-BNC4, MB-RCA4 or MB-DB25 connection	+	+	+	+	+
Panel connection for working with S-Video	–	–	+	+	+

* By installation **PowerVN4 Pro, PowerVN4 Pro4** cards into motherboard slot, supporting 64/66 standard, bus capability remains 66 MHz.

PENTACON CORPORATION

14, Grivtcova st.,
St.-Petersburg,
190000,
Russia

Tel.: +7 812 448-10-10
Fax : + 7 812 448-10-12

E-mail: welcome@cctv.ru
Internet: www.cctv.ru