PaxScan® Flat Panel Imagers

Virtual CP Communications Manual

VARIAN
medical systems

P/N 24581 Rev B
April 2007

Abstract The *PaxScan® Virtual CP Communication Manual (P/N 24581)* provides reference information and procedures for using Varian PaxScan digital imaging component sub-systems.

Technical Support If you cannot find information in this user guide, you can contact us in several ways:

- United States

  Varian X-Ray Products
  1678 So. Pioneer Rd.
  Salt Lake City, UT 84104

  + 1 800 432 4422 Phone
  + 1 801 972 5000 Phone
  + 1 801 973 5023 Fax

- Europe

  Varian X-Ray Products
  Zutphensestraat 160A
  6971ET Brummen
  The Netherlands

  + 31 62 33 66 020 Cell
  + 31 575 566 093 Phone
  + 31 575 566 538 Fax

- East Asia

  Varian X-Ray Products
  4th MY ARK Nihonbashi Bldg.
  10-16 Tomizawa-cho
  Nihonbashi, Chuo-ku
  Tokyo 103-0006, Japan

  + 81 03 5652 4711 Phone
  + 81 03 5652 4713 Fax

- China

  Varian X-Ray Products
  B3-1801, Sunshine Plaza
  68 Anli Road, Chaoyang District
  Beijing 100101, P.R. China

  + 86 10 6496 1585 Phone
  + 86 10 6494 0743 Fax

You can find more information about Digital Radiography on our Website: http://www.varian.com.

Notice Information in this manual is subject to change without notice and does not represent a commitment on the part of Varian to update information. Varian is not liable for errors contained in this guide or for any damages incurred in connection with furnishing or use of this material.

The PaxScan® 1313, 2520V, and 2520E, are Class 1, Type B component sub-systems per the Standard for Medical Electrical Equipment, UL 60601-1 and IEC 60601-1.

CE Mark Varian Medical Systems imaging products are designed and manufactured to meet the Low Voltage Directive 73/23/EEC and EMC 93/42/EEC.

Trademarks PaxScan® is a registered trademark and ViVA™ is a trademark of Varian Medical Systems, Inc. Microsoft® is a registered trademark and Windows™ is a trademark of Microsoft Corporation.

CHAPTER SUMMARY

Contents

Figures

Tables

Virtual CP Interface

This document describes the set of interface functions provided by VirtCp.dll. These functions are similar to those used by the 4030R and VIP Command Processor. Like the 4030R all processing occurs entirely on the host computer.

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 which is similar to that for the 4030R. 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.

The entry point to VirtCp.dll is not protected with a full state machine so it is incumbent on the user to make calls logically. Most of the time the VCP will handle situations silently or generate an error where appropriate, but there is no explicit protection against reentrant calls. 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 open a debug window. Debug messages produced by the dlls are shown and also written to a file 'HcpDebug.txt' when vip_set_debug(FALSE) is called.

In some instance 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 3rd column in Table 1-1 indicates mode applicability as COM (common), RAD, FLU(fluoro) or N/A (not available or not applicable).

Table 1-1 Function Index

1	vip_analog_offset_cal()	COM
2	vip_cal_control(..)	N/A
3	vip_cal_end()	N/A
4	vip_check_link()	COM
5	vip_close_link(..)	COM
6	vip_correct_image(..)	COM
7	vip_dcds_enable(..)	COM
8	vip_enable_auto_cal(..)	N/A
9	vip_enable_sw_handshaking(..)	RAD
10	vip_fluoro_dispose()	N/A
11	vip_fluoro_get_buffer_ptr(..)	FLU
12	vip_fluoro_get_event_name(..)	FLU
13	vip_fluoro_get_prms(..)	FLU
14	vip_fluoro_grabber_start(..)	FLU
15	vip_fluoro_grabber_stop()	FLU

Table 1-1 Function Index

16	vip_fluoro_init_mode(..)	FLU
17	vip_fluoro_init_sys(..)	N/A
18	vip_fluoro_record_start(..)	FLU
19	vip_fluoro_record_stop()	FLU
20	vip_fluoro_set_prms(..)	FLU
21	vip_gain_cal_prepare(..)	COM
22	vip_get_analog_offset_info(..)	N/A
23	vip_get_analog_offset_params(..)	COM
24	vip_get_auto_cal_settings(..)	N/A
25	vip_get_cal_info(..)	COM
26	vip_get_cal_limits(..)	N/A
27	vip_get_config_data	COM
28	vip_get_correction_settings(..)	COM
29	vip_get_current_mode(..)	COM
30	vip_get_dll_version(..)	COM
31	vip_get_gain_scaling_info(..)	N/A
32	vip_get_hw_config(..)	N/A
33	vip_get_image(..)	RAD
34	vip_get_image_counts(..)	N/A
35	vip_get_lih(..)	N/A
36	vip_get_mode_acq_type	RAD
37	vip_get_mode_info(..)	COM
38	vip_get_num_acq_frames(..)	RAD
39	vip_get_num_cal_frames(..)	COM
40	vip_get_offset_cal_shift(..)	COM
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
45	vip_get_sys_mode(..)	COM
46	vip_get_system_version_numbers(..)	COM
47	vip_get_video_timing(..)	N/A
48	vip_get_vista_parameters(..)	N/A
49	vip_get_wl(..)	N/A
50	vip_hw_reset()	N/A
51	vip_initialize_media	N/A
52	vip_io_enable(..)	RAD
53	vip_io_permit_exposure(..)	RAD
54	vip_io_query_status(..)	RAD

Table 1-1 Function Index

55	vip_offset_cal(..)	COM
56	vip_open_receptor_link(..)	COM
57	vip_put_config_data(..)	COM
58	vip_put_image(..)	COM
59	vip_query_error_info(..)	N/A
60	vip_query_prog_info(..)	COM
61	vip_reset_state(..)	COM
62	vip_select_mode(..)	COM
63	vip_select_receptor(..)	N/A
64	vip_self_test(..)	N/A
65	vip_set_analog_offset_params(..)	COM
66	vip_set_cal_acq_data(..)	N/A
67	vip_set_cal_limits(..)	N/A
68	vip_set_correction_settings(..)	COM
69	vip_set_debug(..)	COM
70	vip_set_frame_rate(..)	COM
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
77	vip_set_num_cal_frames(..)	COM
78	vip_set_offset_cal_shift(..)	COM
79	vip_set_rad_scaling(..)	N/A
80	vip_set_recursive_filter(..)	N/A
81	vip_set_sys_mode(..)	COM
82	vip_set_user_sync(..)	COM
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
87	vip_total_reset_media()	N/A
88	vip_validate_media(..)	N/A

Chapter 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.
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 Reserved1; int Reserved2; int Reserved3; int Reserved4; };

Table 2-1 Function Descriptions - Common

		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.
	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* This parameter should normally be 0. Multiple receptors are not yet supported. However, it is used with the 2520E and USB I/O device; in this case setting it to -1 results in the receptor powering down when the link closes.
	Remarks	This function frees resources and memory used by the interface (unloading all calibration data). Should only be called when the acquisition session is finished.
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).

If any of the requested corrections are not available, an error is generated and the return value indicates what corrections are available. See Chapter 5 **Non-fatal error codes** for additional information.

Table 2-1 Function Descriptions - Common

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*, *BOOL auto_sense* = FALSE);
	Parameters	*mode_num* The number of the mode for which gain calibration is to be performed. Should always be the currently selected mode. *auto_sense* This parameter should be set to *FALSE*, as it is used to request a feature that is not supported by the Virtual CP (parameter is ignored).
	Remarks	This function initiates a gain calibration. More information as to how a gain cal is done is included in the rad and fluoro sections.
22	**vip_get_analog_offset_info()**	
	Protocol	int vip_get_analog_offset_info(*int mode_num,* *SAnalogOffsetInfo* aop*);
	Parameters	*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.

Table 2-1 Function Descriptions - Common

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 theparameters that define how the analog offsetcalibration is performed.

Table 2-1 Function Descriptions - Common

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(SCalStats) float OfstMedian; float OfstStdDev; float GainMedian; float GainStdDev; float GainScaling; long Time; }; *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.
	Remarks	Retrieves calibration statistics from the VirtCp.dll.

Table 2-1 Function Descriptions - Common

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;* }; *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* Set to TRUE if correcting line noise, using data from a reserved area in the receptor. Not applicable to most receptors including 2520E.
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 return value indicates what corrections are available (as AND'd with the corrections requested) for the currently selected mode. See Chapter 5 **Non-fatal error codes** for additional information.

Table 2-1 Function Descriptions - Common

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. Must be at least 256 characters. *name* For future use. *size* For future use.
	Remarks	This function returns the version information for the DLL, revision letter, build number, date and time.
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     Reserved1;
    int     Reserved2;
    int     Reserved3;
    int     Reserved4;
    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 Chapter 5 **Acquisition Constants**.

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.
	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.

Table 2-1 Function Descriptions - Common

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;
    int         Reserved1;
    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.

Continued....

	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. *BorderPixel*s 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. *Reserved1* *Reserved2* *Reserved3* *Reserved4* These last 4 values are not used or for internal use – must be zero.
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; }; *StructSize* Must be set by caller to size of structure. *SystemMode* The system mode.
Remarks	Currently always zero.
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 Chapter 5 **System Number Version Types**. For a given system not all values will be supported and a VIP_NOT_IMPL_ERR may be returned. continued...

Table 2-1 Function Descriptions - Common

		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.
55	**vip_offset_cal()**	
	Protoco	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; }; *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.
	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 Chapter 5 **Non-fatal error codes** for additional information.

Table 2-1 Function Descriptions - Common

57	**vip_put_config_data()**	
	Protocol	int vip_put_config_data(*char *full_file_path, char *target_file_name*);
	Parameters	f*ull_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, int 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 Chapter 5 **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.

Table 2-1 Function Descriptions - Common

	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()**	int vip_query_prog_info(int uType, UQueryProgInfo* uq);
	Protocol	*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).
	Parameters	*uq* Pointer to a structure; generally this is SQueryProgInfo: struct SQueryProgInfo // uType = HCP_U_QPI { int StructSize; // Set to sizeof(SQueryProgInfo) int NumFrames; BOOL Complete; int NumPulses; BOOL ReadyForPulse; }; *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. *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.

Table 2-1 Function Descriptions - Common

		Complete Set to TRUE when download completes. Use this to determine when download completes not the comparison of *ProgressLimit* to *ProgressCurrent*.
	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 acquisition or calibration.
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 Chapter 5 **Non-fatal error codes** for additional information.
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 offse tcalibration.

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 theparameters that define how the analog offsetcalibration is performed.

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;
};
```

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
Set to TRUE if correcting line noise, using data from a reserved area in the receptor. Not applicable to most receptors including 2520E.

	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 Chapter 5 **Non-fatal error codes** 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.

Table 2-1 Function Descriptions - Common

		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. Also a minimum frame rate may be applicable. For the 2520E the minimum is normally 1 fps except when the frame override is used (see Chapter 6 **Debug Mode**).
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.

Table 2-1 Function Descriptions - Common

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; }; *StructSize* Must be set by caller to size of structure. *SystemMode* The system mode.
	Remarks	Currently not used as always zero.
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.

Table 2-1 Function Descriptions - Common

86	**vip_sw_handshaking()**	
	Protocol	int vip_sw_handshaking(*int signal_type, BOOL active*);
	Parameters	*signal_type* See Chapter 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).

The three I/O Control functions (beginning vip_io) are enabled only if a HcpIo*.dll has been successfully loaded. When a HcpIo*.dll is not loaded, these functions return VIP_NOT_IMPL_ERR and hardware handshaking is not available.

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.
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 Chapter 5 **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.
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.
52	**vip_io_enable()**	
	Protocol	int vip_io_enable(*int activeMode*);
	Parameters	*activeMode* Must be one of the I/O enable codes from table in Chapter 5 **I/O Control: Enable Codes**
	Remarks	Returns VIP_NOT_IMPL_ERR if HcpIo*.dll is not loaded.

Table 3-1 Function Descriptions – Rad Modes

53	**vip_io_permit_exposure()**	
	Protocol	int vip_io_permit_exposure();
	Parameters	None.
	Remarks	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 **HcpIo*.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 the table in Chapter 5 **I/O Support: I/O Control Machine States**). *expState* Pointer to int: set to the current state of the exposure control state machine (encoded according to the table in Chapter 5 **I/O Support: Exposure Control Machine States**). NULL may be used if this information is not required.
	Remarks	Returns VIP_NOT_IMPL_ERR if **HcpIo*.dll** is not 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 IN VERSION L.01. Use **vip_set_num_acq_frames()** to set the number of acquisition frames.

Table 3-1 Function Descriptions – Rad Modes

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.
86 vip_sw_handshaking()	
Protocol	int vip_sw_handshaking(*int signal_type, BOOL active*);
Parameters	*signal_type* See Chapter 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).

Table 3-1 Function Descriptions – Rad Modes

	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. 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)**.

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.

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. As of build 26, it is no longer a requirement currently that once **vip_fluoro_record_stop()** has been called, the grabber must be stopped (using **vip_fluoro_grabber_stop()**) and re-initialized before calling **vip_fluoro_record_start()** again; i.e. multi-segment recording is permitted; see Chapter 4 **Multi-Segment Recording** for additional information.

The **vip_fluoro_record_start()** has a single 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 Chapter 4 **Multi-Segment Recording** 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);
```

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 useage. Pointers to grab buffers (usually 2) can also be obtained by setting the bufType to any non-zero value.
	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).

Table 4-1 Function Descriptions – Fluoro Modes

	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.

Table 4-1 Function Descriptions – Fluoro Modes

14 vip_fluoro_grabber_start()	
Protocol	int vip_fluoro_grabber_start(*SAcqPrms* acqPrms*);
Parameters	a*cqPrms* 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; void* SnapBuf; 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.

Table 4-1 Function Descriptions – Fluoro Modes

	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. *SnapBuf* Not used in this implementation. Must be NULL. *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 Chapter 4 **Fluoro Parameter Calls** 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.

Table 4-1 Function Descriptions – Fluoro Modes

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.

Table 4-1 Function Descriptions – Fluoro Modes

		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. SeeChapter 4 **Multi-Segment Recording** 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.

Table 4-1 Function Descriptions – Fluoro Modes

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 Chapter 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)**.

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;       // set to sizeof(SSeqPrms)
    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   Reserved1;        // dflt = 0
    int   Reserved2;        // 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 Chapter 4 Introduction to the Fluoro Interface and Multi-Segment Recording.

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

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 sequence buffer most recently updated.

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
};
```

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.

Chapter 5 Error Codes and Constants

There are various constants and error codes used and returned by the PaxScan system imaging system. This section will discuss these values.

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.

Non-fatal Error Codes

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.

System Version Number Types

Various version types may be interrogated by the call vip_get_system_version_inf(). 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.

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	0	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) -OFFSET_IMAGE]/ GAIN_IMAGE
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.

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.

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 OR'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	Idle frame start pulses are generated, but hand switch is not active.
HS_ACTIVE	1	Hand switch may initiate an acquisition
HS_ACTIVE_SW	3	Software handshaking calls may initiate an acquisition

I/O Support: I/O Control Machine States (ioState) (Rad Modes)

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).

I/O Support: Exposure Control Machine States (expState) (Rad Modes)

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

I/O Support: Valid Combinations of I/O Control and Exposure Control State (Rad Modes)

ioState	expState	Description
IO_INIT	EXP_STANDBY	Normal initialization
IO_STANDBY	EXP_STANDBY	Normal idle state
IO_PREP	EXP_AWAITING_PERMISSION	Prep pressed
IO_PREP	EXP_PERMITTED	Prep pressed
IO_READY	EXP_AWAITING_PERMISSION	Ready pressed
IO_READY	EXP_PERMITTED	Ready pressed
IO_ACQ	EXP_PERMITTED	Awaiting imager
IO_ACQ	EXP_REQUESTED	expose_req asserted
IO_ACQ	EXP_CONFIRMED	A.L.E. from X-ray generator (expose_req remains asserted)
IO_FETCH	EXP_COMPLETED	Reading out image
IO_DONE	EXP_COMPLETED	Acquisition complete
IO_ABORT	EXP_STANDBY	Prep or Ready released too early
IO_ABORT	EXP_TIMED_OUT	Imager failed to read out image
IO_INIT_ERROR	EXP_STANDBY	Prep, Ready or A.L.E. asserted
IO_INIT_ERROR	EXP_COMPLETED	DLL mismatch detected
IO_INIT_ERROR	EXP_TIMED_OUT	Imager not responding

Figure 5.1 - I/O Control State Machine

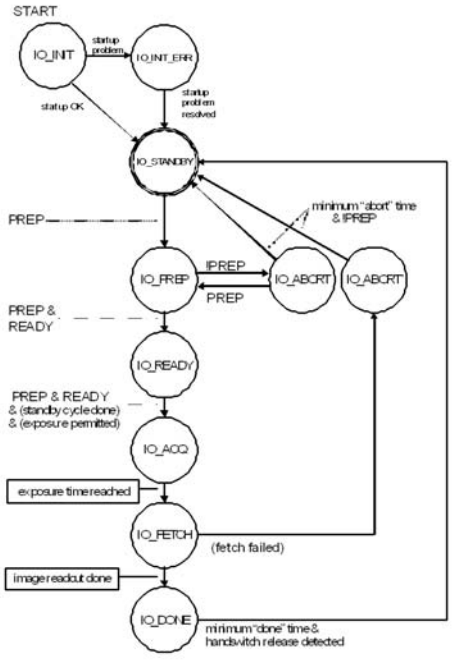
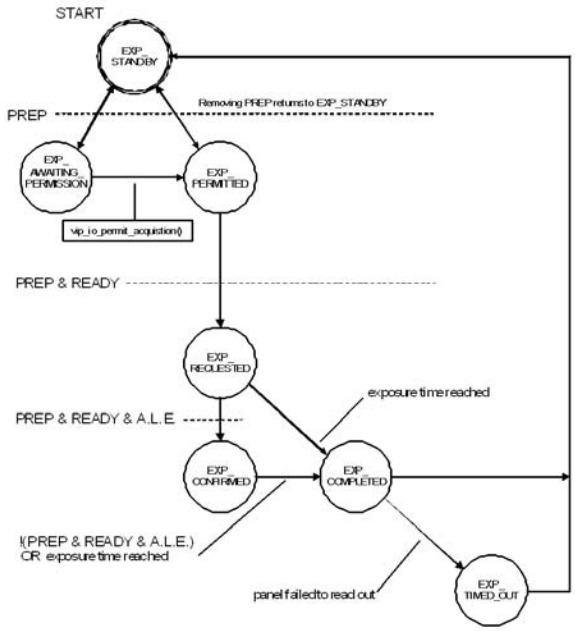


Figure 5.2 - Exposure Control State Machine



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

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).

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'.

Virtual CP Configuration File

The file HcpConfig.ini contains information as to the Virtual CP configuration. It specifies the submodules 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 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.

Frame Period Override

The Pleora module is used by the 2520E to generate frame start pulses, but allows only up approximately 1 s. frame periods. For longer periods it is possible to use the host computer to provide timing via the ethernet link (hence not as precise). For frame periods over 1000 ms., the following addition to the ini file may be made:


```
[ HcpFgPleora]
FramePeriodOverrideMilliSec=1500
```

In this example the frame period will be 1.5 s. When the link is closed, the Pleora will take over the frame start generation, so that the panel continues to be read out. Note that if this override mechanism is used, the frame period will be the same for all modes, and the call vip_set_frame_rate() will be inoperative.

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.

Pleora Configuration File

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

Chapter 7 Technical Support

How To Reach Us

In order to provide you with the most comprehensive technical support, please complete the following problem report before contacting Varian technical support personnel.

If you prefer e-mailing the information to us, a .pdf version of this form is included on the CD ROM you received with your system. You may also fax a completed copy of the problem report on the following page.

- To speak with our technical support personnel, call 1.800.432.4422 or 1.801.972.5000, or
- E-mail the report to paxscan.service@varian.com, then call the above number, or
- Fax a copy of the Problem Report to 1.801.973.5023

Index