VARIAN
medical systems

High-Level Ethernet and Serial Interface Functions

Table of Contents

To assist with navigation within this document, bookmark jumps, which are indicated as underlined text, are used throughout. Clicking on a title in an index listing will move the cursor to the beginning of the text of the title and vice versa.

I. INTRODUCTION

A set of interface functions are provided for controlling the VIP-9 Command Processor over an Ethernet or a serial connection, via a Win32 DLL called *VIP_Comm*, that include vip_comm.h, vip_comm.lib and vip_comm.dll.

In general, all function-return values are of type *int*. The return value always indicates the success/failure of the function call, with a non-zero value meaning an error has occurred. The definitions of the return values are listed in APPENDIX A at the end of this document.

Function arguments, which are declared to be pointers, are generally used for returning data to the caller. These pointers should be addresses of appropriate type variables, and not merely un-initialized pointers.

Finally, all function names are prefixed with *vip_* and all constant and structure names are prefixed with *VIP_*, to avoid namespace conflicts.

The function definitions for the Ethernet and serial external interface are listed in alphabetical order in the next section.

Note that the following functions are available only for the Ethernet version of the interface, because of bandwidth limitations for the serial interface:

- vip_get_config_data()
- vip_get_image()
- vip_put_config_data()
- vip_put_image()

II. INTERFACE FUNCTIONS

This section contains short summaries of the Ethernet and serial interface functions. It is intended for use as a quick reference by users of *vip_comm.dll*. The remaining of this section is divided into two tables. The first table lists function names. The other table contains the function descriptions.

FUNCTION INDEX

FUNCTION DESCRIPTIONS

Each function description has five subsections containing the following information: (1) Function name, (2) Function protocol as used in the Visual C++ *vip_comm.dll*, (3) Serial syntax as used in a terminal application such *Hyper Terminal*, (4) Descriptions of all parameters used in the function and (5) Remarks and notes about the function.

1. Function	**vip_analog_offset_cal()**
Protocol	*int* vip_analog_offset_cal (*int mode_num*);
Serial Syntax	@AOC<*mode_num*><CR>
Parameters	*mode_num* The number of the mode for which the Command Processor will initiate an analog offset calibration.
Remarks	This function allows the user to initiate an analog offset calibration for a specified mode.
2. Function	**vip_check_link()**
Protocol	*int* vip_check_link *void();*
Serial Syntax	@CKL<CR>
Parameters	None.
Remarks	Similar to the *ping* command, this function allows the user to check the communication link with the Command Processor. The user does not normally need to make this call.
3. Function	**vip_close_link()**
Protocol	*int* vip_close_link (*BOOL unregister* = *FALSE*);
Serial Syntax	@CLL<CR>
Parameters	*unregister* If *TRUE*, the link will be unregistered after it is closed. If *FALSE*, no un-registration will occur.
Remarks	This function allows the user to close the communications link. The VIP-9 software does not shut down, but enters a state where it is waiting for a client to establish a new link, via a vip_open_link() command. Note that in early version, it was necessary to call the function vip_register_application() prior to opening a link. This requirement was eliminated and the function vip_register_application() became obsolete. The function vip_close_link() should be called with the default value *FALSE* for *unregister*.

4. Function	**vip_enable_auto_cal()**	
Protocol	*int* vip_enable_auto_cal (*int mode_num*, *BOOL enable*, *int minimum_delay*, *int post_exposure_delay*);	
Serial Syntax	@EAC<*mode_num*>;<*enable*>;<*minimum_delay*>;<*post_exposure_delay*><CR>	
Parameters	*mode_num*	The number of the mode for which the Command processor will enable the auto-offset calibration option.
	enable	If *TRUE*, the auto-offset calibration option will be enabled. If *FALSE*, the auto-offset calibration option will be disabled.
	minimum_delay	The minimum time delay between auto-offset calibrations. It will be ignored if *enable* is *FALSE*. Units: seconds.
	post_exposure_delay	The amount of time delay required after an exposure. It will be ignored if *enable* is *FALSE*. Units: seconds.
Remarks	This function allows the user to enable or disable the auto-offset calibration.	
5. Function	**vip_enable_sw_handshaking()**	
Protocol	*int* vip_enable_sw_handshaking (*BOOL enable*);	
Serial Syntax	@ESH<*enable*><CR>	
Parameters	*enable*	If *TRUE*, software handshaking option will be enabled. If *FALSE*, the option will be disabled. At startup, the Command Processor enables both hardware and software handshaking options.
Remarks	This function allows the user to enable or disable the software handshaking option.	

6. Function	**vip_gain_cal_prepare()**	
Protocol	*int* vip_gain_cal_prepare (*int mode_num*, *BOOL auto_sense* = *FALSE*);	
Serial Syntax	@GCP<*mode_num*>;<*auto_sense*><CR>	
Parameters	*mode_num*	The number of the mode for which the Command Processor will prepare for a gain calibration.
	auto_sense	If *TRUE*, the Command Processor will perform a gain calibration by automatically sensing X-rays on/off. If *FALSE*, the Command Processor will wait for X-rays on/off messages.
Remarks	This function allows the user to prepare the Command Processor for a gain calibration. The minimum-required interval between gain calibrations is expected to be fairly long, on the order of weeks. After this function is called, the system operator should verify that the correct flat field conditions exist and the X-ray beam should be established. The calibration then proceeds when the client sends the proper hardware handshaking signals or equivalent software signals given software handshaking option is enabled and *auto_sense* is *FALSE*.	

7. Function	**vip_get_analog_offset_data()**	
Protocol	*int* vip_get_analog_offset_data	(*int mode_num*, *int *target_value*, *int *tolerance*, *int *median_percent*, *double *fractional_iteration_delta*, *int *number_iterations*);
Serial Syntax	@GAO<*mode_num*><CR>	
Parameters	*mode_num*	The number of the mode whose analog offset data is to be set at the Command Processor.
	target_value	The target value where analog offset calibration occurs.
	tolerance	The *tolerance* value around the *target_value* that makes the acceptable range of values for the analog offset calibration.
	median_percent	The percentage of offset values accepted which are below the target value.
	fractional_iteration_delta	The scaling factor applied in determining the offset adjustment between iterations. It influences the speed of convergence.
	number_iterations	The number of iterations that will be attempted to bring the offsets within range.
Remarks	This function allows the user to retrieve the parameters that define how the analog offset calibration is performed.	

8. Function	**vip_get_analog_offset_stats()**	
Protocol	**This function is no longer supported.**	
	int vip_get_analog_offset_stats (*int mode_mum*, *int num_asics*, *double *standard_deviation*, *int *segment_defect_found*);	
Serial Syntax	**Serial communication for this function is no longer supported.**	
	@GAS<*mode_mum*>;<*num_asics*><CR>	
Parameters	*mode_mum*	The number of the mode whose analog offset statistics is to be retrieved from the Command Processor.
	num_asics	The number of ASIC's in the system.
	standard_deviation	The current standard deviation of the analog offset.
	segment_defect_found	The number of defect segments that have been detected.
Remarks	This function allows the user to retrieve the offset statistics from the Command Processor.	

9. Function	**vip_get_cal_stats()**	
Protocol	*int* vip_get_cal_stats (*int mode_num*, *int *gain_median*, *double *gain_sigma*, *int *offset_median*);	
Serial Syntax	@GCS<*mode_num*><CR>	
Parameters	*mode_mum*	The number of the mode whose calibration statistics is to be retrieved from the Command Processor.
	gain_median	The number of ASIC's in the system.
	gain_sigma	The current standard deviation of the analog offset.
	offset_median	The number of defect segments that have been detected.
Remarks	This function allows the user to retrieve calibration statistics from the Command Processor.	

10. Function	**vip_get_config_data()**	
Protocol	*int* vip_get_config_data (char *full_file_path, char *target_file_name);	
Serial Syntax	**Serial communication for this function is not supported.**	
	@GCD<CR>	
Parameters	*full_file_path*	A null-terminated string, which is the fully qualified path name of the file that contains the information to be retrieved from the Command Processor. Up to 256 characters is allowed, including the null-termination character.
	target_file_name	A null-terminated string, which is the name of the file that contains the data on the Command Processor. Up to 32 characters is allowed, including the termination character. Currently, only the configuration data file named "ConfigDataFile" is available for this function. Please, see APPENDIX B for complete listing of available file names.
Remarks	This function allows the user to retrieve a file such as the configuration data, software or firmware file from a Command Processor.	
11. Function	**vip_get_correction()**	
Protocol	*int* vip_get_correction (BOOL *offset_cal, BOOL *gain_cal, BOOL *defect_map, BOOL *line_noise);	
Serial Syntax	@GCR<CR>	
Parameters	*offset_cal*	If *TRUE*, the Command Processor will use offset-calibrated data during the correction process. If *FALSE*, it will not apply the data on the correction process. If *offset_cal* is *FALSE*, then *gain_cal* is also assumed to be *FALSE*.
	gain_cal	If *TRUE*, the Command Processor will use gain-calibrated data during the correction process. If *FALSE*, it will not apply the data on the correction process.
	defect_map	If *TRUE*, the Command Processor will use defect-map data during the correction process. If *FALSE*, it will not apply the data on the correction process and defective pixels will be visible.
	line_noise	If *TRUE*, the Command Processor will use digital line noise reduction during the correction process. If *FALSE*, it will not use this method.
Remarks	This function allows the user to retrieve information on which correction algorithms are being applied to incoming pixel data. These parameters are global for all modes.	

12. Function	**vip_get_current_mode()**	
Protocol	*int* vip_get_current_mode (*int *mode_num*);	
Serial Syntax	@GCM<CR>	
Parameters	*mode_num*	The number of the currently selected mode.
Remarks	This function allows the user to retrieve the number of the currently selected mode in the Command Processor.	

13. Function	**vip_get_image()**	
Protocol	*int* vip_get_image (*int mode_num*, *int image_type*, *int x_size*, *int y_size*, *WORD *image_ptr*);	
Serial Syntax	**Serial communication for this function is not supported.**	
	@GMG<*mode_num*><*image_type*><*x_size*><*y_size*><CR>	
Parameters	*mode_num*	The number of the mode whose image is to be retrieved from the Command Processor.
	image_type	The image type of the image to be retrieved. Please, see APPENDIX C for complete listing of available image types.
	x_size	The horizontal size of the image to be retrieved. Units: number of pixels.
	y_size	The vertical size of the image to be retrieved. 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 via the Ethernet connection. If the Command Processor is actively acquiring images, no action will be taken and it will return VIP_STATE_ERR for this function.	

14. Function	**vip_get_lih()**	
Protocol	**This function is not supported.**	
	int vip_get_lih (*int mode_num*, *BOOL *lih_active*);	
Serial Syntax	**Serial communication for this function is not supported.**	
	@GLH<*mode_num*><CR>	
Parameters	*mode_num*	The number of the mode whose *l*ast *i*mage *h*old (*lih*) option is to be retrieved from the Command Processor.
	lih	If *TRUE*, lih filtering will be applied when fluoroscopy acquisition is stopped. If *FALSE*, lih filtering will not be applied. The default value of option is *FALSE*. It is ignored for accumulation or radiography type acquisition.
Remarks	This function allows the user to retrieve the last image hold status. Note that lih filtering is only valid in fluoroscopic-type modes.	

15. Function	**vip_get_mode_acq_type()**	
Protocol	*int* vip_get_mode_acq_type	(*int mode_num*, *int *mode_acq_type*, *int *num_frames*);
Serial Syntax	@GMA<*mode_num*><CR>	
Parameters	*mode_num*	The number of the mode whose mode acquisition type is to be retrieved from the Command Processor.
	mode_acq_type	The mode acquisition type as described in APPENDIX E.
	num_frames	The number of frames used to terminate an acquisition process in mode acquisition types VIP_VALID_XRAYS_N_FRAMES and VIP_AUTO_SENSE_N_FRAMES as described in APPENDIX E. This parameter is not used for other mode acquisition types and might contain invalid data.
Remarks	This function allows the user to retrieve the mode acquisition type of a mode.	

16. Function	**vip_get_mode_details()**	
Protocol	*int* vip_get_mode_details	(*int mode_num*, *int *acq_type*, *double *frame_rate*, *double *analog_gain*, *int *lines_per_frame*, *int *columns_per_frame*, *int *lines_per_pixel*, *int *columns_per_pixel*, *char *mode_description*);
Serial Syntax	@GMD<*mode_num*><CR>	
Parameters	*mode_num*	The number of the mode for which mode details will be retrieved from the Command Processor.
	acq_type	The acquisition type defined as VIP_ACQ_TYPE_CONTINUOUS or VIP_ACQ_TYPE_ACCUMULATION, which is described in APPENDIX E.
	frame_rate	The frame rate at the receptor or the number of times the receptor is read out per second. Units: Hz. A zero value indicates that the Command Processor cannot return the proper frame rate value.
	analog_gain	The frond-end gain value applied to the signals before digital-to-analog conversion. A zero value indicates that the Command Processor cannot return the proper analog gain value.
	lines_per_frame	The number of lines per frame of an image or the vertical resolution. This value may be less than the maximum resolution of the receptor if this mode uses pixel binning.
	columns_per_frame	The number of columns per frame of an image data or the horizontal resolution. This value may be less than the maximum resolution of the receptor if this mode uses pixel binning.
	lines_per_pixel	The number of lines per pixel of an image or the number of receptor pixels, which are binned in the vertical direction to make one image pixel. A zero value indicates that the system cannot return the proper value.
	columns_per_pixel	The number of columns per pixel of an image or the number of receptor pixels, which are binned in the horizontal direction to make one image pixel. A zero value indicates that the system cannot return the proper value.
	mode_description	A text string, which describes the mode. The calling program must allocate a string, which can hold up to 32 characters, including the termination character.
	dcds_enable	True if DCDS is enabled, false is DCDS is disabled.
Remarks	This function allows the user to retrieve detailed setting information of a mode.	

17. F unction	**vip_get_num_acq_frames()**	
Protocol	*int* vip_get_num_acq_frames (*int mode_num*, *int *num_acq_frames*);	
Serial Syntax	@GAF<*mode_num*><CR>	
Parameters	*mode_num*	The number of the mode whose number of acquired frames is to be retrieved from the Command Processor.
	num_acq_frames	-1: Start/stop using radiation auto-sense; 0: Start/stop using hardware or software handshaking; *n* > 0: Start using hardware or software handshaking, stop after *n* frames have been accumulated.
Remarks	This function allows the user to retrieve the number of acquired frames that defines how the accumulation-type mode functions. Note for Command Processor software of revision H or later, a similar but easy to use command, vip_get_mode_acq_type() should be called in this function's place.	

18. F unction	**vip_get_num_cal_frames()**	
Protocol	*int* vip_get_num_cal_frames (*int mode_num*, *int *num_cal_frames*);	
Serial Syntax	@GCF<*mode_num*><CR>	
Parameters	*mode_num*	The number of the mode whose number of calibrating frames is to be retrieved from the Command Processor.
	num_cal_frames	The number of calibrating frames. The calibration acquisition should end after this number of frames have been acquired; values should be a power of 2, between 2 and 1,024. Non-conforming values will be rounded down to the nearest power of 2. For example, the number of calibrating frames of 30 is greater than 16 or 2^4 but less than 32 or 2^5. Thus, it will be rounded down to 16.
Remarks	This function allows the user to retrieve the number of frames used for calibrating a mode from the Command Processor.	

19. Function	**vip_get_rad_scaling()**	
Protocol	*int* vip_get_rad_scaling (*int mode_num*, int **scaling_type*, *int *target_value*);	
Serial Syntax	@GRS<*mode_num*><CR>	
Parameters	*mode_num*	The number of the mode whose scaling information is to be retrieved from the Command Processor.
	scaling_type	The scaling type of the mode. Possible values for this parameter are VIP_RAD_SCALE_NONE, VIP_RAD_SCALE_UP, VIP_RAD_SCALE_DOWN and VIP_RAD_SCALE_BOTH as described in APPENDIX E.
	target_value	The target value where scaling occurs.
Remarks	This function allows the user to retrieve the scaling information that defines how an accumulation-type mode image is scaled by the Command Processor.	
20. Function	**vip_get_recursive_filter()**	
Protocol	*int* vip_get_recursive_filter (*int mode_num*, *double *buffer_weight*);	
Serial Syntax	@GRF<*mode_num*><CR>	
Parameters	*mode_num*	The number of the mode whose recursive filtering information is to be retrieved from the Command Processor.
	buffer_weight	The fractional contribution to each output frame by the data already in the recursive filter buffer. A value of 0.0 means no recursive filtering is applied. This value is ignored for radiography modes.
Remarks	This function allows the user to retrieves the recursive filter setting for a mode form the Command Processor.	
21. Function	**vip_get_self_test_log()**	
Protocol	**This function is not currently implemented and reserved for future revision.**	
	int vip_get_self_test_log (*void*)	
Serial Syntax	**This function is not currently implemented and reserved for future revision.**	
	@GST<CR>	
Parameters	*To be determined.*	
Remarks	This function allows the user to retrieve the self-test results generated when vip_self_test() is called.	

22. F unction	vip_get_system_info()	
Protocol	*int* vip_get_system_info	(*int *num_modes*, *int *default_mode_num*, *int *max_lines_per_frame*, *int *max_columns_per_frame*, *int *max_pixel_value*, *BOOL *has_video*, *char *system_description*, *int *startup_configuration* = NULL, *int *num_asics* = NULL);
Serial Syntax	@GSI<CR>	
Parameters	*num_modes*	The number of modes available in the Command Processor.
	default_mode_num	The number of the default mode. It is normally set to zero.
	max_lines_per_frame	The number of lines per frame of an image or the vertical resolution of the full panel.
	max_columns_per_frame	The number of columns per frame of an image or the horizontal resolution of the full panel.
	max_pixel_value	The maximum value of a pixel in the Command Processor. For a Command Processor with 12-bit A/D conversion, this value will be 4095. Note that the minimum pixel value is always zero.
	has_video	If *TRUE*, the system is equipped with a video monitor. If *FALSE*, the system is not equipped with a video monitor.
	system_description	A text string, which describes the system. The calling program must allocate a string, which can hold up to 32 characters, including the termination character.
	startup_configuration	The configuration that determines which activity the Command Processor will do at startup time. Possible values for this parameter are VIP_DISPL_TST_PTTRN_STRTUP, VIP_STANDALONE_STRTUP and VIP_ACTIVE_ACQ_STRTUP as described in APPENDIX D.
	num_asics	The number of ASICs in the system.
	receptor_type	The type of receptor, either normal (1) or DCDS (2).
Remarks	This function allows the user to retrieve the system information from the Command Processor.	

23. Function	**vip_get_system_version_numbers()**	
Protocol	*int* vip_get_system_version_numbers (*int sys_ver_type*, *char *ver_str*);	
Serial Syntax	@GSV<*sys_ver_type*><CR>	
Parameters	*sys_ver_type*	A system version number type as described in APPENDIX D.
	ver_str	A text string, which contains the returned system version number. The calling program must allocate a string, which can hold up to 32 characters, including the termination character.
Remarks	This function allows the user to retrieve a version number string of a system component. The calling program must allocate a string, which can hold up to 32 characters, including the termination character for the version number.	

24. Function	**vip_get_wl()**	
Protocol	**This function is not currently implemented and reserved for future revision.**	
	int vip_get_wl (*int *bot*, *int *top*, *int *mapping*);	
Serial Syntax	**This function is not currently implemented and reserved for future revision.**	
	@GWL<CR>	
	bot	The pixel value below which a pixel is mapped to the lower of the low-contrast region of the mapping curve. For mapping type VIP_WL_MAPPING_LINEAR, this pixel value is set to zero.
	top	The pixel value above which a pixel is mapped to the upper, low-contrast region of the mapping curve. For mapping type VIP_WL_MAPPING_LINEAR, this pixel value is set to the maximum grayscale value.
	mapping	The mapping type, which determines how 16-bit pixel data is to be translated into 8-bit pixel data for display. Possible values for this parameter are VIP_WL_MAPPING_LINEAR, VIP_WL_MAPPING_NORM_ATAN and VIP_WL_MAPPING_CUSTOM as described in APPENDIX D.
Remarks	This function allows the user to retrieve the current window/level translation settings from the Command Processor. Generally, window-level mapping is used to increase the contrast for certain anatomy or regions of interest. Note that this function is only valid when an optional card to provide 8-bit video display is installed.	

25. Function	**vip_offset_cal()**	
Protocol	*int* vip_offset_cal (*int mode_num*);	
Serial Syntax	@OFC<*mode_num*><CR>	
Parameters	*mode_num*	The number of the mode whose offset calibration is to be performed.
Remarks	This function allows the user to initiate an offset calibration on the Command Processor. It is not normally necessary for the user to call this function, unless automatic offset calibration is disabled. Please, see the function vip_enable_auto_cal().	

26. Function	**vip_open_link()**	
Protocol	*int* vip_open_link (*int new_link* = VIP_ETHERNET_LINK, *const char *client_ip_address = NULL*, *const char *server_ip_address = NULL*, *int com_port* = VIP_COM1);	
Serial Syntax	@OPL<CR>	
Parameters	*new_link*	The link type to be established. Possible values for this parameter are VIP_ETHERNET_LINK and VIP_SERIAL_LINK as described in APPENDIX F.
	client_ip_address	The Internet Protocol address of the host computer such as "132.190.17.179".
	server_ip_address	The Internet Protocol address of the Command Processor such as "132.190.17.169".
	com_port	Serial com-port to be established. Possible values for this parameter are VIP_COM1, VIP_COM2, VIP_COM3 and VIP_COM4 as described in APPENDIX F.
Remarks	This function allows the user to establish a communication link between the host computer and the Command Processor. This function must be called upon system startup, restart, or if the function vip_close_link() has been called. Note when *new_link* is equal to VIP_ETHERNET_LINK, the serial parameter, *com_port* is ignored. Likewise, when *new_link* is equal to VIP_SERIAL_LINK, the parameters *client_ip_address* and *server_ip_address* are ignored.	

27. F unction	**vip_put_config_data()**	
Protocol	*int* vip_put_config_data (*char *full_file_path*, char **target_file_name*);	
Serial Syntax	**Serial communication for this function is not supported.**	
	@PCD<CR>	
Parameters	*full_file_path*	A null-terminated string which is the fully qualified path name of the file that contains the information to be stored on the Command Processor. Up to 256 characters is allowed, including the termination character.
	target_file_name	A null-terminated string, which is the file name to be used to store the data on the Command Processor. Up to 32 characters is allowed, including the termination character. Currently, only the configuration data file named “ConfigDataFile” is available for this function. Please, see APPENDIX B for complete listing of available file names.
Remarks	Use of this function is currently reserved. Its purpose is to permit dynamic updating of configuration information for manufacturing and test purposes.	
28. F unction	**vip_put_image()**	
Protocol	*int* vip_put_image (*int mode_num*, *int image_type*, *int x_size*, *int y_size*, *WORD *image_ptr*);	
Serial Syntax	**Serial communication for this function is not supported.**	
	@PMG<CR>	
Parameters	*mode_num*	The number of the mode to which an image is to be transmitted.
	image_type	The image type of the image to be transmitted. Please, see APPENDIX C for complete listing of available image types.
	x_size	The horizontal size of the image to be transmitted. Units: number of pixels.
	y_size	The vertical size of the image to be transmitted. Units: number of pixels.
	image_ptr	A pointer to a memory block which holds the image. The block must be at least of size (2 * x_size * y_size) bytes.
Remarks	This function allows the user to transmit an image from the host computer to the Command Processor via the Ethernet link. If the Command Processor is actively acquiring an image, no action will be taken and the error code VIP_STATE_ERR will be returned.	

29. F unction	**vip_query_error()**
Protocol	**This function is not currently implemented and reserved for future revision.**
	int vip_query_error (*int *error_mask*);
Serial Syntax	**This function is not currently implemented and reserved for future revision.**
	@QER<CR>
Parameters	*error_mask* The error mask that contains error codes described in APPENDIX G.
Remarks	This function allows the user to retrieve the error information on the Command Processor.

30. F unction	**vip_query_progress()**	
Protocol	*int* vip_query_progress (*int *num_frames*, *BOOL *complete*, *int *num_pulses = NULL*, *BOOL *ready_for_pulse = NULL*);	
Serial Syntax	@QPR<CR>	
Parameters	*num_frames*	The number of frames acquired. If self-test is the current or most recently executed process, this value is set to zero.
	complete	If *TRUE*, the acquisition or calibration process is complete. If *FALSE*, the process is in progress.
	num_pulses	The number of "pulses" or X-rays on/off sequences that are detected during the acquisition or calibration.
	ready_for_pulse	If *TRUE*, the Command Processor is ready for the next X-rays "ON" command. If *FALSE*, the Command Processor is ready for the next X-rays "OFF" command.
Remarks	This function allows the user to query the Command Processor about its progress in the course of an image acquisition or calibration. For non-accumulation type modes, *num_pulses* will be equal to *num_frames* and *ready_for_pulse* is not used.	

31. F unction	**vip_reset_state()**
Protocol	*int* vip_reset_state (*void*);
Serial Syntax	@RSS<CR>
Parameters	None.
Remarks	This function allows the user to abort any in-process acquisition or calibration. The Command Processor will return to the default mode.
32. F unction	**vip_select_mode()**
Protocol	*int* vip_select_mode (*int mode_num*);
Serial Syntax	@SLM< *mode_num* ><CR>
Parameters	*mode_num* The number of the mode to which the Command Processor will switch.
Remarks	This function allows the user to select a mode of operation corresponding to the given mode number. If the Command Processor is in an idle state, acquisition does not actually begin until the *prepare* flag is set to *TRUE* either via hardware or software. This function can be used to jump directly from one active acquisition mode to another active acquisition mode with minimal delay. If the Command Processor is in a calibration mode, no action will be taken and the error code VIP_STATE_ERR will be returned.
33. F unction	**vip_self_test()**
Protocol	**This function is not currently implemented and reserved for future revision.**
	int vip_self_test (*void*);
Serial Syntax	**This function is not currently implemented and reserved for future revision.**
	@STT<CR>
Parameters	None.
Remarks	This function allows the user to initiate a self-test on the Command Processor. The function vip_query_progress() can be used to poll the Command Processor to determine when the self test is complete. In this case, the parameter num_frames of the query function has no meaning. The function vip_get_self_test_log() can be used to retrieve the results of the self test process, once it is complete. If the Command Processor is actively acquiring, no action will be taken and the error code VIP_STATE_ERR will be returned.

34. F unction	**vip_set_analog_offset_data()**	
Protocol	*int* vip_set_analog_offset_data (*int mode_num*, *int target_value*, *int tolerance*, *int median_percent*, *double fractional_iteration_delta*, *int number_iterations*);	
Serial Syntax	@SAO<*mode_num*><*target_value*><*tolerance*><*median_percent*><*fractional_iteration_delta*><*number_iterations*><CR>	
Parameters	*mode_num*	The number of the mode whose analog offset data is to be set at the Command Processor.
	target_value	The target value where analog offset calibration occurs.
	tolerance	The *tolerance* value around the *target_value* that makes the acceptable range of values for the analog offset calibration.
	median_percent	The percentage of offset values accepted which are below the target value.
	fractional_iteration_delta	The scaling factor applied in determining the offset adjustment between iterations. It influences the speed of convergence.
	number_iterations	The number of iterations that will be attempted to bring the offsets within range.
Remarks	This function allows the user to set the parameters that define how the analog offset calibration is performed.	

35. Function	**vip_set_correction()**	
Protocol	*int* vip_set_correction (*BOOL offset_cal*, *BOOL gain_cal*, *BOOL defect_map*, *BOOL line_noise*);	
Serial Syntax	@SCR<*offset_cal*><*gain_cal*><*defect_map*><*line_noise*><CR>	
Parameters	*offset_cal*	If *TRUE*, the Command Processor will use offset-calibrated data during the correction process. If *FALSE*, it will not apply the data on the correction process. If *offset_cal* is *FALSE*, then *gain_cal* is also assumed to be *FALSE*.
	gain_cal	If *TRUE*, the Command Processor will use gain-calibrated data during the correction process. If *FALSE*, it will not apply the data on the correction process.
	defect_map	If *TRUE*, the Command Processor will use defect-map data during the correction process. If *FALSE*, it will not apply the data on the correction process and defective pixels will be visible.
	line_noise	If *TRUE*, the Command Processor will use digital line noise reduction during the correction process. If *FALSE*, it will not use this method.
Remarks	This function allows the user to enable and/or disable the offset calibration correction, gain calibration correction, defective pixel correction and digital line noise correction. These parameters are global for all modes. If the Command Processor is actively acquiring, no action will be taken and the error code VIP_STATE_ERR will be returned.	

36. Function	**vip_set_debug()**	
Protocol	*int* vip_set_debug (*BOOL enable*);	
Serial Syntax	@SDB<*enable*><CR>	
Parameters	*enable*	If *TRUE*, the Command Processor will display debug information on a Hyper Terminal window. If *FALSE*, no debug information will be displayed.
Remarks	This function allows the user to enable or disable the display of debug information from the Command Processor on a Hyper Terminal window.	

37. F unction	**vip_set_frame_rate()**	
Protocol	*int* vip_set_frame_rate (*int mode_num*, *double frame_rate*);	
Serial Syntax	@SFR<*mode_num*><*frame_rate * 1000* ><CR>	
Parameters	*mode_num*	The number of the mode whose frame rate is to be set.
	frame_rate	The frame rate in frames per second (fps). For modes with image size 768 x 960, valid frame rates are 7.5, 15.0 and 30.0 fps. And for modes with image size 1536 x 1920, valid frame rates are 1.0, 2.0, 3.0, 3.75, 5.0 and 7.5 fps.
Remarks	This function allows the user to set the frame rate of a mode. Please, see the function vip_get_mode_details() for how to retrieve the image size and frame rate of a mode. Note the serial syntax specifies that the frame rate is multiply by 1000. This is to reserved if the accuracy of up to three decimal digits.	

38. F unction	**vip_set_lih()**	
Protocol	*int* vip_set_lih (*int mode_num*, *BOOL lih_active*);	
Serial Syntax	@SLH<*mode_num*><*lih_active*><CR>	
Parameters	*mode_num*	The number of the mode whose *l*ast *i*mage *h*old (*lih*) option is to be set.
	lih_active	If *TRUE*, lih filtering will be applied when fluoroscopy acquisition is stopped. If *FALSE*, lih filtering will not be applied. The default value of option is *FALSE*. It is ignored for accumulation radiography type acquisition.
Remarks	This function allows the user to enable and disable the lih option. The lih option is only valid for fluoroscopy acquisition type defined as VIP_ACQ_TYPE_CONTINUOUS in APPENDIX E. It is ignored for radiography type. If the Command Processor is actively acquiring an image, no action will be taken and the error code VIP_STATE_ERR will be returned.	

39. Function	**vip_set_mode_acq_type()**
Protocol	*int* vip_set_mode_acq_type (*int mode_num*, *int acq_type*, *int num_frames*);
Serial Syntax	@SMA<*mode_num*><*acq_type*><*num_frames*><CR>
Parameters	*mode_num* — The number of the mode whose mode acquisition type is to be set.
	acq_type — The mode acquisition type as described in APPENDIX E.
	num_frames — The number of frames used to terminate an acquisition process in mode acquisition types VIP_VALID_XRAYS_N_FRAMES and VIP_AUTO_SENSE_N_FRAMES as described in APPENDIX E. This parameter is not used for other mode acquisition types and ignored by the Command Processor. It should be in the range between 0 to 255.
Remarks	This function allows the user to set the mode acquisition type of a mode.

40. Function	**vip_set_num_acq_frames()**
Protocol	*int* vip_set_num_acq_frames (*int mode_num*, *int num_acq_frames*);
Serial Syntax	@SAF<*mode_num*><*num_acq_frames*><CR>
Parameters	*mode_num* — The number of the mode whose number of acquired frames is to be set on the Command Processor.
	num_acq_frames — -1: Start/stop using radiation auto-sense; 0: Start/stop using hardware or software handshaking; *n* > 0: Start using hardware or software handshaking, stop after *n* frames have been accumulated. *n* should be less than or equal to 255.
Remarks	This function allows the user to set the number of acquired frames that defines how the accumulation-type mode functions. The parameter *num_acq_frames* is only valid for radiography acquisition type or VIP_ACQ_TYPE_ACCUMULATION defined in APPENDIX E. It is ignored otherwise. If the Command Processor is actively acquiring an image, no action will be taken, and the error code VIP_STATE_ERR will be returned. Note for Command Processor software of revision H or later, a similar but easy to use command, vip_set_mode_acq_type() should be called in this function's place.

41. Function	**vip_set_num_cal_frames()**	
Protocol	*int* vip_set_num_cal_frames (*int mode_num*, *int num_cal_frames*);	
Serial Syntax	@SCF<*mode_num*><*num_cal_frames*><CR>	
Parameters	*mode_num*	The number of the mode whose number of calibrating frames is to be set on the Command Processor.
	num_cal_frames	The number of calibrating frames. The calibration acquisition should end after this number of frames have been acquired; values should be a power of 2, between 2 and 1,024. Non-conforming values will be rounded down to the nearest power of 2. For example, the number of calibrating frames of 30 is greater than 16 or 2^4 but less than 32 or 2^5. Thus, it will be rounded down to 16.
Remarks	This function allows the user to set the number of frames used for calibrating a mode from the Command Processor. For low dose rate and high frame rate techniques such as fluoroscopy, settings from 128 to 512 should be adequate. For higher dose rate and lower frame rate techniques such as radiography, fewer frames might be acceptable, e.g. 32.	

42. Function	**vip_set_rad_scaling()**	
Protocol	*int* vip_set_rad_scaling (*int mode_num*, *int scaling_type*, *int target_value*);	
Serial Syntax	@SRS<*mode_num*><*scaling_type*><*target_value*><CR>	
Parameters	*mode_num*	The number of the mode whose scaling information is to be set on the Command Processor.
	scaling_type	The scaling type of the mode. Possible values for this parameter are VIP_RAD_SCALE_NONE, VIP_RAD_SCALE_UP, VIP_RAD_SCALE_DOWN and VIP_RAD_SCALE_BOTH as described in APPENDIX E.
	target_value	The target value where scaling occurs.
Remarks	This function allows the user to set the scaling information that defines how an accumulation-type mode is scaled on the Command Processor.	

43. Function	vip_set_recursive_filter()	
Protocol	*int* vip_set_recursive_filter (*int mode_num*, *double buffer_weight*);	
Serial Syntax	@SRF<*mode_num*><*buffer_weight*><CR>	
Parameters	*mode_num*	The number of the mode whose recursive filter attribute is to be set on the Command Processor.
	buffer_weight	The fractional contribution to each output frame by the data already in the recursive filter buffer. A value of 0.0 means no recursive filtering is applied and ignored for radiography modes.
Remarks	This function allows the user to set the buffer weight of the recursive filter. This value is only valid for the acquisition type VIP_ACQ_TYPE_CONTINUOUS and ignored for other acquisition types. If the Command Processor is actively acquiring an image, no action will be taken and the error code VIP_STATE_ERR will be returned.	

44. Function	vip_set_wl()	
Protocol	**This function is not currently implemented and reserved for future revision.**	
	int vip_set_wl (*int bot*, *int top*, *int mapping*);	
Serial Syntax	**This function is not currently implemented and reserved for future revision.**	
	@SWL<*bot*><*top*><*mapping*><CR>	
Parameters	*bot*	The pixel value below which a pixel is mapped to the lower of the low-contrast region of the mapping curve. For mapping type VIP_WL_MAPPING_LINEAR, this pixel value is set to zero.
	top	The pixel value above which a pixel is mapped to the upper, low-contrast region of the mapping curve. For mapping type VIP_WL_MAPPING_LINEAR, this pixel value is set to the maximum grayscale value.
	mapping	The mapping type, which determines how 16-bit pixel data is to be translated into 8-bit pixel data for display. Possible values for this parameter are VIP_WL_MAPPING_LINEAR, VIP_WL_MAPPING_NORM_ATAN and VIP_WL_MAPPING_CUSTOM as described in APPENDIX D.
Remarks	This function allows the user to set the current window/level translation settings on the Command Processor. Generally, window-level mapping is used to increase the contrast for certain anatomy or regions of interest. Note that this function is only valid when an optional card to provide 8-bit video display is installed.	

45. Function	**vip_sw_handshaking()**	
Protocol	*int* vip_sw_handshaking (*int signal_type*, *BOOL active*);	
Serial Syntax	@SHS<*signal_type*><*active*><CR>	
Parameters	*signal_type*	A simulated signal type as defined in APPENDIX D.
	active	If *TRUE*, the signal is to be enabled. If *FALSE*, it is to be disabled.
Remarks	This function allows the user to simulate a signal in software.	

III. Examples

The purpose of this section is to provide a set of examples that can be used to write a software interface for the VIP-9 system. These examples may need to be expanded to fit more specific user requirements.

To use the communications interface, the file vip_comm.h from the comm_dll folder must be included. The vip_comm.lib must be linked with the application and the vip_comm.dll file must be in the search path. The examples were developed as a console application using Microsoft Visual C++ 6.0 so there are a few specifics included to compile within that framework (i.e. #include <windows.h> and the function SleepEx).

For system connection, each example is self-contained; i.e. each example establishes a connection with the system and then closes that connection using vip_open_link and vip_close_link. This is not the expectation for a typical system. The connection should be established once for numerous operations. The IP addresses will need to be updated accordingly for the vip_open_link function.

The examples also assume that mode 0 is a fluoroscopy type mode and that mode 1 is a radiography type mode.

The following is the menu part of the application.

```
#include <windows.h>

#include "vip_comm.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <fstream.h>


#define NUM_MENU_ENTRIES 11


void FluoroAcq(void);

void RadAutoDetectAcq(void);

void RadAutoDetectNFramesAcq(void);

void FluoroGainCal(void);

void RadGainCal(void);

void OffsetCalibration(void);

void GetImage(void);

void PutImage(void);

void GetModeDetails(void);

void GetSystemInfo(void);


void main(void)

{

    int choice = 0;
```

```
// display warnings
cout << "\n\nPlease note, this software assumes mode 0 is fluoro\n" << flush;
cout << "And mode 1 is rad\n" << flush;

while (choice != NUM_MENU_ENTRIES)
{
        // display menu
        cout << "\n\n\nPlease select from the following menu\n\n" << flush;
        printf("\t1. Fluoro Acquisition with software sync signals\n");
        printf("\t2. Rad Acquisition with auto-sense\n");
        printf("\t3. Rad Acquisition with auto-sense and N-frames\n");
        printf("\t4. Fluoro Gain Calibration\n");
        printf("\t5. Rad Gain Calibration\n");
        printf("\t6. Offset Calibration\n");
        printf("\t7. Retrieve Current Image\n");
        printf("\t8. Output Test Image\n");
        printf("\t9. Get Mode Details\n");
        printf("\t10. Get System Info\n");
        printf("\t%d. Exit\n\n\n", NUM_MENU_ENTRIES);
        cout << flush;

        cin >> choice;

        if (choice == 1)
                FluoroAcq();
        else if (choice == 2)
                RadAutoDetectAcq();
        else if (choice == 3)
                RadAutoDetectNFramesAcq();
        else if (choice == 4)
                FluoroGainCal();
        else if (choice == 5)
                RadGainCal();
        else if (choice == 6)
                OffsetCalibration();
        else if (choice == 7)
                GetImage();
                else if (choice == 8)
                        PutImage();
                else if (choice == 9)
                        GetModeDetails();
                else if (choice == 10)
                        GetSystemInfo();
        }
}
```

Acquisition

Acquisition can be performed several ways by combining software or hardware sync signals, auto-detect x-rays and/or collecting a predefined number of frames. This section will present a few examples

Fluoroscopy

Using Software Sync Signals

```
void FluoroAcq(void)
{
    int ret_val;
    // fluoroModeNum is the integer value for the mode number for a fluoro mode
    int fluoroModeNum = 0;

    // this call is needed to make a connection with the system
    ret_val = vip_open_link(VIP_ETHERNET_LINK, "111.111.11.111", "111.111.11.111");

    // select the appropriate mode
    ret_val = vip_select_mode(fluoroModeNum);

    // send prepare = true
    ret_val = vip_sw_handshaking(VIP_SW_PREPARE, 1);

    // send xrays = true
    ret_val = vip_sw_handshaking(VIP_SW_VALID_XRAYS, 1);

    int complete, numPulses, readyForPulse;
    int numFrames = 0;
    // this loop is an example of one way to terminate the acquisition
    while (numFrames < 100)
    {
            ret_val = vip_query_progress(&numFrames, &complete, &numPulses, &readyForPulse);
            // The ethernet can only be polled about once every second.
            // The following function is an example of a delay that might be used.
            SleepEx(1000, FALSE);
    }

    // send xrays = false
    ret_val = vip_sw_handshaking(VIP_SW_VALID_XRAYS, 0);

    // send prepare = false
    ret_val = vip_sw_handshaking(VIP_SW_PREPARE, 0);

    // end connection with the system
    ret_val = vip_close_link();
}
```

Radiography

Using Software Sync Signals

See Fluoroscopy.

Automatically Detect X-Rays On/Off

```
void RadAutoDetectAcq(void)
{
    int ret_val = 0;
    // radModeNum is the integer value for the mode number for a fluoro mode
    int radModeNum = 1;

    // this call is needed to make a connection with the system
    ret_val = vip_open_link(VIP_ETHERNET_LINK, "111.111.11.111", "111.111.11.111");

    // setup acquisition to be auto-sense for x-rays on and off
    ret_val = vip_set_mode_acq_type(radModeNum, VIP_AUTO_SENSE_ALL_FRAMES, 0);

    // select the appropriate mode
    ret_val = vip_select_mode(radModeNum);

    // send prepare = true
    ret_val = vip_sw_handshaking(VIP_SW_PREPARE, 1);

    int complete, numPulses, numFrames;
    int readyForPulse = 0;

    // this is one way to check when the system is ready for x-rays
    while (!readyForPulse)
    {
            ret_val = vip_query_progress(&numFrames, &complete, &numPulses, &readyForPulse);

            // The ethernet can only be polled about once every second.
            // The following function is an example of a delay that might be used.
            SleepEx(1000, FALSE);
    }

    // now the system is ready for x-rays, the following is a simple way of indicating this
    printf("\n\nREADY FOR X-RAYS\n\n");
    SleepEx(5000, FALSE);
    // at this time turn x-rays on/off accordingly

    // send prepare = false to end the acquisition
    ret_val = vip_sw_handshaking(VIP_SW_PREPARE, 0);

    // end connection with the system
    ret_val = vip_close_link();
}
```

Automatically Detect X-Rays; Stop when N Frames Collected

```
void RadAutoDetectNFramesAcq(void)
{
    int ret_val = 0;
    // radModeNum is the integer value for the mode number for a fluoro mode
    int radModeNum = 1;
```

```
// this call is needed to make a connection with the system
ret_val = vip_open_link(VIP_ETHERNET_LINK, "111.111.11.111", "111.111.11.111");

// setup acquisition to be auto-sense for x-rays on,
// but stop collecting frames when 1 frame is collected
ret_val = vip_set_mode_acq_type(radModeNum, VIP_AUTO_SENSE_N_FRAMES, 1);

// select the appropriate mode
ret_val = vip_select_mode(radModeNum);

// send prepare = true
ret_val = vip_sw_handshaking(VIP_SW_PREPARE, 1);

int complete, numPulses, numFrames = 0;
int readyForPulse = 0;

// this is one way to check when the system is ready for x-rays
while (!readyForPulse)
{
        ret_val = vip_query_progress(&numFrames, &complete, &numPulses, &readyForPulse);

        // The ethernet can only be polled about once every second.
        // The following function is an example of a delay that might be used.
        SleepEx(1000, FALSE);
}
// now the system is ready for x-rays, the following is a simple way of indicating this
printf("\n\nREADY FOR X-RAYS\n\n");
SleepEx(100, FALSE);
// at this time turn x-rays on accordingly

while (!complete)
{
        ret_val = vip_query_progress(&numFrames, &complete, &numPulses, &readyForPulse);

        // The ethernet can only be polled about once every second.
        // The following function is an example of a delay that might be used.
        SleepEx(500, FALSE);
        }

        // send prepare = false to end the acquisition
        ret_val = vip_sw_handshaking(VIP_SW_PREPARE, 0);

        // end connection with the system
        ret_val = vip_close_link();
    }
```

Gain Calibration

This section will describe how to send the basic commands to perform a gain calibration. Gain calibration for fluoroscopy requires a continuous beam during frame collection for gain. For radiography, the beam during frame collection for gain is pulsed.

Fluoroscopy

```
    void FluoroGainCal(void)
    {
        int ret_val;
        // fluoroModeNum is the integer value for the mode number for a fluoro mode
        int fluoroModeNum = 0;

        // this call is needed to make a connection with the system
        ret_val = vip_open_link(VIP_ETHERNET_LINK, "111.111.11.111", "111.111.11.111");

        // this command will ensure that any previous action is halted and the system is in an idle state
        ret_val = vip_reset_state();

        // set the number of frames to acquire during the acquisition
        ret_val = vip_set_num_cal_frames(fluoroModeNum, 64);

        // tell the system to prepare for a gain calibration
        ret_val = vip_gain_cal_prepare(fluoroModeNum);

        // send prepare = true
        ret_val = vip_sw_handshaking(VIP_SW_PREPARE, 1);

        // send xrays = true
    ret_val = vip_sw_handshaking(VIP_SW_VALID_XRAYS, 1);

    int numPulses , readyForPulse , complete = 0;
    int numFrames = 0;

    // accumulates the gain frames
    while (numFrames < 64)
    {
            ret_val = vip_query_progress(&numFrames, &complete, &numPulses, &readyForPulse);

    // The ethernet can only be polled about once every second.
    // The following function is an example of a delay that might be used.
            SleepEx(1000, FALSE);
    }

    // tells the user to turn off x-rays to accumulate the offset frames
    // the system will automatically collect these frames
    printf("Xrays OFF NOW!\n");

    // send xrays = false
    ret_val = vip_sw_handshaking(VIP_SW_VALID_XRAYS, 0);

    // wait for the calibration to complete
    while (!complete)
    {
            ret_val = vip_query_progress(&numFrames, &complete, &numPulses, &readyForPulse);

            // The ethernet can only be polled about once every second.
            // The following function is an example of a delay that might be used.
            SleepEx(1000, FALSE);
    }

    // end connection with the system
    ret_val = vip_close_link();
}
```

Radiography

```
void RadGainCal(void)
{
    int ret_val = 0;
    // radModeNum is the integer value for the mode number for a fluoro mode
    int radModeNum = 1;

    // this call is needed to make a connection with the system
    ret_val = vip_open_link(VIP_ETHERNET_LINK, "111.111.11.111", "111.111.11.111");

    int complete = 0;
    int numFrames = 0;
    int pulseCnt = 0;
    int readyForPulse = 0;

    // this command will ensure that any previous action is halted and the system is in an idle state
    ret_val = vip_reset_state();

    // set the number of frames to acquire during the acquisition
    ret_val = vip_set_num_cal_frames(radModeNum, 4);

    // tell the system to prepare for a gain calibration using auto detection of x-rays
    ret_val = vip_gain_cal_prepare(radModeNum, true);

    // send prepare = true
    ret_val = vip_sw_handshaking(VIP_SW_PREPARE, 1);

    // will acquire the offset frames first, NO X-RAYS
    while (numFrames < 4)
    {
            ret_val = vip_query_progress(&numFrames, &complete, &pulseCnt, &readyForPulse);

            // The ethernet can only be polled about once every second.
            // The following function is an example of a delay that might be used.
            SleepEx(1000, FALSE);
    }
```

```
// assume the user will collect 3 pulses
while (pulseCnt < 3)
{
        // reset the counter
        numFrames = 0;

        while (!readyForPulse)
        {
                ret_val = vip_query_progress(&numFrames, &complete, &pulseCnt, &readyForPulse);

                // The ethernet can only be polled about once every second.
                // The following function is an example of a delay that might be used.
                SleepEx(1000, FALSE);
        }

        // let user know the system is ready for x-rays
        printf("\n\nREADY FOR XRAYS\n\n");
        // turn x-rays on/off accordingly

        ret_val = vip_query_progress(&numFrames, &complete, &pulseCnt, &readyForPulse);

        // The ethernet can only be polled about once every second.
        // The following function is an example of a delay that might be used.
        SleepEx(1000, FALSE);
}

// send prepare = false
ret_val = vip_sw_handshaking(VIP_SW_PREPARE, 0);

// wait for the calibration to complete
while (!complete)
{
        ret_val = vip_query_progress(&numFrames, &complete, &pulseCnt, &readyForPulse);

        // The ethernet can only be polled about once every second.
        // The following function is an example of a delay that might be used.
        SleepEx(1000, FALSE);
```

```
    }

    // end connection with the system
    ret_val = vip_close_link();
}
```

Offset Calibration

This section will describe how to perform offset calibrations. The offset calibration procedure is the same for fluoroscopy and radiography. The offset calibration procedure is automatic, i.e. no sync signals are needed.

```
void OffsetCalibration(void)
{
    cout << "\n\nPlease note, this software only executes offset calibration\n";
    cout << "for Fluoroscopy mode\n\n" << flush;

    int ret_val;
    // fluoroModeNum is the integer value for the mode number for a fluoro mode
    int fluoroModeNum = 0;

    // this call is needed to make a connection with the system
    ret_val = vip_open_link(VIP_ETHERNET_LINK, "111.111.11.111", "111.111.11.111");

    int complete = 0;
    int numFrames, pulseCnt, readyForPulse;

    // this command will ensure that any previous action is halted and the system is in an idle state
    ret_val = vip_reset_state();

    // make sure the x-rays are off before performing an offset calibration
    printf("\n\nTurn X-rays OFF\n\n");
    SleepEx(3000, FALSE);

    // fluoroModeNum is the integer value for the mode number for a fluoro mode
    // send the command to tell the system to perform an offset calibration
    ret_val = vip_offset_cal(fluoroModeNum);

    // wait for the calibration to complete
    while (!complete)
    {
            ret_val = vip_query_progress(&numFrames, &complete, &pulseCnt, &readyForPulse);
            // The ethernet can only be polled about once every second.
            // The following function is an example of a delay that might be used.
            SleepEx(1000, FALSE);
    }

    // end connection with the system
    ret_val = vip_close_link();
}
```

Handling Images

These examples show how to retrieve/put a single type of image for fluoroscopy. To retrieve/put another type of image (i.e. defect map), simply change the image type in the command vip_get_image and handle file I/O accordingly. To retrieve a radiography image, change the image size and use the correct mode number. Please consult APPENDIX C for other image types that can be handled.

Retrieve Image

This example will retrieve the current fluoroscopy image (assuming mode 0 is fluoroscopy). This software will result in a file that is stored to disk. The file is called "GetImageTest.dat".

```
void GetImage(void)
{
    cout << "\n\nPlease note, this software only executes an image retrieval\n" << flush;
    cout << "for Fluoroscopy mode and only for the current image\n\n" << flush;

    int ret_val;
    // fluoroModeNum is the integer value for the mode number for a fluoro mode
    int fluoroModeNum = 0;

    // this call is needed to make a connection with the system
    ret_val = vip_open_link(VIP_ETHERNET_LINK, "111.111.11.111", "111.111.11.111");

    int x_size = 768; int y_size = 960;   int npixels = x_size * y_size;
    USHORT *image_ptr = (USHORT *)malloc(npixels * sizeof(USHORT));

    ret_val = vip_get_image(fluoroModeNum, VIP_CURRENT_IMAGE, x_size, y_size, image_ptr);

    char filename[32] = "GetImageTest.dat";
    // file on the host computer for storing the image
    FILE *finput = fopen(filename, "wb");
    if (finput == NULL)
    {
            printf("Error opening image file to put file.");
            exit(-1);
    }
    fwrite(image_ptr, sizeof(USHORT), npixels, finput);
    fclose(finput);
    free(image_ptr);

    // end connection with the system
    ret_val = vip_close_link();
}
```

Put Image

This example will cause the test image to be displayed for fluoroscopy mode (assuming mode 0 is fluoroscopy).

```
void PutImage(void)
{
    cout << "\n\nPlease note, this software only executes outputting\n" << flush;
    cout << "the Fluoroscopy test image\n\n" << flush;

    int ret_val;
    // fluoroModeNum is the integer value for the mode number for a fluoro mode
    int fluoroModeNum = 0;

    // this call is needed to make a connection with the system
    ret_val = vip_open_link(VIP_ETHERNET_LINK, "111.111.11.111", "111.111.11.111");

    int x_size = 768; int y_size = 960;

    ret_val = vip_put_image(fluoroModeNum, VIP_TEST_IMAGE, x_size, y_size, NULL);

    // end connection with the system
    ret_val = vip_close_link();
}
```

Retrieving System Data

The previous examples used hardcoded values for the mode number. The commands discussed in this section can be used to obtain the system or mode information in actual applications. Please consult the textual descriptions of the functions for more information about the data that is retrieved.

System Information

This command can be used to retrieve the number of modes and then the vip_get_mode_details command can be placed in a loop based on the number of modes to retrieve the information for each mode.

```
void GetSystemInfo()
{
    int ret_val;

    // this call is needed to make a connection with the system
    ret_val = vip_open_link(VIP_ETHERNET_LINK, "111.111.11.111", "111.111.11.111");

    int num_modes, default_mode_num;
    int max_lines_per_frame, max_columns_per_frame, max_pixel_value;
    int startup_configuration, video_used;
    int numAsics, receptorType;
    char system_description[32];

    ret_val = vip_get_system_info(&num_modes,
                                  &default_mode_num,
                                  &max_lines_per_frame,
                                  &max_columns_per_frame,
                                  &max_pixel_value,
                                  &video_used,
                                  system_description,
                                  &startup_configuration,
                                  &numAsics,
                                  &receptorType);

    printf("\nsystem data is:\n");
    printf("\nNumModes is %d", num_modes);
    printf("\nDefaultModeNum is %d", default_mode_num);
    printf("\nMaxLinesPerFrame is %d", max_lines_per_frame);
    printf("\nMaxColumnsPerFrame is %d", max_columns_per_frame);
    printf("\nVideo is %d", video_used);
    printf("\nstartupConfiguration is %d", startup_configuration);
    printf("\nnum asics is %d", numAsics);
    printf("\nreceptor type is %d", receptorType);
    printf("\nsystem description is %s\n\n\n", system_description);

    // end connection with the system
    ret_val = vip_close_link();
}
```

Mode Details

This example only retrieves the information for mode 0. The vip_get_system_info command can be used to retrieve the number of modes and then the vip_get_mode_details command can be called for each mode. The mode_type returned by vip_get_mode_details will describe the type of mode as fluoroscopy (value of 0) or radiography (value of 1) as described in Appendix E.

```
void GetModeDetails()
{
    int ret_val;

    // this call is needed to make a connection with the system
    ret_val = vip_open_link(VIP_ETHERNET_LINK, "132.190.17.149", "132.190.17.148");

    int mode_type, dcds_enable;
    double frame_rate, analog_gain;
    int lines_per_frame, columns_per_frame, lines_per_pixel, columns_per_pixel;
    char mode_description[32];

    ret_val = vip_get_mode_details(0,
                                  &mode_type,
                                  &frame_rate,
                                  &analog_gain,
                                  &lines_per_frame,
                                  &columns_per_frame,
                                  &lines_per_pixel,
                                  &columns_per_pixel,
                                  mode_description,
                                  &dcds_enable);
    printf("\n\nmode data is:\n");
    printf("\nModeType is %d", mode_type);
    printf("\nFrameRate is %f", frame_rate);
    printf("\nAnalogGain is %f", analog_gain);
    printf("\nHorizontalImageSize is %d", lines_per_frame);
    printf("\nVerticalImageSize is %d", columns_per_frame);
    printf("\nLinesPerPixel is %d", lines_per_pixel);
    printf("\nColumnsPerPixel is %d", columns_per_pixel);
    printf("\nmode description is %s", mode_description);
    printf("\nDCDS enable is %d\n\n\n", dcds_enable);

    // end connection with the system
    ret_val = vip_close_link();
}
```

IV. APPENDIXES

APPENDIX A: Error Codes Returned by the Command Processor

This table describes the meaning of the return values of the functions. 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. And return value of zero or VIP_NO_ERR implies that the function execution has successfully been complete.

No.	Error Code Name	Value	Description
1.	VIP_NO_ERR	0x0000	The normal return value for most functions.
2.	VIP_COMM_ERR	0x0001	Error occurred in the communications library, e.g., a message could not be sent because of a socket retry/timeout error.
3.	VIP_STATE_ERR	0x0002	A function was called which violated the rules of the server application's state machine, e.g., vip_get_image() was called while image acquisition was active.
4.	VIP_DATA_ERR	0x0004	One of the arguments of the function was invalid.
5.	VIP_SETUP_ERR	0x0020	An error was encountered involving the setup of a given mode.
6.	VIP_NO_CAL_ERR	0x0040	Attempted to activate a mode for which no valid calibration data exists.
7.	VIP_NO_IMAGE_ERR	0x0080	Used if vip_get_image() is called for an image type which cannot be retrieved.
8.	VIP_NOT_IMPL_ERR	0x4000	Used in development/debug.
9.	VIP_OTHER_ERR	0x8000	Some other error value, not otherwise specified.

APPENDIX B: File Name Strings Used in the Command Processor

No.	File Name	Description
1.	AuxDefectImage	Auxiliary defect map image file.
2.	BaseDefectImage	Base defect map image file.
3.	ConfigDataFile	Configuration data file.
4.	GainCalImage	Gain calibrated image file.
5.	*GcFirmwareLocal*	Global control firmware file. Reserved for future use.
6.	OffsetCalImage	Offset calibrated image file.
7.	*RcptFirmware*	Receptor firmware file. Reserved for future use.
8.	SystemDataFile	System data file. Reserved for future use.
9.	*VideoOutFirmware*	Video-out firmware file. Reserved for future use.

APPENDIX C: Image Types

This table describes the possible types of images, which may be transmitted between the host computer and the Command Processor via use of the vip_get_image() and vip_put_image() functions. All image types use 16-bit, unsigned integer pixels.

No.	Image Type Name	Value	Description
1.	VIP_CURRENT_IMAGE	0	This image resides in the accumulation/recursion buffer of the image processing system and can be summarized in the following formula: CURRENT_IMAGE = [(raw image data) – OFFSET_IMAGE] / GAIN_IMAGE; **Actions for vip_put_image():** The image is placed in output buffer memory and transmitted continuously out over the 16-bit digital interface. If the system has video display capability, the image is also displayed.
2.	VIP_OFFSET_IMAGE	1	This image resides in the offset calibration buffer. **Actions for vip_put_image():** The Command Processor replaces the offset calibration buffer currently in use. This only affects operation until the next offset calibration occurs for the given mode, or until the system is restarted, whichever comes first.

3.	VIP_GAIN_IMAGE	2	This image resides in the gain calibration buffer. It is not a raw flat field image, but is already offset-corrected. **Actions for vip_put_image():** The Command Processor replaces the gain file in permanent storage, as well as the gain calibration information currently in use, with the new image.
4.	VIP_BASE_DEFECT_IMAGE	3	This image resides in the base defect map buffer, which is 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. **Actions for vip_put_image():** The Command Processor replaces the base defect map file in permanent storage, as well as the base defect map information currently in use, with the new image.
5.	VIP_AUX_DEFECT_IMAGE	4	This image resides in the auxiliary defect map buffer, which is normally set up in manufacturing/final test. This map is automatically re-calculated every time a gain calibration is performed. A non-zero pixel value indicates the presence of a defect. **Actions for vip_put_image():** The Command Processor replaces the auxiliary defect map file in permanent storage, as well as the auxiliary defect map information currently in use, with the new image.
6.	VIP_TEST_IMAGE	5	This image is a progressive gray, test pattern image, generated by the command processor. This image type is not supported by vip_get_image(). To obtain the test pattern image, use vip_put_image() to place it in the current image buffer and then retrieve the current image with vip_get_image(). **Actions for vip_put_image():** When the function vip_put_image() is called, a test pattern image is displayed until the current mode or operation is changed. The x_size, y_size, and image_ptr arguments are ignored by vip_put_image().

APPENDIX D: Command Processor System Constants

No.	System Type Constants	Value	Description
Window Leveling Types:			
1.	VIP_WL_MAPPING_LINEAR	0	Window/level mapping function is linear.
2.	VIP_WL_MAPPING_NORM_ATAN	1	Window/level mapping function uses a normalized curve based on arctangent. In this case, "normalized" means that the a section of the arctangent curve around its inflection point is stretched vertically down and up so that the bottom of the window is mapped to a gray level of zero (or max gray if inverted) and the top of the window is mapped to the max gray level (or 0 if inverted).
3.	VIP_WL_MAPPING_CUSTOM	2	Window/level mapping function is customized.
Startup Configuration Types:			
4.	VIP_DISPL_TST_PTTRN_STRTUP	0	At startup, the Command Processor will display the test pattern and wait for commands from the host computer.
5.	VIP_STANDALONE_STRTUP	1	At startup, the Command Processor will attempt to detect X-rays on/off.
6.	VIP_ACTIVE_ACQ_STRTUP	2	At startup, the Command Processor will be in active for image acquisition.
Software Handshaking Types:			
7.	VIP_SW_PREPARE	0	Software-simulated as prepare signal.
8.	VIP_SW_VALID_XRAYS	1	Software-simulated as valid X-rays signal.
9.	VIP_SW_RADIATION_WARNING	2	Software-simulated as radiation warning signal.
1(	VIP_SW_RESET	3	Software-simulated as reset signal.

System Version Number Types:

11	VIP_MOTHERBOARD_VER	0	Version number of the motherboard.
12	VIP_SYS_SW_VER	1	Version number of the system software.
13	VIP_GLOBAL_CTRL_VER	2	Version number of the global control board.
14	VIP_GLOBAL_CTRL_FW_VER	3	Version number of the global control board firmware.
15	VIP_RECEPTOR_VER	4	Version number of the receptor.
16	VIP_RECEPTOR_FW_VER	5	Version number of the receptor firmware.
17	VIP_IPS_VER	6	Version number of the IPS board.
18	VIP_VIDEO_OUT_VER	7	Version number of the video-out board.
19	VIP_VIDEO_OUT_FW_VER	8	Version number of the video out board firmware.

APPENDIX E: Mode Type Constants

No.	Mode Type Constants	Value	Description
Acquisition Types:			
1.	VIP_ACQ_TYPE_CONTINUOUS	0	This constant indicates the acquisition type of continuous or fluoroscopy type.
2.	VIP_ACQ_TYPE_ACCUMULATION	1	The constant indicates the acquisition type of accumulation or radiography type.
Mode Acquisition Type:			
3.	VIP_INVALID_ACQ_MODE_TYPE	-1	Invalid acquisition mode type.
4.	VIP_VALID_XRAYS_N_FRAMES	0	Image acquisition starts when a valid X-rays signal is received and stops when n frames have been accumulated.
5.	VIP_VALID_XRAYS_ALL_FRAME S	1	Image acquisition starts when a valid X-rays signal is received and stops when all available frames have been accumulated. **Note that this acquisition mode type is not currently supported**. It is defined for future implementation.
6.	VIP_AUTO_SENSE_N_FRAMES	2	Image acquisition starts when a X-rays beam is detected and stops when n frames have been accumulated.
7.	VIP_AUTO_SENSE_ALL_FRAMES	3	Image acquisition starts when a X-rays beam is detected and stops when all available frames have been accumulated.
Radiography Scaling Types:			
8.	VIP_RAD_SCALE_NONE	0	No scaling will be performed by software.
9.	VIP_RAD_SCALE_UP	1	Only scaling up will be performed by software.
1(	VIP_RAD_SCALE_DOWN	2	Only scaling down will be performed by software.
1]	VIP_RAD_SCALE_BOTH	3	Scaling up or down will be performed by software where appropriate.

APPENDIX F: Communication Type Constants

No.	Mode Acquisition Type	Value	Description
1.	VIP_NO_LINK	-1	No communication link is established.
2.	VIP_ETHERNET_LINK	0	An Ethernet link is established.
3.	VIP_SERIAL_LINK	1	A serial link is established.
4.	VIP_COM1	0	Constant for com-port 1.
5.	VIP_COM2	1	Constant for com-port 2.
6.	VIP_COM3	2	Constant for com-port 3.
7.	VIP_COM4	3	Constant for com-port 4.

APPENDIX G: Error Status Bit Values

This table lists the error status bit codes returned by the function vip_query_error(). However, this function is not currently implemented and reserved for future revision.

No.	Bit	Hexadecimal Value	Description
1.	0	0000 0001	Error on CPU board or related component.
2.	1 - 7	0000 0002 – 0000 0080	Error on component related to CPU board (*TBD*).
3.	8	0000 0100	Error on Image Processor System (IPS) or related component.
4.	9 – 13	0000 0200 – 2000 0000	Error on component related to IPS (*TBD*).
5.	14	0000 4000	Error on Global Control board or related component.
6.	15 – 21	0000 8000 – 0020 0000	Error on component related to Global Control board (*TBD*).
7.	22	0040 0000	Error on ABS board or related component.
8.	23 – 25	0080 0000 – 0100 0000	Error on component related to ABS board (*TBD*).
9.	26 – 31	0200 0000 – 8000 0000	Reserved for future use.

V. SUPPLEMENTAL DOCUMENTS

Further information on how to use Ethernet and serial interface function can be found in *Serial Communication of VIP-9 Command Processor* under the file name *sio_cmds_usage.doc*.