INDEX

REVISION HISTORY

REV #	ORIGINATOR	REV DATE	CCN # / COMMENTS
A	Fred Zoghi / Keith Gray	01-19-13	Document origins are from FP300 Rev I. Added Polling raw Information from device
B	Fred Zoghi	01-31-13	Updated Error Codes (5.1)
C	Fred Zoghi	02-28-13	Updated Error Codes (5.1)
D	Fred Zoghi	04-25-13	Added UUID
E	Fred Zoghi	06-11-13	Updated Cancellation flag, Prepare flag and vip_set_num_acq_frames()
F	Fred Zoghi	08-01-13	Added Temperature range table; updated VIP_CURRENT_IAMGE, VIP_CURRENT_IMG_0 and VIP_CURRENT_IMG_1;
G	Dave Smith	10-02-13	Renamed document for Mammo only interface

1. INTRODUCTION

1.1 NEW IN L06

In L06 the Virtual CP has been updated to allow use with dual gain receptors also referred to as ***ConeBeam*** receptors; i.e. those with readout ASICs that support dual gain. In addition to standard single gain modes, a mode may be configured for dual gain where each pixel is read out twice at low and high gain forming a double size image (DUAL_READ), or where the gain is switched dynamically for each individual pixel to select the gain best suited for the exposure value (DYNAMIC_GAIN or dynamic gain switching (DGS)). ***Currently it is intended that L06 will only support DGS modes.*** In DGS modes each pixel is read with either the HI_GAIN state selected (no or relatively low x-ray exposure conditions) or with the LO_GAIN state selected (relatively high x-ray exposure conditions). All pixel information is indicated in the 16-bit value where the 14 LSBs contain the A/D output value and the 15th bit is zero for the HI_GAIN state or 1 for the LO_GAIN state.

A summary of changes is given below.

1.1.1 Receptor configuration

L06 provides special handling for receptor configuration files that include dynamic gain modes. This allows selection of an auxiliary mode that forces the gain to low gain; this is normally only used during calibration procedures, and will not normally be called external to the VCP. The protocol for the call matches that used for the 4030CB – see ***vip_set_mode_acq_type*** below.

1.1.2 Corrections and Pixel Formats

Specialized correction capabilities are now incorporated in the Virtual CP. They are identical to those developed for the 4030CB receptor and K.04 software. The resulting pixel data format may differ from the standard 16-bit unsigned format. A custom format, which is variously referred to as '2MSBs are exponent' or pseudo floating point, is used to store pixel values up to 131064 in 16 bits. Again this closely follows 4030CB protocols.

ConeBeam modes still provides 14-bit data, but it is capable of operating in dual gain modes. In dynamic gain modes the 15th bit specifies lo(0) or hi(1) gain. In addition since the ratio of the gains normally exceeds 4, the range of values in an image cannot be represented as 16-bit integers. A new corrected pixel data format is used for ConeBeam corrected images.

When an image is retrieved using ***vip_get_image***(… VIP_CURRENT_IMAGE…) – RAD modes, or acquired to sequence buffers – FLUORO modes, then the pixel data format my be retrieved by a call to ***vip_get_correction_settings***(…). Possible values for ***PixDataFormat*** are specified in the new definition file **HcpPxlFormat.h**. The mask CB_CURR_FRMT_MASK (0x00FF00) should be used to access the pixel data format. Possible values:

a. CB_FRMT_NORMAL (0x000000) – *16-bit unsigned integer* – normal format for all receptors except ConeBeam (specifically dual read or dynamic gain mode images). ***This is the only format in use for non-ConeBeam receptors.***

b. CB_FRMT_EXP_2 (0x000200) – *2 MSBs are exponent* – the only corrected format currently supported for ConeBeam. Here the 2 most significant bits are interpreted as forming an exponent (value n=0,1,2,3) for the other 14 bits. The pixel value is calculated as:

*pix = mant * 2^n*
where *pix* is the pixel value (0-131,064), *mant* is the 14-bit value formed by the 14 LSBs and *n* is the exponent formed by the 2 MSBs.

c. CB_FRMT_RAW_DYN_GN (0x002000) – raw uncorrected state as acquired by DGS mode where the 15th bit indicates the gain state as high (0) or low (1).

Note other formats specified in **HcpPxlFormat.h** are used only by ViVA not the Virtual CP.

1.1.3 Calibration

Calibration has undergone a significant revision for DGS modes. *Note that non-ConeBeam (single gain) modes should use the same calibration procedures as before.* Changes have been made to simplify the API for the more complicated calibrations required for DGS. For the 4030CB and K.04, calibration was essentially orchestrated by ViVA for DGS modes, and a heavy burden placed on user software to duplicate that effort. Here much of the functionality required for calibration has been shifted to the VCP simplifying the procedures for the user. The new calibration protocol is described below.

The following is intended as an overview and more information may be found in the reference for the specific calls and sample code.

Offset calibrations are unchanged and done autonomously as before using the ***vip_offset_cal*** call. In the case of DGS modes this call results in two sequential offset calibrations – one with regular DGS settings and one with forced low gain (FLG).

For gain and extended gain calibrations, a new flexible calibration interface is introduced in the L06 VCP. The interface provides for flexibility in the way in which data are acquired and processed. However, current usage should conform to the examples in the sample code. Gain calibrations are performed by a sequence of calls:

1. ***vip_gain_cal_prepare*** – this call is made to enter the calibration state. The type of calibration planned must be specified.
2. ***vip_cal_control*** – requests an acquisition segment of a specific type. This command may also specify an acquisition mode, and number of frames. Typically several vip_cal_control calls are made under differing x-ray conditions.
3. A concluding ***vip_cal_control*** call is made to finalize data processing.
4. ***vip_reset_state*** – this call is made to exit the calibration state.

An acquisition segment may consist of more than one data set. For example when performing an extended gain calibration, at each segment both DGS and FLG data are obtained. By design the x-ray level for each acquisition segment must be set so that level 1 maintains all non-defective pixels at the high gain state. Level 2 results in all non-defective pixels being at the low gain state, and level 3 is higher than level 2. Level 3 must be chosen such that pixels remain linear, but high enough to provide a substantial difference from level 2.

The completion of each individual acquisition segment is indicated by 'Complete' being set in the ***vip_query_prog_info*** call. The completion of the entire calibration now requires the user to call ***vip_cal_end***.

If it is required to cancel at any point the calibration may be cancelled by ***vip_reset_state*** to abort an individual acquisition segment without losing prior data. Or to end the procedure and exit the calibration call ***vip_cal_end***.

More detail is provided in section 7 and in the reference info for the named calls in Table 2.1.

1.2 VCP DESCRIPTION

With the software running on the host computer, all calibration files and configuration files must also reside on the host. The VirtCp.dll expects these files to be organized into a fixed tree of subdirectories. The root of this subdirectory tree may be chosen by the user. The recommended configuration (the ViVA default) is to name the directory **C:\IMAGERs**, with one or more subdirectories whose names are the serial numbers of the receptor panels that have been installed. This path is saved in the registry and only needs to be set once on any computer. If not set during an installation, ViVA will set it when first launched.

The interface definition file is '**HcpFuncDefs.h**'. This file depends upon two additional files: '**FluoroStructs.h**' and '**HcpSundries.h**'. HcpFuncDefs.h uses macros in the function declarations so that it can be used in different ways internally by Varian Medical Systems. From the user perspective there are no additional requirements since in the absence of any relevant #defines, it relaxes to user requirements, and should simply be included in the usual way anywhere where the function set described below are used. Additionally the library file **VirtCp.lib** is provided for developer use and also **HcpErrors.h**. The HcpErrors.h file provides error codes in an enum and corresponding error strings in an array. An example of how to safely dereference error codes is provided (commented out) at the bottom of the file. Also in this file are #defines previously in vip_comm.h and also vip_4030R.h.

At run-time a user links the dll **VirtCp.dll** which requires the present of 4 other dlls when a receptor link is opened:
HcpImgAcq.dll – Controls image acquisition and interfaces to I/O devices which may control the x-ray generator.
HcpRecCtrl.dll – Controls the receptor, stores information parsed from receptor configuration file and interfaces to frame grabber module.

HcpCorrections.dll – Performs image corrections and processes calibration data.
HcpCalibration.dll – Controls acquisition of calibration data.

These 4 modules in turn generally have other device-specific dependencies.

As of L04 the VCP supports multiple receptors. The required receptor is specified by the path to its 'IMAGERs' directory as in L01-L03. However, in L01-L03, it was assumed that only one receptor is available at a time. A MAC address was stored in the HcpConfig.ini file, but only to assist with opening in rare cases where firewalls are problematic (this doesn't occur when using the Pleora performance driver anyway). As of L04, the stored MAC address is used to identify the required receptor. The supplied path specifies the HcpConfig.ini file which normally contains a MAC address -- this is true if a receptor has been opened previously using the specified directory. In this case, only the receptor with the specified MAC address can be opened. If a receptor with the specified MAC address is not available, then an open link request generates an error. If no MAC address is specified in the HcpConfig.ini file -- this is true if the directory was just created using the ViVA tool 'Receptor Setup...' -- then any receptor can be opened. But to avoid ambiguity, the open link call is only successful without a MAC address if there only one receptor available on the network at that time. If no MAC address is specified, and more than one receptor is available then an error is generated.

When a vip_open_receptor_link() call returns successfully, the opened receptor is given a zero-based index by the Virtual CP. This index is returned to the caller in the **RcptNum** field of the **SOpenReceptorLink** data structure. This index should be stored by the caller and used to reference the newly opened receptor in future calls to vip_select_receptor. NOTE that the index will be zero for the first receptor opened, etc but the returned value must be checked and not assumed. In situations where various receptors are opened and closed indexes may be recycled. L04 supports a maximum of 4 receptors. The field MaxRcptCount returns this value with the first open link call (Note: subsequent open link calls return zero in this field.).

As of L04 the entry point to VirtCp.dll is protected with a mutex to prevent overlapping calls. An appropriate timeout is used and this may also be boosted by the calling application – see vip_open_receptor_link details.

In general, all function-return values are of type *int*. The return value always indicates the success or failure of the function call. A non-zero value means that an error has occurred. The definitions of the return values are discussed later.

To avoid namespace conflicts, function names are prefixed with **vip_** and constants are prefixed with **VIP_ or HCP_.**

Interface operation is initiated by calling **vip_open_receptor_link**(..). This call requires that a pointer to a structure of type **SOpenReceptorLink** is passed. The only important information generally in this structure is the path to the receptor directory (a sub-directory of IMAGERs). Many of the functions in the current interface use structures where parameter lists were used in older versions. In nearly all cases, functions where the parameters have changed have been renamed. General usage of all structures should follow this example:

```
// standard initialization
SOpenReceptorLink orl;
memset(&orl, 0, sizeof(SOpenReceptorLink));
orl.StructSize = sizeof(SOpenReceptorLink);
// set any members as needed
strncpy(orl.RecDirPath, “C:\IMAGERs\1234-56L”, MAX_STR);
// make the call
int result = vip_open_receptor_link(&orl);
```

In general default values are zero, and only structure members of interest need be set, when this example is followed. Other members of the SOpenReceptorLink structure should be left as zero normally. The StructSize member is normally checked and must always be set as in the example. Failure to set it to a valid value will normally result in error.

Nearly all of the functions *should not be called until a link has been successfully opened*. One exception to this rule is vip_set_debug(TRUE) which will result in a log file being created and can be used ot open a window as in ViVA. Debug messages produced by the dlls are shown and written to a file ‘**HcpDebug.txt**’ when vip_set_debug(FALSE) is called.

In some instances functions are not yet supported while others are present in the user interface, but not currently called by the user since the functionality is handled automatically. These functions are included in the following listing but not described in the subsequent sections. The ‘USEAGE’ column in Table 1-1 indicates mode applicability as COM (common), RAD, FLU(fluoro) or N/A (not available or not applicable).

The VCP front end operates a simple state machine to filter calls. The states which may result in a call being delayed or rejected are when a procdure such as an acquisition or calibration are in progess. In Table 1.1 the column ‘STATE DEP.’ (state dependence) shows how each call is handled if an acquisition or calbration is active. In addition to the stated activity certain calls receive special handling only if the calibration thread is active. Calls may be handled as follows:
OK -- Call is handled normally and immediately.
SE -- An immediate VIP_STATE_ERR is returned.
These are extreme cases. Another common action is to delay handling the call while waiting for the activity to end. (In case a request is received while still winding up a prior procedure.)
DBSE -- ie Delay Before State Error. When handling DBSE calls, the VCP will delay for up 5 seconds to see if the active state clears, so it can handle the call normally. If the state remains active for 5 seconds, then a VIP_STATE_ERR is returned.
DCTA -- ie Delay when CalibrationThread Active. As referred to above certain calls are handled based on a finer distinction of the state. These calls are OK except when the calibration thread is active – for example when during the flat-field or dark field portion of a gain calibration.

Table 1-1 Function Index

		USEAGE	STATE DEP.
1	vip_analog_offset_cal()	COM	DBSE
2	vip_cal_control(..)	COM	DCTA
3	vip_cal_end()	COM	DCTA
4	vip_check_link()	COM	DBSE
5	vip_close_link(..)	COM	DBSE
6	vip_correct_image(..)	COM	DBSE
7	vip_dcds_enable(..)	COM	DBSE
8	vip_enable_auto_cal(..)	N/A	
9	vip_enable_sw_handshaking(..)	RAD	DCTA
10	vip_fluoro_dispose()	N/A	
11	vip_fluoro_get_buffer_ptr(..)	FLU	OK
12	vip_fluoro_get_event_name(..)	FLU	OK
13	vip_fluoro_get_prms(..)	FLU	OK
14	vip_fluoro_grabber_start(..)	FLU	DBSE
15	vip_fluoro_grabber_stop()	FLU	DBSE
16	vip_fluoro_init_mode(..)	FLU	SE
17	vip_fluoro_init_sys(..)	N/A	
18	vip_fluoro_record_start(..)	FLU	DBSE
19	vip_fluoro_record_stop()	FLU	DBSE
20	vip_fluoro_set_prms(..)	FLU	DBSE
21	vip_gain_cal_prepare(..)	COM	DBSE
22	vip_get_analog_offset_info(..)	N/A	
23	vip_get_analog_offset_params(..)	COM	OK
24	vip_get_auto_cal_settings(..)	N/A	
25	vip_get_cal_info(..)	COM	OK
26	vip_get_cal_limits(..)	COM	OK
27	vip_get_config_data	COM	OK
28	vip_get_correction_settings(..)	COM	OK
29	vip_get_current_mode(..)	COM	OK
30	vip_get_dll_version(..)	COM	OK
31	vip_get_gain_scaling_info(..)	N/A	
32	vip_get_hw_config(..)	N/A	
33	vip_get_image(..)	RAD	OK
34	vip_get_image_counts(..)	N/A	
35	vip_get_lih(..)	N/A	
36	vip_get_mode_acq_type	RAD	OK
37	vip_get_mode_info(..)	COM	OK
38	vip_get_num_acq_frames(..)	RAD	OK
39	vip_get_num_cal_frames(..)	COM	OK
40	vip_get_offset_cal_shift(..)	COM	OK
41	vip_get_rad_scaling(..)	N/A	

42	vip_get_recursive_filter(..)	N/A	
43	vip_get_self_test_log(..)	N/A	
44	vip_get_sys_info(..)	COM	OK
45	vip_get_sys_mode(..)	COM	OK
46	vip_get_system_version_numbers(..)	COM	OK
47	vip_get_video_timing(..)	N/A	
48	vip_get_vista_parameters(..)	N/A	
49	vip_get_wl(..)	N/A	
50	vip_hw_reset()	COM	OK
51	vip_initialize_media	N/A	
52	vip_io_enable(..)	RAD	DCTA
53	vip_io_permit_exposure(..)	RAD	DCTA
54	vip_io_query_status(..)	RAD	OK
55	vip_offset_cal(..)	COM	DBSE
56	vip_open_receptor_link(..)	COM	DBSE
57	vip_put_config_data(..)	COM	SE
58	vip_put_image(..)	COM	DBSE
59	vip_query_error_info(..)	N/A	
60	vip_query_prog_info(..)	COM	OK
61	vip_reset_state(..)	COM	OK
62	vip_select_mode(..)	COM	DBSE
63	vip_select_receptor(..)	COM	DBSE
64	vip_self_test(..)	N/A	
65	vip_set_analog_offset_params(..)	COM	DBSE
66	vip_set_cal_acq_data(..)	N/A	
67	vip_set_cal_limits(..)	COM	OK
68	vip_set_correction_settings(..)	COM	OK
69	vip_set_debug(..)	COM	DBSE
70	vip_set_frame_rate(..)	COM	DBSE
71	vip_set_gain_scaling_info(..)	N/A	
72	vip_set_hw_config(..)	N/A	
73	vip_set_image_counts(..)	N/A	
74	vip_set_lih(..)	N/A	
75	vip_set_mode_acq_type(..)	N/A	
76	vip_set_num_acq_frames(..)	RAD	DBSE
77	vip_set_num_cal_frames(..)	COM	DBSE
78	vip_set_offset_cal_shift(..)	COM	OK
79	vip_set_rad_scaling(..)	N/A	
80	vip_set_recursive_filter(..)	N/A	
81	vip_set_sys_mode(..)	COM	DBSE
82	vip_set_user_sync(..)	COM	OK
83	vip_set_vista_parameters(..)	N/A	
84	vip_set_wl(..)	N/A	
85	vip_signal_frame_start()	N/A	
86	vip_sw_handshaking(..)	COM	DCTA

87	vip_total_reset_media()	N/A	
88	vip_validate_media(..)	N/A	

2. INTERFACE FUNCTIONS - COMMON

This section lists and describes function calls that are useful for all modes – rad or fluoro. Listed below are short summaries of these VirtCp.dll interface functions.

Each function is referenced by the number in Table 1-1, and the description has subsections containing the following information:

- function name
- function protocol as used in the Visual C++ **VirtCp.dll**
- descriptions of all parameters used in the function
- remarks and notes about the function

Table 2-1 Function Descriptions - Common

1	**vip_analog_offset_cal()**	
	Protocol	int vip_analog_offset_cal(*int modeNum*);
	Parameters	*modeNum* Specifies the mode number for which the analog offset is requested.
	Remarks	Currently can only specify the current mode.
2	**vip_cal_control**	
	Protocol	Int vip_cal_control(*ScalCtrl* cc*);

	Parameters	*cc*
		struct SCalCtrl
		{
		int StructSize; // Initialize to //sizeof(SCalCtrl)
		int CtrlType; // specifies acquisition type //requested
		int AccMode; // specifies how frames are //processed
		int NumFrames; // number of frames to //acquire; zero default implies //number of calibration frames or //acquisition frames if rad gain

		float GainRatio; // returned when data
		//processing for extended gain
		//cal is requested using
		CtrlType //=
		HCP_CTRLTYPE_EXTCAL
		int Reserved1;
		int Reserved2;
		int Reserved3;
		};
		
		CtrlType
		Specifies the type of acquisition segment requested, and must be appropriate for the initialized calibration specified in the prior ***vip_gain_cal_prepare*** call. Should be one of the values in the first enum in the ***vip_cal_control*** section of the ***HcpSundries.h*** file.
		
		AccMode
		Specifies the acquisition mode. It determines how data are handled in the acquisition object. Each option combines whether any prior data in the object are to be reset before the acquisition or summed to it, and whether the data are sent for storage immediately at the end of the acquisition segment or held in memory. Should be one of the values in the second enum in the ***vip_cal_control*** section of the ***HcpSundries.h*** file.
		
		NumFrames
		Specifies the number of frames to acquire. If not set but left at the default value of zero then the standard number of calibration frames is used excepting for rad mode flatfields where the number of acquisition frames is used.
		
		GainRatio
		When performing an extended gain calibration, and the data processing step is reached, i.e. CtrlType= HCP_CTRLTYPE_EXTCAL, then the gain ratio is returned here. If the acquisition involved acquisition segment HCP_CTRLTYPE_EXT1, then the value is the

		newly calculated one; otherwise it is the same value as used previously.
	Remarks	Only used currently in conjunction with gain cals and extended gain cals for DGS modes. See discussion in section 7 for more detail and example in FluoroTest.
3	**vip_cal_end()**	
	Protocol	int vip_cal_end();
	Parameters	None
	Remarks	This call is used to exit the calibration state. It is normally generated automatically when a calibration concludes and data are processed. However, an explict call is strongly recommended. It may be used to abort a calibration. All data held in memory are deleted. Those already stored are kept. To end an acquisition segment without losing prior data acquired in other segments, use ***vip_reset_state***.
4	**vip_check_link()**	
	Protocol	int vip_check_link(*SCheckLink* linkCheck*);

	Parameters	*linkCheck*
		struct SCheckLink
		{
		int StructSize; // Initialize to
		//sizeof(SOpenReceptorLink)
		int ImgMedianVal; // result of check_link –
		//image Median
		float ImgStdDev; // result of check_link –
		// image StdDev
		int ImgMedLoLim; // lo limit of acceptable
		// median – zero default implies 100
		int ImgMedHiLim; // hi limit of acceptable
		// median – zero default implies value
		// derived from receptor configuration
		float ImgMedSDRatioLim; // Acceptable ratio
		// Median / StdDev - zero default implies
		// that it must not be below 2
		int NumImgAcq; // number of images to // acquire - zero default is interpreted as 1
		int ChkLnkType;
		int Reserved2;
		int Reserved3;

		int Reserved4; }; *ImgMedianVal* A value returned by the Virtual CP representing the median value from a part of the image. The area analyzed is a central part of the image, where the fraction is ¼ x ¼ of the full image dimensions. *ImgStdDev* A value returned by the Virtual CP representing the standard deviation value from a part of the image. The area analyzed is as defined above. *ImgMedLoLim* A parameter specifying a lower threshold for an acceptable median value. (Test #1) *ImgMedHiLim* A parameter specifying an upper threshold for an acceptable median value. (Test #2) *ImgMedSDRatioLim* A parameter specifying a lower limit for the ratio of the median divided by the standard deviation. (Test #3) *NumImgAcq* The number of images to be acquired. The last one is analyzed. *ChkLnkType* May be used to specify the 'flavor' of the vip_check_link call in certain cases such as rad panels where the receptor is operating with 'fixed frame rate' – not settable through thte API call vip_set_frame_rate. The ChkLinkType may have certain values defined in HcpSundries.h: HCP_CHKLNK_SHRT – In this case the time for the call is minimized and only one frame is captured and analyzed. HCP_CHKLNK_LONG – In this case 2 frames are captured. This allows the frame period to be determined and used later where needed as in calculating timeouts. The default for ChkLinkType is interpreted as normally resulting in the 'SHRT' case except for the first time vip_check_link is called after the process initiates.
	Remarks	This function verifies that a link to a receptor

		is available. It forces the acquisition of one or more uncorrected images and analyzes the last. It then analyzes the image as specified above and applies three tests. If any of the tests fails, an error is returned. If all three tests are passed, HCP_NO_ERR is returned.
5	**vip_close_link()**	
	Protocol	int vip_close_link(*int recNum=0*);
	Parameters	*recNum* For single receptor installations this parameter can be defaulted to zero. In multiple receptor installations, it specifies the receptor number to close (zero-based). To be more precise, values 0-255 are reserved for the receptor index (though a maximum of 4 receptors are supported currently), and higher bits are used to set flags to specify other activity. These flags are defined in the vip_close_link section of HcpSundries.h. An earlier interface protocol provided that a negative value for recNum would be interpreted as resulting in a power shutdown where the USB I/O box is in use. This option is still recognized, but it also closes all links.
	Remarks	Should only be called when the acquisition session is finished. This function frees some of the resources and memory associated with the receptor. Memory allocated for sequence buffers is by default not deallocated until the VirtCp is detached. This behavior can be modified by setting the CLSLNK_RELMEM flag on close link, and also when setting sequence parameters using the vip_fluoro_set_prms call (see 4.3.1).
6	**vip_correct_image()**	
	Protocol	int vip_correct_image(SCorrectImage* corrImg);

	Parameters	*corrImg*
		struct SCorrectImage
		{
		Int StructSize;
		// Initialize to sizeof(SCorrectImage)
		WORD* BufIn;
		int BufInX;

		int BufInY; WORD* BufOut; int BufOutX; int BufOutY; int CorrType; int Reserved1; }; *BufIn* Pointer to the buffer containing the image to be corrected. *BufInX* X dimension of the BufIn. *BufInY* Y dimension of the BufIn. *BufOut* Pointer to the buffer where the corrected image is to be written. May be the same as *BufIn*. *BufOutX* X dimension of the BufOut. *BufOutY* Y dimension of the BufOut. *CorrType* Set to zero. Currently ignored.
	Remarks	This function is not normally needed. It may be used, however to perform corrections on an image to which none have been applied already. If used, care should be exercised to ensure that the image was acquired with the receptor and mode currently selected. When called the pre-selected corrections are applied; i.e. those selected in the receptor configuration file or as updated by a prior call to **vip_set_correction_settings()**. Images retrieved by **vip_get_image()** in rad modes automatically have the pre-selected corrections applied. In fluoro modes the pre-selected corrections are automatically applied unless the setting in for CorrType in

		SAcqPrms is HCP_CORR_NONE (see **vip_fluoro_grabber_start()** in Table 4.1). As of L04, image manipulatons are also available in addition to corrections. The required image manipulations – rotate clockwise 90 degrees, mirror flip X direction, and mirror flip Y direction – are also specified in the call to vip_set_correction_settings. Note that various image effects may be achieved by the combination of these e.g. rotate 180 degrees is equivalent to a flipX combined with a flipY. Note also that when the rotate90 is set, the *BufOutX* and *BufOutY* should be interchanged or an error will occur. If any of the requested corrections are not available, an error is generated and the return value indicates what corrections are available. See section 5.2 for additional information.
7	**vip_dcds_enable()**	
	Protocol	int vip_dcds_enable(*BOOL enable)*;
	Parameters	*enable* When set to *FALSE*, DCDS will be turned off. DCDS will automatically turn back on when an offset or gain calibration is initiated.
	Remarks	Normally only used during calibration procedures.
21	**vip_gain_cal_prepare()**	
	Protocol	int vip_gain_cal_prepare(*int mode_num*, *int calType=0)*;
	Parameters	*mode_num* The number of the mode for which gain calibration is to be performed. Should always be the currently selected mode. *calType* This parameter specifies the type of calibration planned. For non-ConeBeam modes it should be zero as defaulted, and calibration procedures are unchanged. For DGS modes it should be set to one of the values in the ***vip_gain_cal_prepare*** section of ***HcpSundries.h*** . See example in FluoroTest.

	Remarks	This call is used to enter the calibration state. More information as to how a gain cal is done is included in the rad and fluoro sections for standard non-ConeBeam (single gain) modes. For DGS modes, this call enters the VCP into a specified calibration state. It should be followed by appropriate ***vip_cal_control*** calls to acquire and process data. Finally the calibration state is exited using ***vip_cal_end***. Note that the second parameter was not used and ignored in previous versions of the VCP. It has been redefined for its current used in this version. Previously it was defaulted to FALSE (0). Code for previous versions should work as before unless this parameter was set to a non-zero value.
22	**vip_get_analog_offset_in()**	
	Protocol	int vip_get_analog_offset_info(*int mode_num, SAnalogOffsetInfo* aop*);

		mode_num
		The number of the mode for which info is requested.
		
		struct SAnalogOffsetInfo
		{
		int
		StructSize;
		// Initialize to sizeof(SAnalogOffsetInfo)
		int AsicNum;
		int AnalogOfstElapsdTime;
		int* AsicOffsets;
		};

	Remarks	NOT YET IMPLEMENTED.
23	**vip_get_analog_offset_params()**	
	Protocol	int vip_get_analog_offset_params(*int mode_num, SAnalogOffsetParams* aop);*
	Parameters	*mode_num* The number of the mode for which info is requested.

		struct SAnalogOffsetParams
		{
		int
		StructSize;
		// Initialize to
		// sizeof(SAnalogOffsetParams)
		int TargetValue;
		int Tolerance;
		int MedianPercent;
		float FracIterDelta;
		int NumIterations;
		};
		
		TargetValue
		The target value which will be the desired result of an analog offset calibration.
		
		Tolerance
		The value around the *TargetValue* that defines the
		acceptable range of values for the analog offset
		calibration.
		
		MedianPercent
		The percentage of pixel values below the target value.
		
		FracIterDelta
		The scaling factor applied in determining the offset adjustment between iterations. This influences the speed of convergence.
		
		NumIterations
		The maximum number of iterations that will be attempted to bring the offsets within range.

	Remarks	This function allows the user to retrieve the parameters that define how the analog offset calibration is performed.
25	**vip_get_cal_info()**	
	Protocol	int vip_get_cal_info(*int mode, SCalInfo* calInfo*);
	Parameters	*mode* Mode for which statistics will be retrieved.

		struct SCalInfo
		{
		int StructSize;
		//Initialize to sizeof(SCalInfo)
		float OfstMedian;
		float OfstStdDev;
		float GainMedian;
		float GainStdDev;
		float GainScaling;
		long Time;
		int GainState;
		int Reserved1;
		int Reserved2;
		int Reserved3;
		int Reserved4;
		};
		
		StructSize
		Must be set by caller to size of structure.
		
		OfstMedian
		The median offset value.
		
		OfstStdDev
		The standard deviation for the offset.
		
		GainMedian
		The median gain value.
		
		GainStdDev
		The standard deviation for the gain.
		
		GainScaling
		The average scaling value applied to each pixel.
		
		Time
		The time of the last calibration cast to a *time_t* type.
		
		GainState
		Specifies the gain state for DGS modes. See HcpSundries.h for possible values.

	Remarks	Retrieves calibration statistics from the VirtCp.dll. For extended gain calibrations use a new struct ScalInfoExGn defined in

		HcpSundries.h and example of use in FluoroTest.
26	**vip_get_cal_limits()**	
	Protocol	int vip_get_cal_limits(*int mode, SCalLimit* calLimits*);

	Parameters	*mode*
		Mode for which calibration limits will be retrieved.
		
		struct SCalLimits
		{
		int StructSize;
		//Initialize to sizeof(SCalLimits)
		int OfstLimitLo;
		int OfstLimitHi;
		int GainLimitLo;
		int GainLimitHi;
		int Reserved1;
		int Reserved2;
		int Reserved3;
		};
		
		StructSize
		Must be set by caller to size of structure.
		
		OfstLimitLo
		The low offset value.
		
		OfstLimitHi
		The high offset value.
		
		GainLimitLo
		The low gain value.
		
		GainLimitHi
		The high gain value.

	Remarks	Retrieves calibration limits.
27	**vip_get_config_data()**	
	Protocol	int vip_get_config_data(*char *full_file_path, char *target_file_name*);
	Parameters	*full_file_path* A null-terminated string which is the full path name of the file where to which the file is to be copied. Up to 256 characters

		are allowed, including the null-termination character. *target_file_name* A null-terminated string which is the name of the file on the Command Processor containing the data. Only receptor configuration files are supported in this function and *target_file_name* must be set to "ConfigDataFile".
	Remarks	This function allows the user to retrieve the current receptor configuration file.
28	**vip_get_correction_settings()**	
	Protocol	int vip_get_correction_settings(*SCorrections* corr*);

	Parameters	struct SCorrections
		{
		int StructSize;
		BOOL Ofst;
		BOOL Gain;
		BOOL Dfct;
		BOOL Line;
		int PixDataFormat; // ignored by //vip_set_correction_settings
		float GainRatio; // ignored by //vip_set_correction_settings
		int Rotate90;
		int FlipX;
		int FlipY;
		int Reserved1;
		};
		
		Ofst
		Set to TRUE if correcting images for pixel offset (FALSE if disabled).
		
		Gain
		Set to TRUE if correcting images for pixel gain (FALSE if disabled).
		
		Dfct
		Set to TRUE if correcting defective pixels, using the defective pixel map (FALSE if disabled).

		Line Maybe set to TRUE for specialized correction of 'T-pixels'. This is not normally required – check with Varian technical staff before use. (Note that the naming 'Line' is historical.) *PixDataFormat* Specifies the pixel data format. Contains (the bitwise OR of) both the original data format and current format. Information is encoded and extracted using the constants and masks defined in HcpPxlFormat.h. For example if the mode is DGS and corrections are ON, then the original data format is CB_ORIG_DYNAMIC_GAIN which is extracted from the PixDataFormat using the CB_ORIG_FRMT_MASK. The current format is CB_FRMT_EXP_2 which is extracted from the PixDataFormat using the CB_CURR_FRMT_MASK or the CB_CORCTD_FRMT_MASK. *GainRatio* Returns the gain ratio for DGS modes. The gain ratio may be determined during an extended gain calibration. *Rotate90* Specifies the resultant image is to be rotated. Note that when calling vip_correct_image or vip_get_image with Rotate90 specified the dimensions of the buffer to receive the image must be interchanged from the mode normal. *FlipX* Specifies the resultant image is to be mirror flipped in the X directions (about the horizontal axis). *FlipY* Specifies the resultant image is to be mirror flipped in the Y directions (about the vertical axis).
	Remarks	This function allows the user to retrieve information as to which correction algorithms are being applied to incoming pixel data. These parameters are global for all modes.

		The PixDataFormat and GainRatio are applicable to the currently selected mode, and only returned for DGS modes. The return value indicates what corrections are available (as AND'd with the corrections requested) for the currently selected mode. See section 5.2 for additional information.
29	**vip_get_current_mode()**	
	Protocol	int vip_get_current_mode(*int* mode_num*);
	Parameters	*mode_num* Returns the currently selected mode.
	Remarks	This function allows the user to retrieve the number of the currently selected mode.
30	**vip_get_dll_version()**	
	Protocol	int vip_get_dll_version(*char* version, char* name, int size*);
	Parameters	*version* Pointer to a string buffer to receive the dll version information. Version info for all Virtual CP dlls is returned in a concatenated mulit-string format such that each string is separated from the next by a NULL byte and the multi-string is terminated by two NULLs. *name* Pointer to a string buffer to receive the dll names. This info is returned as above in a multi-string format with names in the same order as the version info. *size* The size of the two buffers provided for version and name info. The two buffers must be the same size.
	Remarks	This function returns the version information for the dlls, revision letter, build number, date and time. It may also be used to return additional version info such as the Pleora module versions.
37	**vip_get_mode_info()**	
	Protocol	int vip_get_mode_info(*int mdNum, SModeInfo* mdInfo*);

	Parameters	*mdNum*
		Selects the mode for which details are to be retrieved.
		
		struct SModeInfo
		{
		int StructSize;
		int ModeNum;
		int AcqType;
		float FrameRate;
		float AnalogGain;
		int LinesPerFrame;
		int ColsPerFrame;
		int LinesPerPixel;
		int ColsPerPixel;
		char ModeDescription[MAX_STR];
		char DirReadyModeDescription[MAX_STR];
		int DcdsEnable;
		float MxAllowedFrameRate;
		BOOL UserSync;
		int AcqFrmCount;
		int CalFrmCount;
		int GainRoiUpperLeftX;
		int GainRoiUpperLeftY;
		int GainRoiLowerRightX;
		int GainRoiLowerRightY;
		int UncorrectablePixelRepValue;
		int OffsetCalShift;
		int MaxDefectRange;
		int LeanBufferStatus;
		int MammoModeTypeFlag;
		int Reserved2;
		int Reserved1;
		void* ExtInfoPtr;
		int ExtInfoLen;
		};
		
		StructSize
		Must be set by caller to size of structure.
		
		ModeNum
		Mode number for information (same as requested mdNum).
		
		AcqType
		Acquisition type. Variable is set to 0

		(VIP_ACQ_TYPE_CONTINOUS) for fluoroscopy modes or 1 (VIP_ACQ_TYPE_ACCUMULATION) for rad modes. See section 5.6 ***Acquisition Constants***. IMPORTANT: Use the VIP_ACQ_MASK (defined in HcpErrors.h) to safely obtain the fluoro/rad info. Use the VIP_CB_MASK to obtain additional info as to the gain type – NORMAL, DGS or DUAL READ(not supported in L06 currently). *FrameRate* The frame rate. *AnalogGain* The analog gain. *LinesPerFrame* The number of lines per frame of an image i.e. the vertical resolution of the mode. *ColsPerFrame* The number of lines per frame of an image i.e. the horizontal resolution of the mode. *LinesPerPixel* The number of lines per pixel – this is the vertical binning info for the mode. *ColsPerPixel* The number of columns per pixel – this is the horizontal binning info for the mode. *ModeDescription* A string that describes the mode. *DirReadyModeDescription* A string that describes the mode. It is normally based on the ModeDescription but with any characters not permitted in directory names and also spaces removed. *DcdsEnable* The DCDS enable state. *MxAllowedFrameRate*

		The maximum allowed frame rate for the mode. *UserSync* TRUE implies that the user will provide sync pulses to initiate each frame readout cycle. FALSE implies that sync pulses are internally generated at the selected frame rate. *AcqFrmCount* The number of frames to be acquired for each acquisition cycle. Also used for flat field acquisition during rad-mode gain calibrations. *AcqCalCount* The number of frames to be acquired for offset calibration. Also used for flat field acquisition during fluoro-mode gain calibrations. *GainRoiUpperLeftX* Specifies the region used in the gain calibration for median evaluation – upper left X. *GainRoiUpperLeftY* Specifies the region used in the gain calibration for median evaluation – upper left Y. *GainRoiLowerRightX* Specifies the region used in the gain calibration for median evaluation – lower right X. *GainRoiLowerRightY* Specifies the region used in the gain calibration for median evaluation – lower right Y. *UncorrectablePixelRepValue* Specifies the value used to replace any uncorrectable pixels. *OffsetCalShift* Specifies the offset calibration shift if any. Normally zero.

		MaxDefectRange
		Specifies the maximum range defect correction can be attempted.
		
		LeanBufferStatus
		In rad modes with large frame sizes, it is desirable to limit the number of frame buffers allocated which is referred to as the lean buffer mode. The LeanBufferStatus is zero when normal buffer allocation is employed and non-zero when lean buffer allocation is employed. The current number of buffers allocated by default is 5.
		
		`MammoModeTypeFlag`
		`Bitwise flag, used for 3024M to determine if the mode is Prepulse(bit 0x02 is set) or TOMO (bit 0x04 is set).`
		
		ExtInfoPtr
		ExtInfoLen
		These are for internal use – must be zero.

	Remarks	This function allows the user to retrieve detailed mode information. Note there are also additional fields which are not intended for customer use.
39	**vip_get_num_cal_frames()**	
	Protocol	int vip_get_num_cal_frames(*int mode_num*, *int* num_cal_frames*);
	Parameters	*mode_num* The number of the mode for which the number of calibration frames will be retrieved. *num_cal_frames* The number of frames to be accumulated during offset calibration or fluoro-mode flat fields during gain calibration. (Also retrieved as AcqCalCount by vip_get_mode_info()).
	Remarks	This function allows the user to retrieve the number of frames that will be accumulated during calibration.
40	**vip_get_offset_cal_shift()**	
	Protocol	int vip_get_offset_cal_shift(*int mode_num*, *int* offset_cal_shift*);
	Parameters	*mode_num*

		The number of the mode for which the offset calibration shift will be retrieved. *offset_cal_shift* The value to bias the image. This value will be added to all pixels uniformly.
	Remarks	Corrected pixels, which are unsigned, cannot represent values less than zero. The pixels in a dark image are expected to fluctuate both above and below their average offset values. With the default offset shift of 0, any negative fluctuations would be clipped to zero. A small positive offset shift (such as 100) allows most or all of the distribution to be represented (down to -100): the mean value of the distribution should then be equal to the offset shift.
44	**vip_get_sys_info()**	
	Protocol	int vip_get_system_info(*SSysInfo* sysInfo);*

	Parameters	struct SSysInfo
		{
		int StructSize;
		int NumModes;
		int DfltModeNum;
		int MxLinesPerFrame;
		int MxColsPerFrame;
		int MxPixelValue;
		BOOL HasVideo;
		char SysDescription[MAX_STR];
		int StartUpConfig;
		int NumAsics;
		int ReceptorType;
		int BorderPixels;
		int MxImageValue;
		void* DeviceInfoPtr;
		int Reserved2;
		int Reserved3;
		int Reserved4;
		};
		
		StructSize
		Must be set by caller to size of structure.
		
		NumModes
		Set to number of defined modes in the receptor configuration file.

		DefaultModeNum Zero-based index to the default mode. Currently always zero. *MxLinesPerFrame* The number of lines per frame; i.e., the vertical resolution of the receptor with no binning. *MxColsPerFrame* The number of columns per frame; i.e., the horizontal resolution of the receptor with no binning. *MxPixelValue* The maximum value of a pixel. For the 2520E, which has 12-bit A/D conversion, this value will be 4095. For receptors with 14-bit A/D conversion, this value will be 16383. *HasVideo* Expected to be always set to *false*, indicating that the system is not equipped with an analog video board. *SysDescription* A string that describes the system. *StartupConfiguration* An *int*: set to the startup configuration code = 0. *NumAsics* The number of ASICS. For 2520, 1313 and 4030E receptors this is equal to the horizontal resolution divided by the ASIC width = 128. For receptors such as 4030A which have split readout it is twice this value. *ReceptorType* A numerical value corresponding to the type of receptor. *BorderPixels* Number of receptor border pixels regarded as defective. Normally set to zero.

		MxImageValue Maximum value that may be represented in the image. In the current VirtualCP, this is always the same as the MxPixelValue. *DeviceInfoPtr* This may optionally be used to retrieve additional device specific information. It should point to a structure such as SDeviceInfo1 (the only option currently) defined in HcpSundries.h. The StructSize must be set as usual and additionally the StructType must be set to 1 for SDeviceInfo1.
	Remarks	This function allows the user to retrieve the system information.
45	**vip_get_sys_mode()**	
	Protocol	int vip_get_system_mode(*SSysMode* *sysMode*);

	Parameters	struct SSysMode
		{
		int StructSize;
		int SystemMode;
		int CurrentReceptorIndex; // returns // the currently selected // receptor index (zero-based)
		int CurrentMode; // returns the // currently selected mode (zero- // based)
		int NumReceptorsOpen; // returns // the number of receptors // currently open // (return only - ignored by // vip_set_sys_mode)
		int Reserved4;
		int Reserved3;
		int Reserved2;
		int Reserved1;
		};
		
		StructSize
		Must be set by caller to size of structure.
		
		SystemMode
		The system mode. This corresponds to the

		system mode set in the receptor configuration file. Not used by Virtual CP currently. *CurrentReceptorIndex* The index of the currently selected receptor. *CurrentMode* The index of the currently selected mode. *NumReceptorsOpen* The number of receptors currently open.
	Remarks	SystemMode is currently always zero. The call is useful in a multiple receptor installation to establish the current status with respect to system and mode selected as well as how many receptors are open. Normally control software will track this but this call can be useful for example to resolve the ambiguity that arises if multiple receptors are open and the currently selected receptor is closed.
46	**vip_get_system_version_numbers()**	
	Protocol	int vip_get_system_version_number(*int sys_ver_type,* *char* ver_str*);
	Parameters	*sys_ver_type* Version type that is requested by the call. See section 5.3 ***System Number Version Types***. For a given system not all values will be supported and a VIP_NOT_IMPL_ERR may be returned. *ver_str* Pointer to a character array at least 256 characters in length. A string description of the requested version will be returned.
	Remarks	Allows the user to interrogate the systems for various version information.
50	**vip_hw_reset()**	
	Protocol	int vip_hw_reset();
	Parameters	None
	Remarks	Not normally needed. Provides capability to send a reset request to hardware.
55	**vip_offset_cal()**	

	Protocol	int vip_offset_cal(*int mode_num*);
	Parameters	*mode_num* The number of the mode for which offset calibration will be performed.
	Remarks	This function initiates an offset calibration which runs immediately autonomously.
56	**vip_open_receptor_link()**	
	Protocol	vip_open_receptor_link(SOpenReceptorLink* orl);

	Parameters	struct SOpenReceptorLink
		{
		int StructSize;
		void* VcpDatPtr;
		char RecDirPath[MAX_STR];
		int TestMode;
		int DebugMode;
		int RcptNum;
		int MaxRcptCount;
		int MaxModesPerRcpt;
		void* FgTargetPtr; // Normally zero.
		// Custom use by FG module
		int BufferLen; // Normally zero. Use
		// in conjunction with FgTargetPtr
		int SubModeBinX; // Normally zero.
		// Specifies submode binning
		// or sample rate in X direction
		int SubModeBinY; // Normally zero.
		// Specifies submode binning
		// or sample rate in Y direction
		int TimeoutBoostSec; // if permitted
		// this boosts the timeout used by
		// frame grabber -- max 1000
		int TimeoutBoostMsVcpCall;
		BOOL CallMutexOverride;
		int FgCallbackFlag;
		void* FgCallbackPtr;
		int Reserved2;
		int Reserved1;
		};
		
		StructSize
		Must be set by caller to size of structure.
		
		VcpDatPtr

		For internal use only – must be NULL.. *RecDirPath* Must be set to the path to the directory containing information about the receptor e.g. “C:\IMAGERs\1234-56”. *TestMode* Must be set to zero. Opens link in a special test mode where some pixels are overwritten and other test conditions may apply. *DebugMode* Normally zero. May be used to turn on the Debug window feature of the Virtual CP as does vip_set_debug(). See vip_set_debug description for more detail. *RcptNum* This is not to be set by the user. When a link is opened the Virtual CP assigns the receptor index and returns the value here. This value is then used in subsequent calls to reference the receptor with vip_select_receptor and vip_close_link. The open link call specifies which receptor is to be opened using the RecDirPath as always. *MaxRcptCount* The default maximum receptor count is 4, and this may not be increased in L04. *MaxModesPerRcpt* The default maximum mode count is 16. This may be increased when the first receptor is opened by setting this parameter here. *FgTargetPtr* Custom use only. Should be 0 normally. *BufferLen* Custom use only. Should be 0 normally. *SubModeBinX* Custom use only. Should be 0 normally. *SubModeBinY*

		Custom use only. Should be 0 normally. *TimeoutBoostSec* Where permitted may be used to increase the timeout used by the frame grabber when waiting for images. *TimeoutBoostMsVcpCall* Allows the user to boost the timeout used by the VCP interface when waiting for the mutex that protects the API against simultaneous calls. May be used if VIP_STATE_ERRs occur. *CallMutexOverride* This may be set to override the mutex protection. It should only be set when the VCP is called from a single thread. *FgCallbackFlag* It is possible to set up a callback function which will be called in the event that the ethernet link is lost while the frame grabber is active. (after calling vip_grabber_start, and before all requested frames have been processed). If this is required then two fields must be set in the SOpenReceptorLink structure – this one and the following one. Set the FgCallbackFlag to the value HCP_FG_CALLBACK_FLAG as defined in HcpSundries.h. *FgCallbackPtr* If the callback is required then set the flag as above and set FgCallbackPtr to a function pointer which should be defined as void VcpUserCallback(); If the callback is set up then it will be called in the event that a link is lost as described above. The response time may vary but will probably be about 1 second. ***WARNING:*** If the callback is used the function must be designed to return quickly (<100ms) and must not itself call into the VCP API.
	Remarks	This call performs a number of initialization tasks, and prepares the receptor for acquisition. Must be called before almost any other call.

		The return value indicates what corrections are available (as AND'd with the corrections requested) for mode 0. See section 5.2 for additional information.
57	**vip_put_config_data()**	
	Protocol	int vip_put_config_data(*char *full_file_path, char *target_file_name*);
	Parameters	*full_file_path* A null-terminated string which is the full path name of the file which is to become the new receptor configuration file. Up to 256 characters are allowed, including the null-termination character. *target_file_name* A null-terminated string which is the name of the file on the Command Processor containing the data. This version supports receptor configuration files - setting *target_file_name* to "ConfigDataFile". Additionally it supports receptor firmware files – setting *target_file_name* to "RcptFirmware".
	Remarks	This function allows the user to set a new receptor configuration file. Note that the old one will be overwritten. Also it is the user's responsibility to re-sync the Virtual CP by closing and re-opening the link immediately afterwards. Firmware download may take an extended time, and the vip_put_config_data() is configured to return immediately when a firmware download is requested. The user may poll the VirtCp.dll with calls to vip_query_prog_info using the SQueryProgInfoFw structure. The values in this structure report progress. The ProgressLimit value represents the number of operations required and the ProgressCurrent the number done. These values can be used as the basis of a progress bar or time estimate. 'Complete' is set when the download completes.

58	**vip_put_image()**	
	Protocol	int vip_put_image(*int mode_num*, *int image_type*, i*nt x_size*, *int y_size*, *WORD* image_ptr*);
	Parameters	*mode_num* The number of the mode for which an image is to be transmitted. *image_type* The type of image to be retrieved. See section 5.4 ***Image Types*** in this document for a complete listing of available image types. *x_size* The horizontal size of the image to be retrieved. e.g. for 2520E this must be set to 1536. Units: number of pixels. *y_size* The vertical size of the image to be retrieved. e.g. for 2520E this must be set to 1920. Units: number of pixels. *image_ptr* A pointer to a memory block which holds the image. The block must be at least 2 · x_size · y_size bytes.
	Remarks	This function allows the user to load an image to the Virtual CP. This would typically be a calibration image or a defect map (it cannot be an acquisition image).
60	**vip_query_prog_info()**	
	Protocol	int vip_query_prog_info(int uType, UQueryProgInfo* uq);
	Parameters	*uType* Specifies the type of structure in the union pointed to by uq. Normally (excepted for firmware downloads) must be zero. Values derive from the enum in HcpSundries.h (HCP_U_QPI etc). *uq* Pointer to a structure; generally this is SQueryProgInfo:

		struct SQueryProgInfo // uType = HCP_U_QPI
		{
		`Int StructSize;`
		`int NumFrames;`
		`BOOL Complete;`
		`int NumPulses;`
		`BOOL ReadyForPulse;`
		`int ProgLimit;`
		`char StatusStr[MIN_STR];`
		`int Cancellation;`
		`int Prepare;`
		`int ApiCallFlag;`
		
		};
		
		NumFrames
		The number of frames acquired.
		
		Complete
		If TRUE, the acquisition or calibration process is complete. If FALSE, the process is in progress.
		
		NumPulses
		The number of "pulses" or x-rays on/off sequences that are detected during the calibration.
		
		ReadyForPulse
		If *TRUE*, the VirtCp.dll is ready for the next x-rays "ON" command. If FALSE, the VirtCp is ready for the next x-rays "OFF" command.
		
		Cancellation
		. For 3024M, this flag is not used. Instead call to vip_io_enable(HS_CANCEL) call is used: if the return value is 0 cancellation is successful, if the return value is `HCP_REC_CANCEL_FAILED, then receptor denied the cancellation request and acquisition will continue till completion.`
		
		Prepare
		Normally 0. 3024M doesn't use this flag.
		
		Also for firmware downloads (see vip_put_config_data()) the *uType*=HCP_U_QPIFW, and the following

		structure must be used:
		struct
		SQueryProgInfoFw//uType=HCP_U_QPIX
		{
		int StructSize;// Initialize to
		//sizeof(SQueryProgInfoFw)
		int ProgressCurrent;
		int ProgressLimit;
		BOOL Complete;
		};
		
		ProgressCurrent
		Approximate number of operations completed.
		
		ProgressLimit
		Approximate number of operations required.
		
		Complete
		Set to TRUE when download completes. Use this to determine when download completes not the comparison of *ProgressLimit* to *ProgressCurrent.*
		
		For receptors that support it, the following uType codes retrieve diagnostic data:
		HCP_U_QPIRCPT – receptor identification
		HCP_U_QPIFRAME – frame data
		HCP_U_QPITEMPS – temperatures
		HCP_U_QPIVOLTS – voltages
		HCP_U_QPIDIAGDATA – raw diag data
		These codes may be ORed with the option bitcode
		HCP_U_QPI_CRNT_DIAG_DATA to request current data from the receptor.

	Remarks	This function allows the user to query the VirtCp.dll about its progress during the course of an image acquisition or calibration. It is used normally only during rad mode acquisitions. It is also used during both rad and fluoro mode gain cals. See respective descriptions of vip_sw_handshaking(..) in rad and fluoro mode sections for more information.
61	**vip_reset_state()**	
	Protocol	int vip_reset_state();
	Parameters	None.

	Remarks	This function allows the user to abort any incomplete calibration. IT MUST NOT be used to cancel *acquisitions* – IMAGE LOSS COULD OCCUR. The correct procedure for cancelling an acqusition is described in the remarks at table 3.1 for vip_io_enable. Also see code example in RadPanelTest. Note that when performing calibrations using ***vip_cal_control***, this call aborts only the current acquisition segment. If required, the data obtained in prior segments may be retained and used. To abort the calibration completely and exit the calibration state call ***vip_cal_end***.
62	**vip_select_mode()**	
	Protocol	int vip_select_mode(*int mode_num*);
	Parameters	*mode_num* The number of the mode to be selected.
	Remarks	This function allows the user to select a mode of operation by zero-based index. NOTE: If an error is returned the virtual CP could be in an intermediate state where some modules are out of sync in terms of mode number. The user must reselect a valid mode or reset the link if that is not possible. The return value indicates what corrections are available (as AND'd with the corrections requested) the newly selected mode. See section 5.2 for additional information. If the return value is HCP_DEFECT_FILE_UUID_MISMATCH then the defect correction data's UUID for that mode does not match the UUID of the receptor. Refer to the error messages for details.
63	**vip_select_receptor()**	
	Protocol	int vip_select_receptor(*int recSel*);
	Parameters	*recSel* Specifies the receptor index to be selected. This is the index return by the RecNum parameter in the SOpenReceptorLink structure.
	Remarks	The recSel parameter may also specify the

		mode number (see example in HcpSundries.h – vip_select_receptor section). If no mode number is specified then mode zero is always selected.
65	**vip_set_analog_offset_params()**	
	Protocol	int vip_set_analog_offset_params(*int mode_num, SAnalogOffsetParams* aop);*

	Parameters	*mode_num*
		The number of the mode for which info is provided.
		
		struct SAnalogOffsetParams
		{
		int
		StructSize;
		// Initialize to
		// sizeof(SAnalogOffsetParams)
		int TargetValue;
		int Tolerance;
		int MedianPercent;
		float FracIterDelta;
		int NumIterations;
		};
		
		TargetValue
		The target value which will be the desired result of an analog offset calibration.
		
		Tolerance
		The value around the *TargetValue* that defines the
		acceptable range of values for the analog offset
		calibration.
		
		MedianPercent
		The percentage of pixel values below the target value.
		
		FracIterDelta
		The scaling factor applied in determining the offset adjustment between iterations. This influences the speed of convergence.

		NumIterations The maximum number of iterations that will be attempted to bring the offsets within range.
	Remarks	This function allows the user to set the parameters that define how the analog offset calibration is performed.
67	**vip_set_cal_limits()**	
	Protocol	int vip_set_cal_limits(*int mode, SCalLimits* *calLimits*);

	Parameters	*mode*
		Mode for which calibration limits will be set.
		
		struct SCalLimits
		{
		int StructSize;
		//Initialize to sizeof(SCalLimits)
		int OfstLimitLo;
		int OfstLimitHi;
		int GainLimitLo;
		int GainLimitHi;
		int Reserved1;
		int Reserved2;
		int Reserved3;
		};
		
		StructSize
		Must be set by caller to size of structure.
		
		OfstLimitLo
		The low offset value.
		
		OfstLimitHi
		The high offset value.
		
		GainLimitLo
		The low gain value.
		
		GainLimitHi
		The high gain value.

	Remarks	Sets calibration limits.
68	**vip_set_correction_settings()**	
	Protocol	int vip_set_correction_settings(*SCorrections* *corr*);
	Parameters	struct SCorrections

		{
		int StructSize;
		BOOL Ofst;
		BOOL Gain;
		BOOL Dfct;
		BOOL Line;
		int PixDataFormat; // ignored by
		//vip_set_correction_settings
		float GainRatio; // ignored by
		//vip_set_correction_settings
		int Rotate90;
		int FlipX;
		int FlipY;
		int Reserved4;
		};
		
		Ofst
		Set to TRUE if correcting images for pixel offset (FALSE if disabled).
		
		Gain
		Set to TRUE if correcting images for pixel gain (FALSE if disabled).
		
		Dfct
		Set to TRUE if correcting defective pixels, using the defective pixel map (FALSE if disabled).
		
		Line
		Maybe set to TRUE for specialized correction of 'T-pixels'. This is not normally required – check with Varian technical staff before use. (Note that the naming 'Line' is historical.)
		
		PixDataFormat
		Ignored here.
		
		GainRatio
		Ignored here.
		
		Rotate90
		Specifies the resultant image is to be rotated. Note that when calling vip_correct_image or vip_get_image with Rotate90 specified the dimensions of the buffer to receive the image must be interchanged from the mode normal.

		In the vip_correct_image case this implies the input buffer is specified differently from the output buffer. *FlipX* Specifies the resultant image is to be mirror flipped in the X directions (about the horizontal axis). *FlipY* Specifies the resultant image is to be mirror flipped in the Y directions (about the vertical axis).
	Remarks	This function allows the user to set information as to which correction algorithms are being applied to incoming pixel data. These parameters are global for all modes. The return value indicates what of the requested corrections are available for the currently selected mode. See section 5.2 for additional information.
69	**vip_set_debug()**	
	Protocol	int vip_set_debug(*int enable*);
	Parameters	*enable* Set to one of the following values as defined in HcpSundries.h. HCP_DBG_OFF 0 // no debug HCP_DBG_ON 1 // debug on – output //written to file when //debug is turned off HCP_DBG_ON_FLSH 2// debug on – output //written to file //continuously HCP_DBG_ON_DLG 3// debug on – output //written to file when //debug is turned off and //output to a dialog //window
	Remarks	1. Should normally be zero. 2. When debug is enabled with HCP_DBG_ON, then the debug output of all the modules is recorded and saved to a file when the debug is turned off by a

		vip_set_debug(HCP_DBG_OFF) call. 3. When debug is enabled with HCP_DBG_FLSH, then the debug output of all the modules is recorded and saved to a file continuously while operations are occurring. This mode could be beneficial when trying to find a problem which might cause a program to crash. Debug info up to approximately the time of the crash should be preserved. However, because of time used updating the file, this mode is not preferred when time critical tasks are being performed. 4. When debug is enabled with HCP_DBG_ON_DLG then operation with regard to the file is similar to that with HCP_DBG_ON. In addition, a window is opened to which output is written. ***This mode should only be used with MFC type applications and will not work with console applications.*** The file is saved as 'HcpDebug.txt'. Normally it will be saved to the 'IMAGERs' receptor directory in use. It may also be saved to the most recently used receptor directory or as a last resort if no other path is available when the file is opened to 'C:\temp'. The availability of the path depends on when the debug is turned on/off with respect to calls to open_receptor_link and the debug mode in use determines when the file is opened. A limit of about 10-20MB of text output is set unless HCP_DBG_FLSH is used. Note that excepting that HCP_DBG_FLSH is used, the file is written out only when vip_set_debug(HCP_DBG_OFF) is explicitly called. If the program exits without doing this, the text info is lost. You should arrange your program to automatically make the call to vip_set_debug(HCP_DBG_OFF) when it exits to avoid this. You may only modify the debug mode by

		turning off and the back on (e.g. you cannot just turn on the dialog with HCP_DBG_ON_DLG after activating with HCP_DBG_ON).
70	**vip_set_frame_rate()**	
	Protocol	int vip_set_frame_rate*(int mdNum, double frame_rate*);
	Parameters	*mdNum* The mode number for which the frame rate is to be set. At present must be the currently selected mode. *frame_rate* The frame rate requested (frames per second).
	Remarks	Must not exceed the maximum frame rate for the mode. May also be limited if real-time corrections are employed which will also be dependent on the computer system employed. NOTE: For fixed frame rate receptors such as rad panels, it returns a special error code HCP_FXD_RATE_ERR.
77	**vip_set_num_cal_frames()**	
	Protocol	int vip_set_num_cal_frames(*int mode_num*, *int num_cal_frames*);
	Parameters	*mode_num* The number of the mode for which the number of calibration frames will be set. *num_cal_frames* The number of frames to be accumulated during offset calibration or fluoro-mode flat fields during gain calibration.
	Remarks	This function allows the user to set the number of frames that will be accumulated during calibration.
78	**vip_set_offset_cal_shift()**	
	Protocol	int vip_set_offset_cal_shift(*int mode_num*, *int* offset_cal_shift*);
	Parameters	*mode_num* The number of the mode for which the offset calibration shift will be set. *offset_cal_shift*

		The value to bias the image. This value will be added to all pixels uniformly.
	Remarks	Corrected pixels, which are unsigned, cannot represent values less than zero. The pixels in a dark image are expected to fluctuate both above and below their average offset values. With the default offset shift of 0, any negative fluctuations would be clipped to zero. A small positive offset shift (such as 100) allows most or all of the distribution to be represented (down to -100): the mean value of the distribution should then be equal to the offset shift.
81	**vip_set_sys_mode()**	
	Protocol	int vip_set_system_mode(*SSysMode* sysMode*);

	Parameters	struct SSysMode
		{
		int StructSize;
		int SystemMode;
		int CurrentReceptorIndex; // selects // the currently selected // receptor index (zero-based)
		int CurrentMode; // selects the // currently selected mode (zero- // based)
		int NumReceptorsOpen; // returns // the number of receptors // currently open // (return only – cannot be set)
		int Reserved4;
		int Reserved3;
		int Reserved2;
		int Reserved1;
		};
		
		StructSize
		Must be set by caller to size of structure.
		
		SystemMode
		The system mode. This corresponds to the system mode set in the receptor configuration file. Not used by Virtual CP currently.

		CurrentReceptorIndex The index of the currently selected receptor. *CurrentMode* The index of the currently selected mode. *NumReceptorsOpen* The number of receptors currently open.
	Remarks	SystemMode is currently always zero. The call is useful in a multiple receptor installation - selects a receptor and mode simultaneously. Functionality duplicates that in vip_select_receptor.
82	**vip_set_user_sync()**	
	Protocol	int vip_set_user_sync(*int mdNum, BOOL user_sync*);
	Parameters	*mdNum* The number of the mode for which the user sync will be set. At present must be the current mode. *user_sync* Determines whether the frame start signal is internally generated (FALSE) or supplied by the user (TRUE).
	Remarks	When TRUE the user must supply an appropriate signal to the user sync input. Some receptors such as rad panels do not provide a user sync input. This is intended for fluoro modes.
86	**vip_sw_handshaking()**	
	Protocol	int vip_sw_handshaking(*int signal_type*, *BOOL active*);
	Parameters	*signal_type* See section 5.5 ***Software Handshaking Constants***. Must be either VIP_SW_PREPARE or VIP_SW_VALID_XRAYS. *active* If *TRUE*, the signal will be enabled. If *FALSE*, the signal will be disabled.
	Remarks	This function is used differently for rad and fluoro modes. Refer to the following sections for additional information.

		In rad modes This function is used in place of hardware handshaking signals to coordinate image acquisition with X-ray generation. Setting VIP_SW_PREPARE=*TRUE* signals the Virtual CP to prepare for the acquisition of an X-ray image, but does not indicate whether the X-ray generator is ready. When the X-ray generator is ready, the VIP_SW_VALID_XRAYS= *TRUE* should be set. After the image has been acquired, both of these signals should be set to *FALSE*. NOTE: when the optional I/O interface is in use, this call would be used only for gain calibration (only the VIP_SW_PREPARE option).

3. INTERFACE FUNCTIONS - RAD MODES

3.1 ANALOG OFFSET CALIBRATION WITH RAD PANELS 4343 & 4336

Two possible procedures for calibrating the rad panels were identified, and supported in software. The method below referred to as PlanB is the preferred method and used by default. See also section 6.3.4 for additional information on ini file settings.

PlanA:
1. With initial conditions -- DCDS off, glass_global_comp 2500, asic_global_comp 0 -- determine mean count value = MEAN_VAL.
2. With DCDS off, perform normal ASIC analog offset cal with target = MEAN_VAL - 8192.
3. With DCDS off, perform calibration of asic_global_comp so as to reduce 'lowest count value' to 500.
4. With DCDS on, calibrate DCDS offset to achieve a median count value of 1000 counts.

PlanB:
1. With initial conditions -- DCDS off, glass_global_comp 2500, asic_global_comp 0 -- calibrate asic_global_comp to achieve a mean count of 8192.
2. With DCDS off, perform normal ASIC analog offset cal (target value 1500 -- actually as set in receptor config or API call).
3. With DCDS on, calibrate DCDS offset to achieve a median count value of 1000 counts.

The software implementation has provided for both options with PlanB being selected by default. Also since this is specific to Rad panels, a means to determine when to use the new procedure is

required. Normally PlanB is used when the RP module selected in the HcpConfig.ini file is "HcpRpRad" (or "HcpRpRad*"), otherwise the standard procedure which is PlanB without step 1 is used. The procedure required can be specified in the HcpConfig.ini file as shown below, but if one of the rad panel procedures is selected and the RP module does not support the global offset that will result in an error.
Some minor changes have been made to the overall analog offset algorithms:

- The DCDS target was previously determined as the Target / 2. This is still so except a minimum of 750 is applied. i.e. setting the Target parameter below 1500 will not change the DCDS target. It is also now possible to set the DCDS target in the ini file without this minimum being applied.

- The number of calibration frames used in the analog offset procedure has been changed from the config/API setting of number of calibration frames to the number of acquisition frames for rad modes. This significantly speeds up the calibration without any loss of accuracy. The number of calibration frames required can also be set in the ini file if this behavior is not wanted.

In the PlanA procedure, a low value is determined in step 3. This was modified to a median where the median fraction parameter is scaled. By default it is scaled by 0.1, but this scale factor can also be set in the ini file. If it is set to zero then the algorithm relaxes back to the low value (actually the lowest pixel value above the offset threshold Io).

3.2 RADIOGRAPHIC IMAGE ACQUISITION

The VirtualCP supports two methods of radiographic image acquisition: software handshaking, in which the receptor is operated under software commands (and is therefore only approximately synchronized with an X-ray generator) and hardware handshaking, in which the receptor image acquisition is synchronized with the X-ray generator exposure using hardware signals. Pre-L.04, hardware handshaking required an external I/O box. The I/O box allows the application program to monitor the status of an operator control, which might be a two-position handswitch.

The I/O box interface adds three I/O control functions (all beginning with vip_io) that are available only if the proper HcpIo*.dll (HcpIoAdu200.dll) has been loaded: **vip_io_enable(..)** to enable or disable the hardware handshaking acquisition; **vip_io_query_status(..)** to poll the operator control status, and **vip_io_permit_exposure()**, to be called when the operator control status indicates that an exposure is being requested. If no HcpIo*.dll is loaded, these functions return VIP_NOT_IMPL_ERR and hardware handshaking is not available.

In a system built around the I/O box, the software handshaking functions are primarily used to acquire a dark image for test purposes.

3.3 IMAGE ACQUISITION WITH THE RAD PANELS 4343R, 4336R AND 3024M

The Rad panels 4343R, 4336R and 302M support a direct interface to the X-ray generator, eliminating the I/O box. This improves the synchronization timing, but removes the ability to monitor the operator control inputs. For these receptors, a special HcpIoRad.dll is loaded, in which only the **vip_io_enable(..)** call is active. The **vip_io_query_status(..)** call returns VIP_NOT_IMPL_ERR, while **vip_io_permit_exposure()** returns HCP_NO_ERR but is otherwise not functional.

For the Rad panels and 3024M, when HcpIoRad.dll is loaded, the software handshaking functions are used to acquire a dark image for test purposes.

Special note: during gain calibration, the software handshaking functions DO enable the receptor exposure control logic, so that the software handshaking gain calibration sequence may be used with Rad panels 4343R, 4336R and 3024M. This is the recommended gain calibration method for these receptors. Alternatively, for 4343R and 4336R, the hardware handshaking gain calibration sequence may be used, but without the status information from the I/O box, the level of control is limited.

Additional note: for the Rad panels 4343R and 4336R, the recommended configuration is to load HcpIoRad.dll. If the receptor HcpConfig.ini file is modified not to load HcpIoRad.dll, hardware handshaking will not be available. In that case, the software handshaking functions will enable the receptor exposure control logic for every acquisition. This configuration is provided for test purposes and is not recommended for these two panels. Without HcpIoRad.dll, the system is not able to detect certain types of acquisition errors.

3.4 DESCRIPTION OF RAD FUNCTIONS

This section lists and describes function calls that are useful for rad modes. Listed below are short summaries of these VirtCp.dll interface functions.

Each function is referenced by the number in Table 1-1, and the description has subsections containing the following information:

- function name
- function protocol as used in the Visual C++ **VirtCp.dll**
- descriptions of all parameters used in the function
- remarks and notes about the function

Table 3-1 Function Descriptions – Rad Modes

9	**vip_enable_sw_handshaking()**	
	Protocol	int vip_enable_sw_handshaking(*BOOL enable*);
	Parameters	*enable* Determines whether subsequent commands are accepted from the I/O interface (if available; *enable*=FALSE) or from software calls to

		vip_sw_handshaking(..) (*enable*=TRUE).
	Remarks	This command may be used to switch between software and hardware handshaking, when hardware handshaking is available. Not needed otherwise. Default behavior is hardware handshaking when an I/O card is present and software handshaking when an I/O card is not present. For L.04, with HcpIoRad.dll loaded, setting software handshaking TRUE allows the vip_sw_handshaking calls to be used to acquire an image with the receptor exposure control logic disabled (a dark image). Setting software hand-shaking FALSE allows the vip_io_enable calls to be used to enable the receptor exposure control logic and acquire an exposed image.
33	**vip_get_image**()	
	Protocol	int vip_get_image(*int mode_num*, *int image_type*, *int x_size*, *int y_size*, *WORD* image_ptr*);
	Parameters	*mode_num* The number of the mode for which the image is to be retrieved. NOTE: this parameter should normally be the current mode number; the selection is significant only when retrieving an offset or gain calibration image. *image_type* The type of image to be retrieved. See section 5.4 ***Image Types*** in this document for a complete listing of available image types. *x_size* The horizontal size of the image to be retrieved. e.g. for 2520E this must be set to 1536. Units: number of pixels. *y_size* The vertical size of the image to be retrieved. e.g. for 2520E this must be set to 1920. Units: number of pixels. *image_ptr* A pointer to a memory block which will receive the image. The block must be at least of size 2*x_size*y_size bytes.
	Remarks	The function allows the user to retrieve an image that was acquired with the frame grabber. When

		called the returned image has the specified corrections applied. Where supported, this may be used to retransmit the image from the receptor, by using the HCP_RESEND_TARGET_FLAG. In this case the image is corrected as normal. Also this may be used to resend an image stored on the receptor. It may be used in conjunction with the vip_fluoro_get_prms call which provides the range of available indices. (Although the vip_fluoro_get_prms is normally used for fluoro modes, it may in this case also be used for rad modes.)
36	**vip_get_mode_acq_type()**	
	Protocol	int vip_get_mode_acq_type(*int mode_num*, *int* mode_acq_type*, *int* num_frames*);
	Parameters	*mode_num* The number of the mode for which the mode acquisition type will be retrieved. *mode_acq_type* This is always set to VIP_VALID_XRAYS_N_FRAMES (=0). *num_frames* The number of frames used to terminate an acquisition process.
	Remarks	This function allows the user to retrieve the mode acquisition type for a specified mode.
38	**vip_get_num_acq_frames()**	
	Protocol	int vip_get_num_acq_frames(*int mode_num*, *int* num_acq_frames*);
	Parameters	*mode_num* The number of the mode for which the number of acquired frames will be retrieved. *num_acq_frames* The number of frames to be accumulated during acquisition. (Also retrieved as AcqFrmCount by vip_get_mode_info()).
	Remarks	This function allows the user to retrieve the number of frames that will be accumulated during

		acquisition. For fixed frame rate rad panels with distinct acquisition protocol, this value may only be set to 1. This call is effectively of no value.
52	**vip_io_enable()**	
	Protocol	int vip_io_enable(*int activeMode*);
	Parameters	*activeMode* Must be one of the I/O enable codes from Table 5.7. HcpIoRad.dll allows codes: HS_STANDBY, HS_ACTIVE and HS_CANCEL.
	Remarks	Returns VIP_NOT_IMPL_ERR if **Hcplo*.dll** is not loaded. With HcpIoRad.dll, HS_ACTIVE enables the receptor for one acquisition. After each acquisition, an HS_STANDBY call must be made to reset the system. The HS_STANDBY may also be used to disable the receptor exposure logic before an exposure occurs (that is, while vip_query_prog_info returns ReadyForPulse=1 and NumFrames=0). After HS_STANDBY, vip_query_prog_info should be polled until until ReadyForPulse=0 (receptor is disabled) or NumFrames>0 is detected (exposure occurs). If an exposure occurs, it will not be possible to disable the receptor until the exposure sequence has been fully captured. The HS_CANCEL is an alternative for disabling the receptor– it should be followed by HS_STANDBY.
53	**vip_io_permit_exposure()**	
	Protocol	int vip_io_permit_exposure();
	Parameters	None.
	Remarks	With HcpIoRad.dll, this call has no effect, but returns HCP_NO_ERR for compatibility. With HcpIoAdu200.dll (L.01/L.03), this call is required as a safety feature. The user application must grant permission each time the X-ray generator is to be triggered. This should be called whenever the application is ready to handle an image and vip_io_query_status() reads back an exposure state code of EXP_AWAITING_PERMISSION. The state code changes to EXP_PERMITTED as soon as this call is made. Returns VIP_NOT_IMPL_ERR if **Hcplo*.dll** is not loaded.
54	**vip_io_query_status()**	
	Protocol	int vip_io_query_status(*int *ioState, int *exposureState*);

	Parameters	*ioState* Pointer to *int*: set to the current state of the I/O control state machine (encoded according to table 5.8). *expState* Pointer to int: set to the current state of the exposure control state machine (encoded according to table 5.9). NULL may be used if this information is not required.
	Remarks	With HcpIoRad.dll loaded, this call is not supported: VIP_NOT_IMPL_ERR is returned. This call is supported only by L01/L03 when HcpIoAdu200.dll is loaded.
75	**vip_set_mode_acq_type()**	
	Protocol	int vip_set_mode_acq_type(*int mode_num*, *int mode_acq_type*, *int num_frames*);
	Parameters	*mode_num* The number of the mode for which the mode acquisition type will be set. *mode_acq_type* This is always set to VIP_VALID_XRAYS_N_FRAMES (=0). *num_frames* The number of frames used to terminate an acquisition process.
	Remarks	NOT IMPLEMENTED since mode_acq_type is not settable. Use vip_set_num_acq_frames to set number of acquisition frames.
76	**vip_set_num_acq_frames()**	
	Protocol	int vip_set_num_acq_frames(*int mode_num*, *int num_acq_frames*);
	Parameters	*mode_num* The number of the mode for which the number of acquired frames will be set. *num_acq_frames* The number of frames to be accumulated during acquisition.
	Remarks	This function allows the user to set the number of frames that will be accumulated during

		acquisition. This command is not used by 3024M. For 3024M, the number of acquisition frames is fixed and can't be changed.
86	**vip_sw_handshaking()**	
	Protocol	int vip_sw_handshaking(*int signal_type*, *BOOL active*);
	Parameters	*signal_type* See section 5.5 ***Software Handshaking Constants***. Must be either VIP_SW_PREPARE or VIP_SW_VALID_XRAYS. *active* If *TRUE*, the signal will be enabled. If *FALSE*, the signal will be disabled.
	Remarks	In rad modes this function is used in place of hardware handshaking signals to coordinate image acquisition with X-ray generation. Setting VIP_SW_PREPARE=*TRUE* signals the Virtual CP to prepare for the acquisition of an X-ray image, but does not indicate whether the X-ray generator is ready. When the X-ray generator is ready, the VIP_SW_VALID_XRAYS= *TRUE* should be set. After the image has been acquired, both of these signals should be set to *FALSE*. NOTE: when the optional I/O interface is in use, this call would be used only for gain calibration (only the VIP_SW_PREPARE option). This command is used during gain calibration (note different from fluoro modes): 1. A rad gain calibration is begun by issuing the **vip_gain_cal_prepare()** command. 2. X-rays should be off as a dark field calibration is done immediately with: **vip_sw_handshaking(VIP_SW_PREPARE, TRUE).** 3. The VirtCp should then be polled with **vip_query_prog_info(..)** until *NumFrames* is at least the number of calibration frames (determined from e.g. vip_get_mode_info(..) – AcqCalCount). 4. An x-ray flat field is then done which may involve repeated pulses. Either (software handshaking) initiate the x-ray source and send **vip_sw_handshaking(VIP_SW_VALID_XRAYS,**

		TRUE) OR (hardware handshaking) use the handswitch to generate an x-ray pulse. For Rad panels, either software or hardware handshaking may be used. Either call enables a hardware-synchronized exposure. The software handshaking call is recommended for 4343R and 4336R. 5. The VirtCp should then be polled with vip_query_prog_info(..) until *NumFrames* is at least the number of acquisition frames (determined from e.g. vip_get_mode_info(..) – AcqFrmCount). 6. Several pulses may be delivered by repeating steps 4 & 5. 7. To terminate the gain cal send: **vip_sw_handshaking(VIP_SW_PREPARE, FALSE).**

4. INTERFACE FUNCTIONS - FLUORO MODES

4.1 INTRODUCTION TO THE FLUORO INTERFACE

The Virtual CP requires the presence of a frame grabber. The set of calls beginning **vip_fluoro_** are designed around this for use in fluoro modes. The virtual CP allocates buffers for use in conjunction with the frame grabber.

There are normally 2 buffers – 'grab' buffers – allocated which are accessed directly by the frame grabber. It will normally write to these buffers alternately - 'ping-pong' fashion – after a call is made to **vip_fluoro_grabber_start()**, and continue doing so until **vip_fluoro_grabber_stop()** is called. Pleora provide a buffer class which allocates multiple buffers for a single class object. There are also some number of buffers – 'sequence' buffers – allocated which are accessed under programmatic control. The number of sequence buffers can be set by the user (see SSeqPrms structure), but may also be automatically allocated if necessary to a larger number for calibration operations. When **vip_fluoro_record_start()** is called buffers are copied in order of capture from the grab buffers to the sequence buffers – beginning with buffer index 0 – until **vip_fluoro_record_stop()** is called or the requested number of frames have been captured. If a specific number of frames is requested, at least that number of buffers must be allocated. If acquisition is free-running then buffers are written in 'circular' fashion, meaning that once the allocated buffers have been filled, the earliest frames acquired will be overwritten. Note that if this happens and the sequence acquisition is stopped arbitrarily, the sequence 'start index' will generally not be zero. The start index may be discovered after the sequence has stopped by calling **vip_fluoro_get_prms()** referencing SSeqStats. Multi-segment recording is permitted; see section 4.4 for additional information.

The **vip_fluoro_record_start()** has a defaulted parameter – StopAfterN=0 – which may be used to specify or change the number of frames to be acquired. The value zero is interpreted as using the value set in a prior call to **vip_fluoro_set_prms()** referencing SSeqPrms or the default value of zero. Zero is interpreted as free-running acquisition as referred to above.

The user should be aware that setting a non-zero value in the **vip_fluoro_record_start()** call may result in re-allocation of sequence buffers which subjects the acquisition to a possible memory allocation error at a critical point. It is strongly recommended that, for any real data acquisition operation, StopAfterN is set in a call to **vip_fluoro_set_prms()**. StopAfterN in the **vip_fluoro_record_start()** call may be conveniently used for calibration operations if desired. Also see section 4.4 for additional information.

A successful call (return value = HCP_NO_ERR) to **vip_fluoro_grabber_start()** may be interpreted as implying that the frame grabber is ready for x-ray acquisition. This is somewhat analogous to Command Processor systems where a call to **vip_sw_handshaking(VIP_SW_PREPARE, TRUE),** is followed by queries to **vip_query_prog_info(..)** to check that ReadyForPulse is TRUE. Here the successful return to **vip_fluoro_grabber_start()** is considered sufficient. As an extra verification a call may also be made to **vip_fluoro_get_prms()** referencing SLivePrms, and the VideoStatus member checked against the StartUp indicated in SAcqPrms.

Many of the fluoro calls have structure pointers as parameters. Use of these should follow this example:

```
SAcqPrms acqPrm;
memset(&acqPrm, 0, sizeof(SAcqPrms));
acqPrm.StructSize = sizeof(SAcqPrms);

// set any members as required for custom use
// default behavior is defined by zero for each member

int result = vip_fluoro_grabber_start(&acqPrm);
```

4.2 DESCRIPTION OF FLUORO FUNCTIONS

This section lists and describes function calls that are useful for fluoro modes. Listed below are short summaries of these VirtCp.dll interface functions.

Each function is referenced by the number in Table 4-1, and the description has subsections containing the following information:

- function name
- function protocol as used in the Visual C++ **VirtCp.dll**
- descriptions of all parameters used in the function

- remarks and notes about the function

Table 4-1 Function Descriptions – Fluoro Modes

11	**vip_fluoro_get_buffer_ptr()**	
	Protocol	int vip_fluoro_get_buffer_ptr(*WORD** buf, int bufIdx, int bufType=0*);
	Parameters	*buf* This is a pointer to a WORD* which is a variable declared by the user to which the buffer pointer will be written. *bufIdx* This is the buffer index (zero-based) for which the pointer is requested. If no buffer is available for the index specified the return value of the function is HCP_GRAB_ERR. *bufType* This parameter defaults to zero which specifies the sequence buffers as will be the normal usage. Pointers to grab buffers (usually 2) can also be obtained by setting the bufType to 1.
	Remarks	Pointers to buffers should be handled with great care. The buffer size will currently be that specified by the mode (number of pixels x 2 bytes). (Provision is made - in the SSeqPrms structure - for the user to specify an ROI for capture. However this is not supported currently and should not be used. Receptors with dual-gain capabilities such as 4030CB may also involve different buffer sizes, but again these are not currently supported by the Virtual CP). Buffer pointers should not be stored for future use. Except as noted below, the buffer pointer may be considered valid until a new command is sent to the Virtual CP or for the duration of an acquisition. Various commands may involve the re-allocation of buffers; for example mode selection will generally result in re-allocation of all buffers - both grab and sequence. But also calibrations may require re-allocation of sequence buffers. Auto-offset – when available – may break the rule that a pointer is valid until a new command is issued since under auto-offset, offset calibrations launch automatically;

		however, a time limit is specified between prior activity and an automated offset calibration during which the buffer pointer may be used safely. As of build 26, this call is available during an acquisition. The user must ensure that no access is attempted to the buffer being written by the Virtual CP (indexed by the one *after* that specified in SLivePrms).
12	**vip_fluoro_get_event_name()**	
	Protocol	int vip_fluoro_get_event_name(*int eventType, char* eventName*);
	Parameters	*eventType* References a member of the HcpFgEvent enum defined in FluoroStructs.h. The only event type currently intended for the user is HCP_FG_FRM_TO_DISP which is set when a new frame is ready to display. A future type will be defined when the 'Just-In-Time' corrections method is implemented. This event will be set by the user to request another corrected frame. *eventName* A pointer to a character buffer which should be able to hold at least 32 characters. This name may be used in a call to the Windows API CreateEvent().
	Remarks	This function is a generic method for retrieving a reference name to a synchronization object. Current usage is to determine when a frame is ready to display or for other manipulation
13	**vip_fluoro_get_prms()**	
	Protocol	int vip_fluoro_get_prms(*int structType, void* structPtr*);
	Parameters	*structType* References a member of the HcpFluoroStruct enum defined in FluoroStructs.h. *structPtr* A pointer to a structure of the type specified by *structType.*
	Remarks	This function provides a generic method for obtaining various parameter settings. More on usage is provided in section 4.3.

14	**vip_fluoro_grabber_start()**	
	Protocol	int vip_fluoro_grabber_start(*SAcqPrms* acqPrms*);

	Parameters	*acqPrms*
		Pointer to a SAcqPrms structure.
		struct SAcqPrms
		{
		int StructSize;
		int StartUp;
		int ReqType;
		int CorrType;
		void* CorrFuncPtr;
		void* ThresholdSelect;
		double* CopyBegin;
		double* CopyEnd;
		int ArraySize;
		int MarkPixels;
		int FrameErrorTolerance;
		void* LivePrmsPtr;
		};
		
		StructSize – User must set to sizeof(SAcqPrms)
		
		StartUp – Set from HcpFluoroStatus enum in FluoroStructs.h.
		Default = 0 (HCP_REC_IDLE) is interpreted as HCP_REC_GRABBING. Frames are being written to the grab buffers but not saved to the sequence buffers.
		If set to HCP_REC_RECORDING, it automatically calls **vip_fluoro_record_start()**.
		If set to HCP_REC_PAUSED, the grabber is ready to start but no frames are being written to the grab buffers. A subsequent call to **vip_fluoro_record_start()** results in the immediate acquisition of frames to grab and sequence buffers.
		
		ReqType – Reserved use. Must be zero.
		
		CorrType – Specifies real time corrections required. Set from enum HcpCorrType in FluoroStructs.h. Only values currently accepted are HCP_CORR_NONE or HCP_CORR_STD. In the latter case the corrections determined in

		the receptor configuration or updated by **vip_set_correction_settings()** are applied. *CorrFuncPtr* – May be used to specify a custom corrections routine - instead of that integrated into the Virtual CP - that will be called in real-time. Not recommended for most users. If used it would need to point to a function with the same prototype as: **int vip_correct_image(SCorrectImage* corrImg);** Must be set to NULL unless a user defined function is supplied. *ThresholdSelect* – Not implemented; must be NULL. *CopyBegin* – May be set to point to an array of doubles (provided by the user) to which timing information will be written. Each frame captured will have the time (seconds) written here when it becomes available in the grab buffer. This is the same information used by ViVA when building the VideoTimingLog.txt. *CopyEnd* – May be set to point to an array of doubles (provided by the user) to which timing information will be written. Each frame captured will have the time (seconds) written here when it becomes available in the sequence buffer. This is the same information used by ViVA when building the VideoTimingLog.txt. *ArraySize* – The size of the array supplied by the previous two members. If the array is smaller than the number of frames captured, the earliest members of the array are overwritten. *MarkPixels* – This may be used in some debug or test situations. It causes a line of pixels to be overwritten to the value 100 in the grab buffer. The length of the line corresponds to the number of frames captured. The line begins with pixel index 122881 which is (120,1) for a 2x2 mode with a 4030A receptor. When viewing the video it results in a line of

		increasing length 'walking' across the image. Must be zero for all normal usage. *FrameErrorTolerance* – May be set to indicate that some number of frame grabber errors can be ignored. For most application should be zero. *LivePrmsPtr* – This pointer is returned by the Virtual CP. It points to a SLivePrms structure which is allocated when the link opens and de-allocated when it closes. It may be used with caution to get status information at run time without the need for repeated calls into the interface. NOTE: See section 4.3 for more information on the SLivePrms structure members. The *NumFrames, BufIndex* & *BufPtr* are updated in the Virtual CP immediately before the Virtual CP sets the HCP_FG_FRM_TO_DISP event. These members may therefore be safely read immediately after the event is set. The *VideoStatus* and *ErrorCode* could change at any time and hence should be read from the structure with caution. If the value read differs from that expected, it would be a good idea to check it with a call to vip_fluoro_get_prms() referencing SLivePrms.
	Remarks	Starts the grabbing. Frames are being written to the grab buffers (unless started PAUSED). A return value of HCP_NO_ERR implies that the Virtual CP is ready to acquire x-ray images.
15	**vip_fluoro_grabber_stop()**	
	Protocol	int vip_fluoro_grabber_stop();
	Parameters	None
	Remarks	Stops the grabber. May be called without a prior call to **vip_fluoro_record_stop()**.
16	**vip_fluoro_init_mode()**	
	Protocol	int vip_fluoro_init_mode(*SFluoroModePrms* modePrms*);
	Parameters	*modePrms* Pointer to a *SFluoroModePrms* structure. struct SFluoroModePrms {

```
        int   StructSize;
        int   FrameX;
        int   FrameY;
        int   BinX;
        int   BinY;
        int   RecType;// =0 default
        float FrmRate;// =0.0 default (if not
needed)
        int   UserSync;// =0 default
        int   TrigSrc;// =0 default
        int   TrigMode;// =0 default
        void* GrabPrms;// =NULL default - not

        //implemented
        void* TimingPrms;// =NULL default - not
                                    //implemented
        char* FilePath;// =NULL default - path to
                                    //cnfg file
};
```

StructSize – User must set to sizeof(SFluoroModePrms)

FrameX – Frame dimension horizontal.

FrameY – Frame dimension vertical.

BinX – Pixel binning horizontal.

BinY – Pixel binning vertical.

RecType – This specifies the receptor type in use and may be discovered by a call to vip_get_sys_info(). It is used if necessary to trim the frame location relative to the virtual frame.

FrmRate – No longer needed/used here. Should be set to zero.

UserSync – Not significant for most frame grabbers including Bitflow and Pleora.

TrigSrc – Not significant for most frame grabbers including Bitflow and Pleora.

TrigMode – Not significant for most frame

		grabbers including Bitflow and Pleora. *GrabPrms* – This may be set to point to a SGrbPrms structure which should have its StructSize set correctly. If a valid pointer is provided, information will be returned describing the grab buffers: number, dimensions and bytedepth (always 2). If not required this pointer may be left as NULL. *TimingPrms* – Not implemented; ignored. *FilePath* – Not implemented; ignored.
	Remarks	Note that this is called automatically when a vip_select_mode is called (and vip_select_mode(0) is itself generated automatically when vip_open_receptor_link is called). Should not be needed normally.
18	**vip_fluoro_record_start()**	
	Protocol	int vip_fluoro_record_start(*int stopAfterN=0, int* startFromBufIdx=-1);
	Parameters	*stopAfterN* Sets the value for StopAfterN. Zero is interpreted as no change to a previously set value (through a call to vip_fluoro_set_prms() referencing SSeqPrms). If StopAfterN is zero the acquisition is free-running and must be stopped by a call to **vip_fluoro_grabber_stop()** or **vip_fluoro_record_stop()** *startFromBufIdx* Sets the start index for the buffers where the captured frames are to be saved. A negative value will be interpreted as the next available buffer – which is reset to zero when the grabber is stopped and restarted. See section 4.4 for additional information.
	Remarks	Starts the copying of frames from the grab buffers to the sequence buffers with corrections being applied if appropriate.
19	**vip_fluoro_record_stop()**	
	Protocol	int vip_fluoro_record_stop()
	Parameters	None
	Remarks	Stops the recording. No more frames will be

		copied to the sequence buffers.
20	**vip_fluoro_set_prms()**	
	Protocol	int vip_fluoro_set_prms(*int structType, void* structPtr*);
	Parameters	*structType* References a member of the HcpFluoroStruct enum defined in FluoroStructs.h. *structPtr* A pointer to a structure of the type specified by *structType.*
	Remarks	This function provides a generic method for setting various parameters. More on usage is provided in section 4.3.
86	**vip_sw_handshaking()**	
	Protocol	int vip_sw_handshaking(*int signal_type*, *BOOL active*);
	Parameters	*signal_type* See section 5.5 ***Software Handshaking Constants***. Must be either VIP_SW_PREPARE or VIP_SW_VALID_XRAYS. *active* If *TRUE*, the signal will be enabled. If *FALSE*, the signal will be disabled.
	Remarks	In fluoro modes this function is used only in gain cals. (This is different from Command Processor based systems. Acquisitions in fluoro modes should use the vip_fluoro_ command set above.) This command is used during gain calibration (note different from rad modes): 1. A fluoro gain calibration is begun by issuing the **vip_gain_cal_prepare()** command. 2. Next a **vip_sw_handshaking(VIP_SW_PREPARE, TRUE).** 3. The VirtCp should then be polled with **vip_query_prog_info(..)** until *ReadyForPulse* is TRUE. 4. An x-ray flat field is done first and when the receptor is prepared and x-ray beam on the command

		vip_sw_handshaking(VIP_SW_VALID_XRAYS, TRUE) should be called. 5. The VirtCp should then be polled with **vip_query_prog_info(..)** until *NumFrames* is at least the number of calibration frames (determined from e.g. vip_get_mode_info(..) – AcqCalCount). 6. After terminating the x-ray beam and allowing a short interval for residual charge to clear, send **vip_sw_handshaking(VIP_SW_VALID_XRAYS, FALSE)**. 7. The VirtCp should then again be polled with **vip_query_prog_info(..)** until *Complete* is TRUE. 8. Send a **vip_sw_handshaking(VIP_SW_PREPARE, FALSE).**

4.3 FLUORO PARAMETER CALLS

The calls to vip_fluoro_get_prms() and vip_fluoro_set_prms() are essentially generic calls that specify a structure type and a structure pointer. This section describes how to use these calls. The structure types are specified in an enum, HcpFluoroStruct, in FluoroStructs.h. Only the structure types described here are implemented currently, and - except for HCP_FLU_SEQ_PRMS – these only for vip_fluoro_get_prms(). HCP_FLU_SEQ_PRMS is implemented for both vip_fluoro_get_prms() and vip_fluoro_set_prms().

HCP_FLU_SEQ_PRMS

With HCP_FLU_SEQ_PRMS as the structure type, the structure pointer should point to a SSeqPrms structure. This call may be made when the frame grabber is idle to get or set the parameters described below.

```
struct SSeqPrms
{
  int   StructSize;
  int   NumBuffers;       // number request - a smaller number may be allocated
                                  // and returned at this same location
  int   SeqX;             // dflt = 0 interpret = grbX
  int   SeqY;             // dflt = 0 interpret = grbY
  int   SumSize;          // dflt = 0 interpret =1
  int   SampleRate;       // dflt = 0 interpret =1
  int   BinFctr;          // dflt = 0 interpret =1
  int   StopAfterN;       // dflt = 0
```

```
  int   OfstX;              // dflt = 0
  int   OfstY;              // dflt = 0
  int   RqType;             // must be zero
        int   SnugMemory;       // dflt = 0
};
```

NumBuffers
This is the number of sequence buffers to be allocated. If there is insufficient memory available, a smaller number may actually be allocated and the user should check this member when a vip_fluoro_set_prms() call returns.

SeqX, SeqY
These set the size of the buffer required. When zero, these dimensions are determined from the mode dimensions. Otherwise the user may specify the capture of an ROI instead of the whole image. NOT YET SUPPORTED.

SumSize
When set to a number greater than 1, the specified number of frames are integrated to provide each captured frame in a sequence buffer. The capture rate will be correspondingly smaller than the receptor frame rate. NOT YET SUPPORTED.

SampleRate
When set to a number greater than 1, say N, then N-1 frames are ignored for each one captured in a sequence buffer. The capture rate will be correspondingly smaller than the receptor frame rate. NOT YET SUPPORTED.

BinFctr
May be set to either 1 or 2. When set to 2, software binning (2x2) is done. NOT YET SUPPORTED.

StopAfterN
This specifies a number of frames after the capture of which, recording will end. If it is set to zero, then acquisition is free-running, and buffers are overwritten in circular fashion. This parameter may also be set in the vip_fluoro_record_start(); when zero (default) in vip_fluoro_record_start(), it is interpreted as ‘use the prior value’. It is strongly recommended that it only be set in vip_fluoro_record_start() for low importance operations such as calibrations.
See also discussion in 4.1 & 4.4.

OfstX, OfstY
These may be used in conjunction with *SeqX, SeqY* to set an ROI for capture. NOT YET SUPPORTED.

SnugMemory
Arbitrary Value -- implies that the VCP will handle memory in its normal manner (once allocated, it stays allocated until VirtCp.dll detaches OR CLSLNK_RELMEM flag set on close link). When SnugMemory is set to SNUG_MEM_FLAG, then the VCP will deallocate any excess memory above that needed to keep NumBuffers available.

HCP_FLU_LIVE_PRMS

With HCP_FLU_LIVE_PRMS as the structure type, the structure pointer should point to a SLivePrms structure. This call may be made at any time. This structure may only be referenced by vip_fluoro_get_prms(). A copy of the global SLivePrms structure containing current video status info is returned.

```
struct SLivePrms
{
  int   StructSize; // set to sizeof(SLivePrms)
  int   NumFrames;
  int   BufIndex; // index to buffer currently most recent for display
  void* BufPtr;    // pointer to buffer currently most recent for display
  int   VideoStatus;
  int   ErrorCode;
};
```

NumFrames
The number of frames acquired.

BufIndex
The index of the sequence buffer most recently updated.

BufPtr
A pointer to the buffer most recently updated. If recording is in progress the VideoStatus member is HCP_REC_RECORDING and the pointer is to the sequence buffer. Corrections specified in the corrections settings AND the CorrType in the vip_fluoro_grabber_start call are applied already.

If grabbing is in progress but not recording then the VideoStatus member is HCP_REC_GRABBING and the pointer is to the grab buffer. NO corrections are applied in this case. If you wish to display the corrected image you should call vip_correct_image before displaying the buffer contents.

VideoStatus
The value is one of those in the enum HcpFluoroStatus defined in FluoroStructs.h.

ErrorCode
The value should normally be zero. A non-zero value means that an error has occurred. See error code in HcpErrors.h or HcpConstants.h as appropriate.

HCP_FLU_STATS_PRMS

With HCP_FLU_STATS_PRMS as the structure type, the structure pointer should point to a SSeqStats structure. This call should be after an acquisition is complete to determine information about the completed acquisition. This structure may only be referenced by vip_fluoro_get_prms().

```
struct SSeqStats
{
  int   StructSize; // set to sizeof(SSeqStats)
  int   SmplFrms;
  int   HookFrms;
  int   CaptFrms;
  int   HookOverrun;
  int   StartIdx;
  int   EndIdx;
  float CaptRate;
};
```

SmplFrms
This is the number of frames which potentially contributed to the sequence. If the sample rate is set to 2 then it is half the number of 'hooked' frames. This number reflects the total number of sampled frames not limited by the number of sequence buffers allocated.

HookFrms
This is the number of frames grabbed by the frame grabber and may be larger than the number of sampled frames if the sample rate is larger than 1. This number reflects the total number of hooked frames not limited by the number of sequence buffers allocated.

CaptFrms
This is the number of frames written to the sequence buffers. This number reflects the total number of captured frames not limited by the number of sequence buffers allocated.

HookOverrun
The routine that is called when a frame becomes available in a grab buffer can in principle be entered re-entrantly. If this happens the *HookOverrun* counter is incremented. This value should be zero. A non-zero value does not necessarily imply a problem but would indicate some likelihood of one.

StartIdx
As noted previously, the acquisition may be chosen to be free-running where more frames may be captured than sequence buffers exist. If this happens the earliest frames are overwritten and if the acquisition is stopped at an arbitrary point, the start frame in the sequence may be anywhere. *StartIdx* gives the index to the earliest frame (zero-based index) captured in the sequence buffers. ViVA uses this information to show and save frames in the correct order.

EndIdx
Gives the index to the last frame captured. Generally either N-1 where N is the total number of frames captured and N is less than or equal to the number of sequence buffers OR *StartIdx* - 1 when N is larger than the number of sequence buffers.

CaptRate
Overall rate at which frames were written to the sequence buffers (frames per second).

HCP_FLU_TIME_INIT

With HCP_FLU_TIME_INIT as the structure type, the structure pointer should point to a SSeqTimer structure. This call is not intended for non-Varian use as it returns a pointer to a Varian defined object. However, when called, it resets the internal timer used for timing info that is written to the double arrays reference by the SAcqPrms structure. (See description of vip_fluoro_grabber_start() above.) If no call is made, the zero for timing info is when the link opened.

```
struct SSeqTimer
{
  int   StructSize; // set to sizeof(SSeqTimer)
  void* SeqTimerPtr; // pointer to a Varian defined object
};
```

HCP_FLU_INDEX_RANGE

With HCP_FLU_INDEX_RANGE as the structure type, the structure pointer should point to a SIndexRange structure. Where supported a number of recently acquired images are stored on the receptor. This call allows discovery of the range of available indices referencing the images. A call may be made to vip_get_image to retrieve on of the stored images. See Table 3.1 for more info.)

4.4 MULTI-SEGMENT RECORDING

NOTE: Multi-segment recording provides significant flexibility, and consequently more care is demanded of the user application controlling acquisitions.

- As of build 26, it is permissible to record more than one segment of frames for only one grabber_start call.

- An additional parameter in record_start allows the buffer index for the first frame of the segment to be specified. *Note that the increased flexibility allows the user to overwrite previously acquired frames. No warning or error is generated.*

- If StopAfterN is set to a non-zero value then the TOTAL number of frames captured will be StopAfterN. e.g. Suppose you set StopAfterN to 5 through a call to set_prms or as a parameter in the first record_start. Recording stops after 5 frames are captured. Suppose next you want to capture 10 more frames. The second call to record_start MUST set the stopAfterN=15. However, open-loop - start/stop with StopAfterN=0 - is OK still, but the original value of StopAfterN for the first segment must be zero and maintained at zero for all subsequent calls.

- Use of multi-segment recording requires increased care by the user since a second call to record_start involves increased opportunity for errors to be generated when the second record-start call is made. Also buffers may be overwritten. Errors may be generated:
- --a) When the first record_start call is made, if the number of requested frames exceeds the number of buffers available, the VCP will attempt to re-allocate buffers which could result in a memory allocation error. (This possibility is not new.)
- --b) When the second or subsequent record_start calls are made, if the number of 'effective' (see more below) frames exceeds the number of buffers available, no re-allocation is attempted, and an error is generated.
- --c) If the specified buffer index is higher than the maximum available, an error is generated.

- If the second (new) parameter - 'startFromBufIdx' - in record_start is negative (default), the number of 'effective' frames is equal to StopAfterN, and the multiple segments fill the buffers contiguously.

- If startFromBufIdx is not negative, the number of 'effective' frames may be greater than StopAfterN by the number of buffers in any implied gaps between segments. Or it may be decreased from StopAfterN by the number of buffers in any implied overlaps of segments.

- In all cases the acquisition stops when the total number of acquired frames is StopAfterN irrespective of any gaps or overlaps in segment disposition.

5. ERROR CODES AND CONSTANTS

This section discusses the various constants and error codes used and returned by the PaxScan imaging software.

5.1 ERROR CODES

All function return values are of type *int* and indicate the success/failure of the function call. A non-zero return value means an error has occurred. A return value of zero or **HCP_NO_ERR** or **HCP_NO_ERR** implies that the function execution has been successful.

Error codes have been consolidated in the file HcpErrors.h (or HcpConstants.h and dependent files). All error codes used in previous receptors are supported though some have been redefined so that currently all error codes are in the range 0-128 (though this may change). ViVA handles old error code values as well as new ones and an example of a routine to do this is provided (commented out) at the end of HcpErrors.h. Error strings in the form of an array are given in the file HcpErrors.h. When supported vip_query_error will provide the string and additional error information.

The following is a list of VirtualCP error codes, each with a brief description of the error and an indication of possible recovery actions. In general, system installation or configuration errors require operator intervention and should be reported. Internal VirtualCp errors should be logged

and reported. Runtime errors during acquisition or calibration may be handled by trying the operation again. If a simple retry does not help, closing the link and opening it again should be sufficient to restore the system.

0 HCP_NO_ERR
Successful return for API call.

2 HCP_STATE_ERR
Non-fatal error, normally the result of thread interactions.
Recovery: retry API call a few (2 or 3, perhaps 5) times – if failure continues, cancel operation (for example by calling vip_reset_state) and start again. If the operation is not successful, close and open link. If the error persists, this may indicate an internal logic error.

3 HCP_NOT_OPEN
Application error: application made an API call that requires an open link, but the link is not open.
Recovery: try API call again with link open.

4 HCP_DATA_ERR
Miscellaneous internal error in handling receptor data buffers. May also be returned by vip_query_prog_info if the diagnostic data is not recognized, or by analog offset calibration if the receptor is not recognized.
Recovery: try the API call again.

5 HCP_NOT_IMPL_ERR
Application error: application made an API call that is not supported.
Recovery: no improvement possible.

6 HCP_OTHER_ERR
Installation/configuration error. Returned by vip_enable_sw_handshaking when the HcpConfig.ini file does not contain the IoDll=HcpIoRad setting. In that case, hardware handshaking acquisition will not be possible. Otherwise, error may be ignored.

8 HCP_NO_DATA_ERR
Application attempted to fetch data before it is ready.
Recovery: try the API call again after a short delay.

12 HCP_BAD_FILE_PATH
System installation error: specified imager directory not found.

15 HCP_FILE_WRITE_ERR
System installation error: imager directory and mode subdirectories need to be writeable, and currently logged-in account.

18 HCP_MEMALLOC_ERR
Runtime error: the VirtualCp was not able to allocate system memory that it needed for an operation. This may indicate a system memory leak.
Recovery: close link and open again.

19 HCP_NOVIDEO_ERR
Runtime error: indicates that the video acquisition system has not been initialized: returned by the fluoroscopy API calls.

20 HCP_NOTREADY_ERR
On check link, indicates that receptor not stabilized. If persists, may indicate that the receptor needs analog offset calibration.
Recovery: try API call again.

22 HCP_INIT_ERR
System installation error: the receptor subdirectory does not contain a needed mode subdirectory and the VirtualCp is unable to create a new subdirectory (the main directory is write-protected, or the current user does not have the necessary permission).

29 HCP_BAD_POINTER
Internal VirtualCp software error – report.

30 HCP_TIMEOUT
After the receptor is activated for handware handshaking acquisition, if nothing happens after a long time, the acquisition will be timed out. Recovery: restart the acquisition.

34 HCP_FUNC_ADDR
Internal VirtualCp software error – report.

35 HCP_ABORT_OPERATION
Not an error – indicates that a calibration was successfully cancelled by operator request.

47 HCP_OFST_ERR
System installation error: offset calibration file missing: repair or recalibrate.

48 HCP_GAIN_ERR
System installation error: gain calibration file missing or invalid: repair or recalibrate.

49 HCP_DFCT_ERR – calibration file missing: repair
System installation error: defect map missing or invalid. Repair.

52 HCP_IMG_DIM_ERR
System configuration error.

54 HCP_UNHANDLED_EXCEP
Internal VirtualCp software error – report.
Recovery: close link and open again.

57 HCP_TOO_FEW_MODES
System installation error: invalid configuration file.

58 HCP_MAX_ITER
Analog offset calibration failed: reached the maximum number of iterations without converging to the required target pixel values.
Recovery: try analog offset calibration again.
.

59 HCP_DIAG_DATA_ERR
Indicates a problem with the receptor: images do not contain diagnostic status information.

60 HCP_DIAG_SEQ_ERR

Link problem: an image frames are identified with sequence numbers. This error is returned by vip_query_prog_info when an out-of-sequence frame is detected. This indicates that a frame has been dropped.
Recovery: stop acquisition and start again.

61 HCP_DIAG_STATE_ERR
Link problem. The 4343R and 4336R require a Gigabit Ethernet connection. This error can occur if the link comes up in 100 Mbit mode (perhaps because of a defective cable). Errors can also occur if the receptor power is lost.
Recovery: stop acquisition and start again.

66 HCP_NO_GAIN_CAL_ERR
Flat field images do not have enough dose.
Recovery: notify operator to increase dose and retry the calibration procedure.

68 HCP_DEV_AMBIG
System installation error: HcpConfig.ini file error does not specify the receptor MAC address.

69 HCP_MAC_NOT_FOUND
Receptor not found on link.
Recovery: retry open-link when receptor connected.

70 HCP_MULTI_REC
Multiple receptors do not specify the same RP and FG modules. The combination of receptors is not supported.

71 HCP_RESELECT_FAIL
Problem opening link.
Recovery: close all links and try opening again.

72 HCP_RECID_CONFLICT
Problem opening link: multiple receptors with same ID.

73 HCP_CAL_ERROR
Runtime error: the calibration routine is not able to keep up with the data rate of the receptor. This is mainly for fluoroscopy applications and is not expected for radiographic receptors.
Recovery: try calibration operation again.

78 HCP_FRAME_SIZE_WARN
Only possible if bad image passthrough is turned on to warn that the image size is incorrect.

79 HCP_CALACQ_WARN
Warning that requested number of offset frames is greater than normal. If the request is valid, ignore this warning.

80 HCP_NULL_FUNC
Internal VirtualCp software error – report.
Recovery: close link and open again.

81 HCP_NO_SUBMOD_ERR
The VirtCp.dll was not able to load the lower-level support DLL. This may be caused by a configuration problem, or may indicate that the wrong version of the VirtualCp has been installed.

Check installation.

82 HCP_NO_MODE_SEL
Internal error: no mode is currently selected.

83 HCP_PLEORA_ERR
Frame grabber problem during acquisition.
Recovery: try acquisition again.

84 HCP_FG_INIT_ERR
System configuration problem.

85 HCP_GRAB_ERR
Frame grabber problem during acquisition.
Recovery: try acquisition again.

86 HCP_STARTGRAB_ERR
Frame grabber problem during acquisition.
Recovery: try acquisition again.

87 HCP_STOPGRAB_ERR
Frame grabber problem during acquisition.
Recovery: try acquisition again.

88 HCP_STARTREC_ERR
Frame grabber problem during acquisition.
Recovery: try acquisition again.

89 HCP_STOPREC_ERR
Frame grabber problem during acquisition.
Recovery: try acquisition again.

90 HCP_NO_CONNECT_ERR
Problem initializing Pleora device.
Recovery: close link and open again.

91 HCP_DATA_TRANS_ERR
Problem sending command to receptor – retry operation.
Recovery: close link and open again.

92 HCP_NO_EVENT_ERR
Problem during acquisition.
Recovery: try acquisition again.

93 HCP_FRM_GRAB_ERR
Problem during acquisition.
Recovery: try acquisition again.

94 HCP_NOT_GRAB_ERR
Problem during acquisition.
Recovery: try acquisition again.

95 HCP_BAD_PARAM
Application error: application passed an invalid parameter value.

96 HCP_ZERO_FRM
Problem during acquisition.
Recovery: try acquisition again.

97 HCP_RES_ALLOC_ERR
System error – VirtualCp was unable to allocate Windows resource needed for operation.
Recovery: close link and open again.

98 HCP_BAD_IMG_DIM
System installation/configuration issue: configuration file does not contain valid dimensions for the receptor or mode.

100 HCP_REC_NOT_SUPP
System installation or configuration error: the loaded VirtualCp version does not support the receptor type specified by the HcpConfig.ini file or the receptor configuration file. The wrong version of VirtualCp may have been installed, or there is a problem with either the HcpConfig.ini file or receptor configuration file (either the wrong file was installed, or the file has been corrupted).

101 HCP_REC_CNFG_ERR
Receptor configuration file is corrupted. May be a system installation error.

102 HCP_SHORT_BUF_ERR
Returns by vip_get_dll_version when application has passed a buffer that is not long enough to receive the version information.

103 HCP_RES_ALREADY_ALLOC
Runtime error, indicates that a resource (typically a thread) has already been created for the requested operation. Best recovery may require closing and opening the link.

104 HCP_BAD_REQ_ERR
Indicates an internal error during calibration or acquisition.
Recovery: try operation again. If that fails, close link and open again.

105 HCP_REC_NOT_READY
Returned by vip_check_link if the receptor has output a dark image, but the image quality indicates that the receptor is not yet stable.
Recovery: try operation again. If that consistently fails, the receptor may need a new analog offset calibration.

106 HCP_PLR_CONNECT
Receptor not found during open link - retry when receptor connected.

107 HCP_INTERNAL_LOGIC_ERROR
Internal VirtualCp software error – report.
Recovery: close link and open again.

108 HCP_PLR_SERIAL

Runtime problem sending a command to the receptor. Indicates potential problem with communication to the receptor serial port, or receiving serial information, as in the case with retirieving diagnostics data over serial.
Recovery: close link and open again, if error conditions persist, power cycle receptor and re-open link.

110 HCP_DRV_VERS_ERR
System installation problem: the wrong version of a driver has been installed (such as the Pleora frame grabber driver).

111 HCP_STRCT_ERR
Application passed a pointer to a structure in which the StructSize member is not set to the correct size. This may occur if the application was built to a version of the VirtualCp that is not compatible with the installed version.

112 HCP_DLL_VERS_ERR
Unexpected VirtualCp DLLS were loaded. This may occur if multiple versions of the VirtualCp DLLs are present on the system, or if the VirtualCp was not properly installed.

113 HCP_FXD_RATE_ERR
System configuration error.

114 HCP_TEMP_OVER
Indicates that the receptor has overheated and shut down. Wait until receptor has cooled before proceeding.

115 HCP_LINK_SPEED_ERR
Indicates that link speed of network interface card which the receptor connects to may not be up to speed. This is a hardware problem which could be caused by bad Ethernet cable connection, broken wires in the cable or connector, etc. Image transfer through Ethernet link may fail. If replacing Ethernet cable does not fix the problem, contact Varian support for further assistant.

128 HCP_NO_IMAGE_ERR
Returned by vip_check_link if no image is captured. May indicate that the link is broken. Returned by vip_get_image if the requested image type is not available.
Recovery: try operation again. If that fails, close link and open again.

129 HCP_WIFI_IOBOX_ERR
Returned by vip_open_receptor_link if I/O box cannot be reached. May indicate that the connection to I/O box is broken.
Recovery: check cable, make sure I/O box is on and fully booted. Then try operation again.

130 HCP_WIFI_NO_ASSOCIATION
Returned by vip_open_receptor_link if the wireless receptor cannot be associated with the I/O box.
Recovery: Check the correct receptor is selected and try operation again.

131 HCP_WIFI_NO_WAKEUP
Returned if the receptor is failed to wake up from asleep.
Recovery: try operation again.

132 HCP_WIFI_RECEPTOR_ERR
Returned if receptor cannot be reached.
Recovery: try operation again.

133 HCP_WIFI_BATTERY_LOW
Returned if battery level is bellow threshold.
Recovery: replace with a fully charged battery and try operation again

134 HCP_WIFI_LINK_QUALITY_LOW
Returned if wireless link quality is too low. May indicate that there is too much interference or the receptor is too far away.
Recovery: move the receptor closer to I/O box and try operation again. If that fails, close link and open again with a tether cable.

135-142 HCP_ERROR_TEMPERATURE
Returned when one or more of the temperatures sensors in the receptor has reached, or exceeded temperature threshold established by Varian
Recover, close link, power down and wait until receptor has cooled before proceeding.

143-174 HCP_ERROR_VOLTAGE
Returned when when one or more of the voltage sensors in the receptor has reached, or exceeded voltage threshold established by Varian.
Recover, close link, power down and wait until receptor has cooled before proceeding.

175 HCP_DEVICE_BUSY
Returned by vip_query_prog_info (HCP_U_QPIDIAGDATA_POLL) if the receptor is currently busy.

176 HCP_REC_CANCEL_FAILED
Returned by `vip_io_enable(HS_CANCEL).` If the cancel occurs after the Expose_Request, the function returns HCP_REC_CANCEL_FAILED, indicating image acquisition is going through. If the cancel occurs prior to Expose_Request, the function returns 0 (SUCCEEDED).

177 HCP_DEFECT_FILE_UUID_MISMATCH
Returned by `vip_select_mode(). This error message is retuned if the defect correction data file's UUID for that mode does not match the UUID of the receptor`. Internally, VCP changes the mode to the requested mod so that the user can call gain calibration to re-generate the defect map files if requested. In this case you must take the appropriate action by asking the operator to do a Gain Calibration or install the correct defect map files.

5.2 NON-FATAL ERROR CODES USED TO SHOW CORRECTION CAPABILITY

Certain calls as discussed below return non-fatal error codes that can be used to determine what corrections are available. The return code is based on the requested corrections (as determined from receptor configuration file with updates if any from calls to vip_set_correction_settings) AND the available corrections:
HCP_NO_ERR – all requested corrections are available
HCP_OFST_ERR – no corrections are available
HCP_GAIN_ERR – of requested corrections only offset is available
HCP_DFCT_ERR - of requested corrections only offset and gain are available

Note if you want to get information as to available corrections only, call vip_set_correction_settings with Ofst, Gain and Dfct all set to 1 (error return will be indicative of currently available corrections) then call again to restore corrections as you actually want them if necessary.

vip_open_link(..)
Mode 0 is always selected.
The corrections in the receptor configuration are applied.
The return value indicates what corrections are available for mode 0 (as ANDed with the corrections requested). For example if all corrections are turned off the return will be HCP_NO_ERR irrespective of corrections available. If all corrections are turned on and only offset corrections are available the return value will be HCP_GAIN_ERR.

vip_select_mode(N)
Mode N is selected.
Corrections remain the same as already set.
The return value indicates what corrections are available for mode N as described above.

vip_set_correction_settings(..)
The requested corrections are set.
The return value indicates what corrections are available for the currently selected mode as described above.

vip_get_correction_settings(..)
The currently requested corrections are returned.
The return value indicates what corrections are available for the currently selected mode as described above. This call may be used to verify that corrections will succeed prior to the start of an acquisition or safer yet call vip_correct_image() with a dummy buffer.

vip_correct_image(..)
If any of the requested corrections are not available, an error is generated and the return value indicates what corrections are available. No corrections are applied. Note that one or more calls to vip_correct_image() are implicit in a call to vip_get_image() in rad modes or a fluoro acquisition where real-time corrections are turned on. Explicit calls to vip_correct_image() are not necessary.

ViVA
ViVA uses these returns by warning when corrections settings and availability are incompatible. If an attempt is made to start an acquisition while that situation exists it generates an error.

5.3 System Version Number Types

Various version types may be interrogated by the call vip_get_system_version_numbers(). Some version types such as boards and firmware specific to the Command Processor are not relevant to all receptors and will return VIP_NOT_IMPL_ERR.

Name	Value	Description
VIP_MOTHERBOARD_VER	0	Version number of the Command Processor motherboard.
VIP_SYS_SW_VER	1	Version number of the system software.
VIP_GLOBAL_CTRL_VER	2	Version number of the global control board.
VIP_GLOBAL_CTRL_FW_VER	3	Version number of the global control board firmware.
VIP_RECEPTOR_VER	4	Version number of the receptor.
VIP_RECEPTOR_FW_VER	5	Version number of the Receptor firmware.
VIP_IPS_VER	6	Version number of the IPS board.
VIP_VIDEO_OUT_VER	7	Version number of the 8-bit video board.
VIP_VIDEO_OUT_FW_VER	8	Version number of the 8-bit video board firmware.

5.4 Image Types

The calls vip_get_image() and vip_put_image() specify an image_type. This should be one of the values in the following table. Note that vip_put_image cannot be used in conjunction with VIP_CURRENT_IMAGE or VIP_TEST_IMAGE for non-Command Processor systems.

Name	Value	Description
VIP_CURRENT_IMAGE_1	0xF40000101	This image resides in the accumulation buffer of the image processing system and can be summarized in the following formula: CURRENT_IMAGE = [(raw image data) – post

Name	Value	Description
		OFFSET_IMAGE]/ GAIN_IMAGE; this is used for main shot acqusitions.
VIP_CURRENT_IMAGE_0	0xF4000010	This image resides in the accumulation buffer of the image processing system and can be summarized in the following formula: CURRENT_IMAGE = [(raw image data) – stored OFFSET_IMAGE]/ GAIN_IMAGE; this parameter is used for prepulse acquisition.
VIP_OFFSET_IMAGE	1	This image is the offset correction data obtained during an offset calibration.
VIP_GAIN_IMAGE	2	This image is the gain correction data obtained during a gain calibration. It is offset-corrected, not a raw flat field image.
VIP_BASE_DEFECT_IMAGE	3	This image is the base defect map data which are normally set up in manufacturing/final test. This map is not affected by calibration. A non-zero pixel value indicates the presence of a defect.
VIP_AUX_DEFECT_IMAGE	4	This image is the auxiliary defect map which is recalculated during every gain calibration. A non-zero pixel value indicates the presence of a defect.
VIP_TEST_IMAGE	5	A test image is generated by the Command Processor.
VIP_RECEPTOR_TEST_IMAGE	6	Initiates the generation of a test image by the receptor.
VIP_RECEPTOR_TEST_IMAGE_OFF	7	Turns off the test image by the receptor.
VIP_ANALOG_OFFSET_IMAGE	8	Special calibration feature
VIP_PREVIEW_IMAGE	9	Radiographic image corrected with a stored offset file instead of captured offset data
VIP_RAD_OFFSET_I	10	The captured offset data that

Name	Value	Description
MAGE		was used to correct the final (SCT or RCT) radiographic image.

EXTENDED IMAGE TYPES FOR RADIOGRAPHIC RECEPTORS (L.04 ONLY):

Name	Value	Description
VIP_CURRENT_IMG_2	0xF4000012	Current image corrected using 1 post offset frame (valid only for SCT modes: same as VIP_CURRENT_IMAGE); this is not used for 3024M
VIP_CURRENT_IMG_RAW	0xF400001F	Exposed frame without corrections (same as VIP_CURRENT_IMAGE with corrections turned off).
VIP_OFFSET_IMG_1	0xF4000021	First offset frame after the exposed frame.
VIP_OFFSET_IMG_0	0x01	Stored offset frame.
VIP_OFFSET_IMG_2	0xF4000022	Second offset frame after the exposed frame (valid for SCT modes only)
VIP_OFFSET_IMG_AV	0xF400002F	Average of all post-acquired offset frames. For RCT mode, the same as VIP_OFFSET_IMG_1.
VIP_RESEND_TARGET_FLAG	0xF5000000	Retransmit the exposed and offset frames from the receptor, and perform correction using these offset frames.

5.5 SOFTWARE HANDSHAKING CONSTANTS

These constants are used in conjunction with the vip_sw_handshaking() call, specifying the *signal_type*.

Name	Value	Description
VIP_SW_PREPARE	0	TRUE signals the receptor to prepare for the acquisition of an X-ray image. The FALSE signal, which indicates no further acquisition is expected, should not be sent until the acquisition is finished.
VIP_SW_VALID_XRAYS	1	TRUE signals that the X-ray generator is active (either the beam is on or the generator is waiting for a signal). FALSE signals the interface that the acquisition is finished.

5.6 ACQUISITION CONSTANTS

These specify whether the mode type fluoro or rad. This value is returned by vip_get_mode_info() – *AcqType*. Safe handling of this value requires that it be bitwise AND'd with VIP_ACQ_MASK = 0xFF since some receptors provide additional information in higher bits.

Name	Value	Description
VIP_ACQ_TYPE_CONTINUOUS	0	A fluoro mode. Images are acquired continuously while PREPARE is TRUE and additionally saved to sequence buffers when VALID_XRAY is TRUE.
VIP_ACQ_TYPE_ACCUMULATION	1	A rad mode. Images are acquired under software or hardware handshaking control. Typically 1 image is acquired and saved though if the number of accumulation frames is set to more than 1 then the sum is formed.

I/O CONTROL: ENABLE CODES (RAD MODES)

Name	Value	Description
HS_STANDBY	0	Disable receptor exposure control logic. Initial state of system.
HS_ACTIVE	1	Enable receptor exposure control logic.
HS_CANCEL	4	Option for 4336R: disables acquisition

5.7 I/O SUPPORT: I/O CONTROL MACHINE STATES (IOSTATE/EXPSTATE) (RAD MODES)

NOTE: these codes are obsolete, and are not applicable to Rad panels 4343R and 4336R. These only apply to L.01/L.03 when an external I/O box is used. for hardware handshaking.

Name	Value	Description
IO_STANDBY	0	Awaiting hand switch input
IO_PREP	1	Prep pressed, waiting for Ready
IO_READY	2	Ready pressed, waiting for imager to be ready
IO_ACQ	3	Image acquisition in progress
IO_FETCH	4	Image being transferred from imager to host
IO_DONE	5	Image acquisition is complete
IO_ABORT	6	Prep or Ready released before start of acquisition (or timeout on receptor readout).
IO_INIT	7	Initial state after vip_open_link() call
IO_INIT_ERROR	8	Prep, Ready or A.L.E. asserted during initialization (or receptor not working).

Name	Value	Description
EXP_STANDBY	0	Awaiting hand switch input
EXP_AWAITING_PERMISSION	1	Prep or Ready state, user app needs to respond
EXP_PERMITTED	2	User app has granted

		permission to expose
EXP_REQUESTED	3	Expose_request output asserted
EXP_CONFIRMED	4	A.L.E. is high
EXP_TIMED_OUT	5	Error condition
EXP_COMPLETED	6	Falling edge of A.L.E. or cycle time done

5.8 RECEPTOR STATUS/DIAGNOSTIC DATA

New receptors supported by L04 (initially 4343R and 4336R) embed status information in a few pixels at the edge of each image. This information includes:

1. Receptor ID: model type, firmware version, board ID number.
2. Acquisition status: frame sequence number and exposure status.
3. Diagnostic data: internal temperatures and voltages.

Whenever the VirtualCP captures image frames from the receptor (whether a successful vip_check_link, acquisition or calibration), this information is stored. It may be retrieved through the **vip_query_prog_info(..)** API call by using special request codes. After an acquisition, the information is extracted from the exposed (X-ray image) frame, not the offset frames that follow. If the request is made while an acquisition or calibration is in progress, the information is extracted from the most recently captured frame.

For monitoring temperatures or voltages while the system is idle, a periodic call to vip_check_link (or offset calibration) can be used to refresh the diagnostic data. As an alternative, the option bit HCP_U_QPI_CRNT_DIAG_DATA can be ORed with the request code to force a hidden check-link operation as part of the vip_query_prog_info call.

Receptor information HCP_U_QPIRCPT

Calling sequence:

```
SQueryProgInfo uqpi;
memset(&uqpi, 0, sizeof(uqpi));
uqpi.qpircpt.StructSize = sizeof(SQueryProgInfoRcpt);

uType = HCP_U_QPIRCPT;

result = vip_query_prog_info(uType, &uqpi);
```

On return, member structure **uqpi.qpircpt** is set up as follows:

```
struct SQueryProgInfoRcpt
        {
                int             StructSize;
                int             PanelType;
                int             FwVersion;
                WORD        BoardSNbr[3];
                WORD        ReservedWd;
                int         ReservedRcpt2;
                int         ReservedRcpt1;
};
```

PanelType = 3 for 4343R, 2 for 4336R
FwVersion = (Major Revision * 256) + (Minor build number)
BoardSNbr[0..2] = 48-bit number, unique for each unit

Frame information HCP_U_QPIFRAME

Calling sequence:
SQueryProgInfo uqpi;
memset(&uqpi, 0, sizeof(uqpi));
uqpi.qpiframe.StructSize = sizeof(SQueryProgInfoFrame);

uType = HCP_U_QPIFRAME;

result = vip_query_prog_info(uType, &uqpi);

On return, member structure **uqpi.qpiframe** is set up as follows:

```
struct SQueryProgInfoFrame // uType = HCP_U_QPIFRAME
        {
                int             StructSize;
                int             RcptFrameId;
                int                     Exposed;
                unsigned long   AcqTime;
                int             FrameType;
                int             Cancellation;
                int             Prepare;
                int             ReadyForExposure;
        };
```

RcptFrameId = 0..255
AcqTime = millisecond tick count from system clock (zero point not specified)
Exposed = raw exposure bits from frame
FrameType = enum representing decoding of bits in Exposed, as shown below

Frame Description	Exposed	FrameType
Idle frame, exposure enabled	0x0000	RAD_FRAME_IDLE =0
Offset frame	0x2000	RAD_FRAME_OFFSET =1
Exposed frame	0x8000	RAD_FRAME_EXPOSED =2
Idle frame, exposure disabled	0x4000	RAD_FRAME_DISABLED =3
Offset frame (alternate state)	0x6000	RAD_FRAME_OFSDIS =4
Nonstandard frame	other	RAD_FRAME_OTHER =5

Cancellation = special status returned by 4336R.
Prepare = special status returned by 4336R.
ReadyForExposure = receptor has confirmed that it is ready for exposure.

Temperature information HCP_U_QPITEMPS

Calling sequence:
SQueryProgInfo uqpi;
memset(&uqpi, 0, sizeof(uqpi));
uqpi.qpitemps.StructSize = sizeof(SQueryProgInfoTemps);

uType = HCP_U_QPITEMPS;
If the system is not currently in the HS_ACTIVE state, current data may be requested by modifying the uType request:
uType = HCP_U_QPITEMPS | HCP_U_QPI_CRNT_DIAG_DATA;

result = vip_query_prog_info(uType, &uqpi);

On return, member structure **uqpi.qpitemps** is set up as follows:

```
struct SQueryProgInfoTemps
        {
                int             StructSize;
                int             NumSensors;
                float           Celsius[16];
};
```

NumSensors = number of entries filled-in
Celsius[0..NumSensors-1] = measured temperatures (degress Celsius)

Temperature Range

Prior to each acquisition, self-check is done and will directly poll the temperature data from the receptor and compare the data to the following table and will return error if any vlue is over the maximum allowed value.

Temperature Sensor	Max Value (in ^{0}C)	Source (Reference)
Temperature 1	72	Digital Board (Components)
Temperature 2	51	Digital Board (Glass)
Temperature 3	72	Digital Board (Components)
Temperature 4	72	Digital Board (Components)
Temperature 5	72	Snap on board (Components)
Temperature 6	72	Snap on board (Components)
Temperature 7	Not USED	N/A
Temperature 8	Not USED	N/A

Voltage information HCP_U_QPIVOLTS

Calling sequence:
SQueryProgInfo uqpi;
memset(&uqpi, 0, sizeof(uqpi));
uqpi.qpivolts.StructSize=sizeof(SQueryProgInfoVolts);

uType=HCP_U_QPIVOLTS,
If the system is not currently in the HS_ACTIVE state, current data may be requested by modifying the uType request:
uType = HCP_U_QPIVOLTS | HCP_U_QPI_CRNT_DIAG_DATA;

result = vip_query_prog_info(uType, &uqpi);

On return, member structure **uqpi.qpivolts** is set up as follows:

```
struct SQueryProgInfoVolts
        {
                int                     StructSize;
                int             NumSensors;
                float            Volts[16];
        };
```

NumSensors = number of entries filled-in
Volts[0..NumSensors-1] = measured voltages (in volts).

Raw information from cache data
HCP_U_QPIDIAGDATA

This function will retrieve the cashed diagnostics data saved in memory from the last acquired image; it will fail if there has been no previous image acquisitions or the the data is too old.

Calling sequence:
SQueryProgInfo uqpi;
memset(&uqpi, 0, sizeof(uqpi));
uqpi.qpidiag.StructSize=sizeof(SDiagData);

uType=HCP_U_QPIDIAGDATA,

result = vip_query_prog_info(uType, &uqpi);

On return, member structure **uqpi.qpidiag** is set up as follows:

```
struct SDiagData
        {
                int             StructSize;
                unsigned long   AcqTime;
                WORD            Protocol;
                WORD            PanelType;
                WORD            FwVersion;
                WORD            Exposed;
```

```
        WORD        RcptFrameId;
        WORD        BoardSNbr[3];
        WORD        TemperatureData[8];
        WORD        VoltageData[16];
        WORD        Reserved[28];
};
```

Polling raw Information from device
HCP_U_QPIDIAGDATA_POLL

This function will retrieve the diagnostics data directly from the device;

Calling sequence:

```
SQueryProgInfo uqpi;
memset(&uqpi, 0, sizeof(uqpi));
uqpi.qpidiag.StructSize=sizeof(SDiagData);

uType=HCP_U_QPIDIAGDATA_POLL,

result = vip_query_prog_info(uType, &uqpi);
```

On return, member structure **uqpi.qpidiag** is set up as follows:

```
struct SDiagData
        {
                int         StructSize;
                unsigned long   AcqTime;
                WORD        Protocol;
                WORD        PanelType;
                WORD        FwVersion;
                WORD        Exposed;
                WORD        RcptFrameId;
                WORD        BoardSNbr[3];
                WORD        TemperatureData[8];
                WORD        VoltageData[16];
                WORD        Reserved[28];
};
```

6. CALIBRATION AND CONFIGURATION FILES

The VirtCp.dll reads in calibration files so that it can correct raw images acquired from the X-ray imager panel. The files are read when vip_open_receptor_link() is called: the *RecDirPath* member of the SOpenReceptorLink structure is expected to be a path name to the calibration tree. The recommended convention is to name this tree C:\IMAGERs*serialnbr*. For example, if the imager serial number is 1234-56, this path is C:\IMAGERs\1234-56.

Under this directory are subdirectories holding the calibration files for separate modes. Each mode represents a different set of operating parameters as determined from vip_get_mode_info(). For many applications, the panel is always in radiography mode, and there is only one X-ray exposure per image, so there only needs to be one mode.

The mode subdirectory names start with a two digit mode number and an underscore. The first mode is mode 0, and its directory name starts with "00_", followed by a name base on the *ModeDescription* member of the SModeInfo structure. Characters not allowed in directory names together with spaces are stripped out and the full path to the mode directory is given in the character string *DirReadyModeDescription* member of the SModeInfo structure.

Using these choices (serial number "*serialnbr* " and mode 0 name "*modename*"), the directory tree looks something like:

C:\IMAGERs\ *serialnbr* \ HcpConfig.ini
C:\IMAGERs\ *serialnbr* \ RecepConfig.dat
C:\IMAGERs\ *serialnbr* \ vivacy.xml

C:\IMAGERs\ *serialnbr*\00_ *ModeDescription*\ofst_img.viv
C:\IMAGERs\ *serialnbr*\00_ *ModeDescription*\gain_img.viv
C:\IMAGERs\ *serialnbr*\00_*ModeDescription*\defect_map.bin
For DGS modes there are two sets of calibration files for high and low gain states. Also during the extended gain calibration a sub-directory - DYN_cal_data - is created for storage of the acquired data.

6.1 CALIBRATION FILES

The original defect_map.dat file, which is a bit image that identifies the defective pixels at the time of manufacture. This file is updated each time the panel is recalibrated, so that pixels that fail after manufacture will be removed from the display. The defect_map file assigns 2 bits to each pixel: one for original (factory) defects and one for defects that are discovered later.

The ofts_img.viv file is a standard VIVA format file that contains the average uncorrected data for a series (usually 8) of dark fields images (no X-rays on the panel).

The gain_img.viv file is a standard VIVA format file that contains the average offset-corrected data for a series of flat field images (X-ray directly on panel, but no subject).

To support 16-bit imager, the gain image file has been extended to contain 24-bit of gain data for each pixel. The additional 8-bit are appended to the gain_img.viv file. Therefore, the size of

the gain image file is determined by the following: [(viv_header + (x_size * y_Size * 2) + (x_size * y_size)].

Note 1: Only used if 16 bit imager is selected.

6.2 RECEPTOR CONFIGURATION FILE

The receptor configuration file contains the receptor configuration as generated by the configure.exe program. This file is parsed and downloaded to the receptor when the link opens. Originally called 'RecepConfig.dat', the actual file to be used must be listed in the HcpConfig.ini file (see below). The virtual CP will use the first file in the section '[HcpConfigFiles]' that has the .dat extension. The name may contain wildcards in which case there must only be one file in the receptor directory that matches the name template; otherwise the link open will fail. Do not replace the .dat with a wildcard as the link open will fail.

If no valid receptor configuration entry is found in the ini file it will default the name to 'RecepConfig.dat'.

6.3 VIRTUAL CP CONFIGURATION FILE

The file HcpConfig.ini contains information as to the Virtual CP configuration. It specifies the sub-modules that the HcpRecCtrl.dll should look for, and is contained in a section [ReceptorControl]. It also specifies the names of additional files which are required for system configuration purposes. This [ConfigFiles] section is used by ViVA when a new receptor is manually added.

As of build 25, a 'FileRev' value may be present in the [VirtCp] section. Its absence will result in prior behavior. This setting was introduced to allow different versions of the Pleora configuration file – vivacy.xml – to be used. When FileRev is present and non-zero, the file name required is decorated. e.g.

[VirtCp]
FileRev=2

the Virtual CP looks for vivacy002.xml. This file should be the one supplied and used for version 2 receptors. In addition, correction file names are similarly decorated. e.g. an offset calibration produces the file ofst_img002.viv.

Certain additions may be made manually by the user to customize their application. These are documented below.

Generator Warmup Time

The options in this section are not supported by L.04 with the Rad panels 4343R and 4336R. These apply only when HcpIoAdu200.dll is used with the I/O box hardware handshaking interface.

The handswitch interface may be configured to require a minimum time interval from the PREP state to the READY state (to give the X-ray generator time to prepare). A separate section is required for this option. The value is set as follows:

[IoControl]
MinimumPrepMillisec=4000

The value 4000 in this example specifies that the handswitch must be held in the PREP position for at least 4 seconds before the READY position is recognized. If this option is not specified in the configuration file, the default value of 0 is used, so that no delay is required between PREP and READY.

The [IoControl]section also allows other I/O state machine timing parameters to be set: the maximum time to allow for fetching the image from the receptor, and the minimum times to remain in the IO_DONE state (acquisition complete) and IO_ABORT (handswitch released prior to X-ray generator firing) states. The default value for each is 2000, giving a 2 second delay:

[IoControl]
MaximumFetchMillisec=2000
MinimumDoneMillisec=2000
MinimumAbortMillisec=2000

The minimum times are provided so that I/O states are guaranteed to be seen by an application program that is polling slowly (perhaps once per second). These should not be set to less than 200 milliseconds.

The MinimumAbortMillisec time applies only in the case where an acquisition is aborted and the handswitch remains released. If the handswitch is operated again, the I/O state machine responds immediately, and may exit the IO_ABORT state before the programmed minimum time has elapsed.

Debug Mode

As described for the vip_set_debug() call (Table 2-1, #69), a debug mode may be set through calls to vip_set_debug() or when the vip_open_receptor_link() call is made. It may also be defaulted in the HcpConfig.ini file by adding an entry:

```
[VirtCp]
DebugMode=1
;use one of the numerical values as given in  HcpSundries.h or
;Table 2-1, #69 above.
```

This is equivalent to sending the value specified in the HcpConfig.ini in the open_receptor_link structure. If both specify a non-zero value, that in the open_receptor_link structure will take precedence. Subsequent calls to vip_set_debug have the effect stated above. You must send vip_set_debug(HCP_DBG_OFF) in order to get the text file written unless using the HCP_DBG_ON_FLSH mode.

StructSize Override

The StructSize member in most of the structs used in function call parameters is validated against the expected size. If required to accommodate code already developed which does not set this member properly this test can be overridden in some cases. If this option is required add an entry to the HcpConfig.ini file as below:

```
[VirtCp]
StructSizeOverride=TRUE
```

Analog Offset

A number of settings may be used to modify the analog offset calibration as described below an din section 3.1.

```
 [Calibration]
; Standard procedure (no global offset), AoProcedureType=0
; For PlanA, AoProcedureType=1
; For PlanB, AoProcedureType=2
AoProcedureType=1
; Used for PlanA only as described above
AoMedianFracScalerA=0.3
; DCDS target value
AoDcdsTarget=800
; # of cal frames to be acquired and averaged on each iteration
AoNumCalFrames=3
```

Check Link Frames Number

The number of frames acquired when vip_check_link is called is defaulted to 3 for fluoro modes but may be set to a different number if required (not applicable to rad modes).

[ReceptorControl]
CheckLinkFrames=2

6.4 PLEORA CONFIGURATION FILE

The file vivacy.xml or vivacy00N.xml is used by the Pleora software to initialize the frame grabber configuration.

7. CALIBRATION OF DGS MODES

*NOTE: For non-ConeBeam (standard single gain modes) use the same procedures as used previously – see **vip_sw_handshaking** for both fluoro and rad modes.*

NOTE: For DGS modes, it is required to have done (or have calibration data for) an extended gain calibration before an offset or regular gain calibration can be performed.

For DGS modes, new calibration procedures are introduced in L06.

Offset calibrations are unchanged and done autonomously as before using the ***vip_offset_cal*** call. In the case of DGS modes this call results in two sequential offset calibrations – one with regular DGS settings and one with forced low gain (FLG).

For gain and extended gain calibrations, a new flexible calibration interface is introduced in the L.05 VCP. The interface provides for flexibility in the way in which data are acquired and processed. However, current usage should conform to the examples in the sample code. Gain calibrations are performed by a sequence of calls.

7.1 GAIN CALIBRATION

For gain calibrations the following call sequence should be used:

7.1.1 Initialize

Initialize the calibration with ***vip_gain_cal_prepare***, entering the VCP into the calibration state. The *calType* parameter should be set to HCP_CALTYPE_GAIN – see the *vip_gain_cal_prepare* section of ***HcpSundries.h***.

7.1.2 Acquisition Segments

In general several ***vip_cal_control*** calls are issued, each requesting an acquisition segment of a specific type. These normally differ in x-ray exposure conditions. The appropriate x-ray exposure condition should be established before the call is made and held until after it completes.

The completion of each individual acquisition segment is indicated by 'Complete' being set in the ***vip_query_prog_info*** call.

An acquisition segment may be aborted without affecting prior segments by calling ***vip_reset_state***. This does not result in the complete reset of the calibration module. Data acquired in prior segments remain in memory where appropriate.

For DGS gain calibration 3 calls to vip_cal_control are required. These are to perform offset (dark-field) acquisitions (one call actually results in DGS and FLG acquired data), and separate calls for flat-field HI_GAIN and LO_GAIN calibration. The HI_GAIN acquisition is actually done in the DGS state, and it is important to set the x-ray level such that all pixels remain in the HI_GAIN state. For the LO_GAIN segment, the x-ray level may be left the same or increased to a level more appropriate to the low gain state. The FLG state is entered so gain switching is not dependent on the x-ray level here.

The following acquisition segments must be performed:

7.1.2.1 Darkfield

X-ray is set OFF. A call to vip_cal_control is made with CtrlType=HCP_CTRLTYPE_OFST, and AccMode= HCP_ACCMODE_SUM_AND_HOLD. Allowed AccModes are those in the second enum in the vip_cal_control section of ***HcpSundries.h***.

7.1.2.2 Flatfield – HI_GAIN

X-ray is set to appropriate level – high enough for satisfactory gain calibration but low enough to ensure all pixels remain in the HI_GAIN state.

A call to vip_cal_control is made with CtrlType=HCP_CTRLTYPE_GAIN_HI_G, and AccMode= HCP_ACCMODE_SUM_AND_HOLD.

7.1.2.3 Flatfield – LO_GAIN

X-ray is set to appropriate level – high enough for satisfactory gain calibration but low enough to ensure all pixels remain linear in the LO_GAIN state. Note that in this case the LO_GAIN state is forced by the selection of the auxiliary sub-mode.

A call to vip_cal_control is made with CtrlType=HCP_CTRLTYPE_GAIN_LO_G, and AccMode= HCP_ACCMODE_SUM_AND_HOLD.

7.1.3 Process Data

When all data required have been acquired, a call is made to vip_cal_control with AccMode = HCP_ACCMODE_END_AND_SEND. (CtrlType is ignored.)

7.1.4 End Calibration

A call to ***vip_cal_end*** is made to terminate the calibration and exit the calibration state. This call is normally generated automatically by the call to process data, but should be made to ensure the VCP state is set properly when the calibration is over.

This call may also be made at any point before the process data call to abort the calibration. No data would be stored in the case here if that were done; this is because the AccMode specified the data to be held at the end of each acquisition segment.

7.2 EXTENDED GAIN CALIBRATION

The completion of each individual acquisition segment is indicated by 'Complete' being set in the ***vip_query_prog_info*** call.

An acquisition segment may be aborted without affecting prior segments by calling ***vip_reset_state***. This does not result in the complete reset of the calibration module. Data acquired in prior segments remain in memory where appropriate.

Extended gain calibrations may only be done for DGS modes. For extended gain calibrations the following call sequence should be used:

7.2.1 Initialize

Initialize the calibration with ***vip_gain_cal_prepare***, entering the VCP into the calibration state. The *calType* parameter should be set to HCP_CALTYPE_EXTGAIN – see the *vip_gain_cal_prepare* section of ***HcpSundries.h***.

7.2.2 Acquisition Segments

For extended gain calibrations, either at lease 3 acqusition segments are required. A 4th is optional, and only necessary if recalibration of the gain ratio is intended. In extended gain calibrations, each acquisition segment is performed under specified x-ray exposure conditions, and for each 2 sets of data are obtained using DGS and FLG.

The following acquisition segments are required unless otherwise noted:

7.2.2.1 Darkfield
X-ray is set OFF. A call to vip_cal_control is made with CtrlType=HCP_CTRLTYPE_EXT0 (AccMode must be defaulted to zero and need not be set in any segement here).

7.2.2.2 Flatfield1
This acquisition is optional. Data are used to calculate a new value for the gain ratio. The x-ray level must be set such that all pixels remain in the HI_GAIN state. A call to vip_cal_control is made with CtrlType=HCP_CTRLTYPE_EXT1.

7.2.2.3 Flatfield2
The x-ray level must be set such that all pixels switch to LO_GAIN. A call to vip_cal_control is made with CtrlType=HCP_CTRLTYPE_EXT2.

7.2.2.3 Flatfield3
The x-ray level must be set such that all pixels switch to LO_GAIN and at a higher level (say ~2x) than for *Flatfield2*; the x-ray level must be such that pixel response remains linear. A call to vip_cal_control is made with CtrlType=HCP_CTRLTYPE_EXT3.

7.2.3 Process Data

When all data required have been acquired, a call is made to vip_cal_control with CtrlType = HCP_CTRLTYPE_EXTCAL. The GainRatio member returns the value of the gain ratio – a new value if *Flatfield1* was done or otherwise the old value.

7.2.4 End Calibration

A call to ***vip_cal_end*** is made to terminate the calibration and exit the calibration state. This call is normally generated automatically by the call to process data, but should be made to ensure the VCP state is set properly when the calibration is over.

This call may also be made at any point before the process data call to abort the calibration. In this case data are already stored for any completed acquisition segment in the mode sub-directory ‘DYN_cal_data’ but not used for calibration unless the process data call has been made.

7.2.5 Extended Gain Calibration Statistics

Extended gain calibration statistics are retrieved using a special construct of vip_get_cal_info. A new struct is defined and may be passed with this call by casting its address to SCalInfo* and setting a special ID value. For details see definitions in the ***vip_get_cal_info*** section of ***HcpSundries.h*** and the sample code.

VARIAN
medical systems

Document Signing Page

This document has been reviewed and electronically signed by the following people:

Workflow ID: 33535836, Doc Title: FP1284_L06_MammograpphyVirtualCpInterface.doc, Doc No:, Approver: Aleida Harrison (aharris0), Title: Regulatory Affairs Specialist, Date: Wednesday, Oct 02 2013 10:01 AM Pacific Daylight Time, Meaning: I acknowledge my electronic signature carries the same meaning as a handwritten signature

Workflow ID: 33535836, Doc Title: FP1284_L06_MammograpphyVirtualCpInterface.doc, Doc No:, Approver: Fereydoon Zoghi (fzoghi), Title: Software Lead Engineer, Date: Wednesday, Oct 02 2013 10:03 AM Pacific Daylight Time, Meaning: I acknowledge my electronic signature carries the same meaning as a handwritten signature

Workflow ID: 33535836, Doc Title: FP1284_L06_MammograpphyVirtualCpInterface.doc, Doc No:, Approver: Keith Gray (kgray), Title: Software Manager, Date: Wednesday, Oct 02 2013 01:33 PM Pacific Daylight Time, Meaning: I acknowledge my electronic signature carries the same meaning as a handwritten signature

Workflow ID: 33535836, Doc Title: FP1284_L06_MammograpphyVirtualCpInterface.doc, Doc No:, Approver: David Rollins (drollins), Title: Software Lead Quality Engineer, Date: Wednesday, Oct 02 2013 04:17 PM Pacific Daylight Time, Meaning: I acknowledge my electronic signature carries the same meaning as a handwritten signature

Workflow ID: 33535836, Doc Title: FP1284_L06_MammograpphyVirtualCpInterface.doc, Doc No:, Approver: Randall Weldemere (rweldeme), Title: DHF Coordinator, Date: Thursday, Oct 03 2013 09:06 AM Pacific Daylight Time, Meaning: I acknowledge my electronic signature carries the same meaning as a handwritten signature