VARIAN
medical systems

Serial Communication of VIP-9 System

Table of Contents

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

I. INTRODUCTION

The VIP-9 system can be controlled through the standard serial input/output ports that are readily available in most computers. The serial input/output capability of the VIP-9 system (VIP-9-SIO) is a secondary communication link to supplement the primary Ethernet communication link. VIP-9-SIO can be accessed directly through terminal applications such as Hyper-Terminal, a utility application of Microsoft Windows NT, or indirectly through a dynamic link library (DLL) *vip_comm.dll* that provides C function calls. This document describes these two communication methods and interpretations of the messages used by VIP-9-SIO.

FUNCTION INDEX

No.	Triad	Function Name	No.	Triad	Function Name
1.	AOC	vip_analog_offset_cal()	2.	CKL	vip_check_link()
3.	CLL	vip_close_link()	4.	SDC	vip_dcds_enable()
5.	EAC	vip_enable_auto_cal()	6.	ESH	vip_enable_sw_handshaking()
7.	GCP	vip_gain_cal_prepare()	8.	GAO	vip_get_analog_offset_data()
9.	GAS	vip_get_analog_offset_stats()	10.	GCS	vip_get_cal_stats()
11.	GCD	vip_get_config_data()	12.	GCR	vip_get_correction()
13.	GCM	vip_get_current_mode()	14.	GMG	vip_get_image()
15.	GLH	vip_get_lih()	16.	GMA	vip_get_mode_acq_type()
17.	GMD	vip_get_mode_details()	18.	GAF	vip_get_num_acq_frames()
19.	GCF	vip_get_num_cal_frames()	20.	GRS	vip_get_rad_scaling()
21.	GRF	vip_get_recursive_filter()	22.	GST	vip_get_self_test_log()
23.	GSI	vip_get_system_info()	24.	GSV	vip_get_system_version_numbers()
25.	GWL	vip_get_wl()	26.	OFC	vip_offset_cal()
27.	OPL	vip_open_link ()	28.	PCD	vip_put_config_data()
29.	PMG	vip_put_image()	30.	QER	vip_query_error()
31.	QPR	vip_query_progress()	32.	RSS	vip_reset_state()
33.	SLM	vip_select_mode()	34.	STT	vip_self_test()
35.	SAO	vip_set_analog_offset_data()	36.	SCR	vip_set_correction()
37.	SDB	vip_set_debug()	38.	SFR	vip_set_frame_rate()
39.	SLH	vip_set_lih()	40.	SMA	vip_set_mode_acq_type()
41.	SAF	vip_set_num_acq_frames()	42.	SCF	vip_set_num_cal_frames()
43.	SRS	vip_set_rad_scaling()	44.	SRF	vip_set_recursive_filter()
45.	SWL	vip_set_wl()	46.	SHS	vip_sw_handshaking()

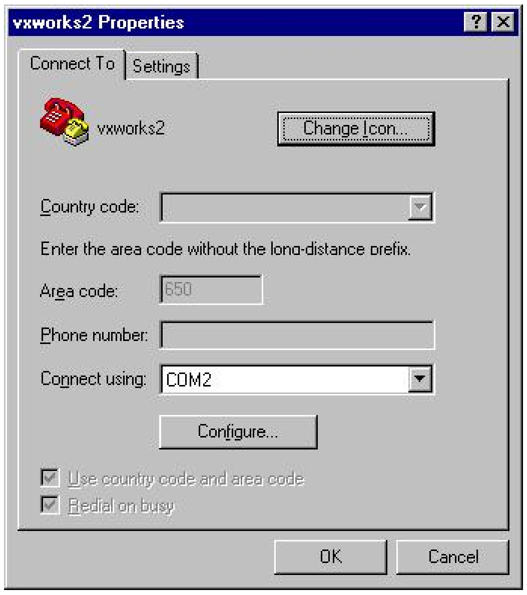
II. VIP-9-SIO COMMUNICATION USING A TERMINAL APPLICATION

VIP-9-SIO accepts ASCII text as a request message and replies with an ASCII text response message. Thus, a terminal application, that allows user to type in text for a request message and displays the response message from a communication device can be used to communicate with VIP-9-SIO.

The following serial communication settings are required for VIP-9-SIO:

Bits per second:	38,400
Data bits:	8
Parity:	None
Stop bit:	1

During development of VIP-9-SIO, the Hyper-Terminal application was used with the following settings to communicate with the VIP-9 command processor:


vxworks2 Properties

vxworks2

Country code:

Enter the area code without the long-distance prefix.

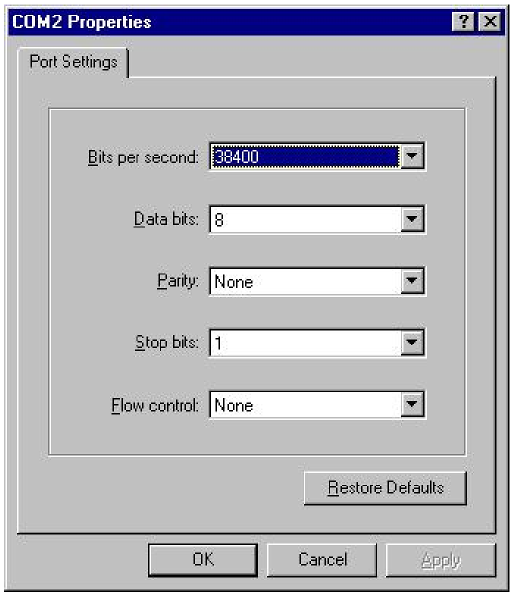
Area code: 650

Phone number:

Connect using: COM2

Use country code and area code

Redial on busy


COM2 Properties

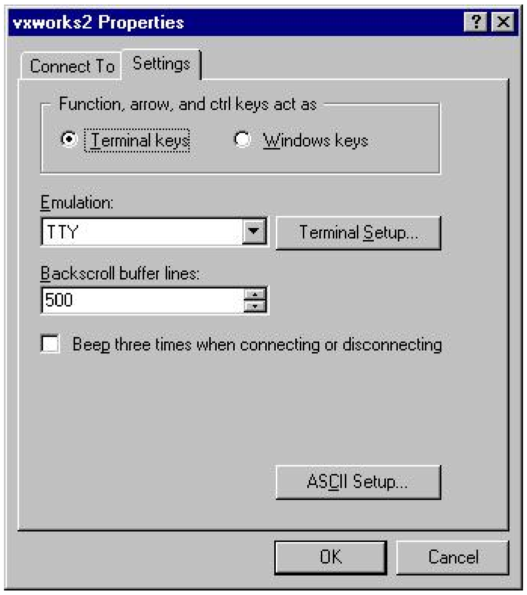
Bits per second: 38400

Data bits: 8

Parity: None

Stop bits: 1

Flow control: None


vxworks2 Properties

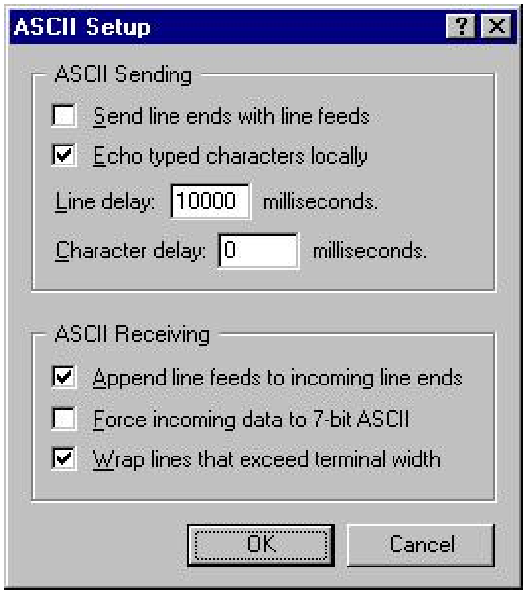
Connect To | Settings

Function, arrow, and ctrl keys act as: [✓ Terminal keys] Windows keys

Emulation: TTY

Backscroll buffer lines: 500

☐ Beep three times when connecting or disconnecting


ASCII Setup

ASCII Sending

☐ Send line ends with line feeds

☑ Echo typed characters locally

Line delay: 10000 milliseconds.

Character delay: 0 milliseconds.

ASCII Receiving

☑ Append line feeds to incoming line ends

☐ Force incoming data to 7-bit ASCII

☑ Wrap lines that exceed terminal width

With Hyper-Terminal, an ASCII text request message can be sent to VIP-9-SIO and an ASCII text response can be displayed on the Hyper-Terminal window.

For example, the user can enable automatic calibration by typing in **@EAC0;1;3600;60<CR>** where:

- @ denotes the beginning of the request,
- **EAC** denotes the enable-automatic-calibration function,
- 0 denotes mode 0 or fluoroscopy mode,
- the semicolons ';' separate the parameters,
- **1** denotes the integer value of the Boolean *true* (0 for *false*),
- 3600 denotes the minimum delay between calibrations in seconds,
- 60 denotes the post exposure delay in seconds,
- and **<CR>** denotes the character return and the end of the request message.

In a normal case, VIP-9-SIO responds to the above request with a the following message **@EAC0<CR>** where:

- @ denotes the beginning of the response,
- **EAC** denotes the enable-automatic-calibration function,
- 0 denotes the error code of 0 or no error,
- and **<CR>** denotes the character return and the end of the request message.

Not all parameters of a message can be easily recognized. For example, the response for the get mode details shown below requires some calculations so the user can understand its content.

```
@GMD0;1;7500;4000;1920;1536;1;1;1382114409;1869050465;1885894912;757
091951;191979172;0;301989889<CR>
```

The last eight integer parameters contain the text string for the description of the requested mode, which can be calculated as follows:

`1382114409 (DEC)`	`= 52-61-64-69 (HEX)`	`= R-a-d-i`
`1869050465 (DEC)`	`= 6f-67-72-61 (HEX)`	`= o-g-r-a`
`1885894912 (DEC)`	`= 70-68-79-00 (HEX)`	`= p-h-y-\0`

The string is terminated by the null character '\0'. Thus, no further calculations are necessary. The description reads "**Radiography**".

III. VIP-9-SIO COMMUNICATION USING THE VIP_*COMM.DLL*

The SIO DLL for Visual C++ called *vip_comm.dll* provides a more convenient way to communicate with VIP-9-SIO than using a terminal application. Within a Visual C++ program, the user can call SIO functions and retrieve data that is ready for use. First, the user needs to initialize and open a com-port by calling the following function:

```
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);
// Example: Open a serial link using COM PORT 2.
int error = vip_open_link(VIP_SERIAL_LINK, VIP_COM2);
```

Note that this function is also used to initialize and open the Ethernet link. To initiate a serial link, call the function with the following parameters: (1) the first parameter must be passed in as VIP_SERIAL_LINK, (2) the second and third parameters are unnecessary for a serial link. The user can simply ignore these parameters and (3) the next parameter is an available com-port between VIP_COM1 to VIP_COM4, which are defined with integers from 0 to 3, respectively.

Then the user can call specific functions to retrieve data from the VIP-9 system. For example, the enable automatic calibration function can be accomplished with the following function:

```
int vip_enable_auto_cal    (int mode_num, BOOL enable,
                            int minimum_delay,
                            int post_exposure_delay);
// An example of this function call is as follows:
int error = vip_enable_auto_cal(0, TRUE, 3600, 60);
```

All retrieved data will be formatted as indicated by the parameter type. For example, the vip_get_mode_details() function returns mode description as string of characters passed out through the parameter *mode_description* as written below:

```
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);
```

IV. VIP-9-SIO MESSAGE FORMAT

Interpretations of messages in serial communication between a host computer and VIP-9 target unit are based on the standard document Proposed American National Standard for Automated Vision Components – Cameras – AIA Serial Communication Protocol Specification. A message is transmitted as a stream of bytes. The values of the bytes and their orders determine the content of a message. This section presents how messages are structured in the VIP-9 serial communication.

VIP-9-SIO messages are transmitted in on of the following two formats:

1. *@AAA[n_1[;n_2...]]<CR>*

2. *@AAA^n_{error}<CR>*

where:

- *@* denotes the beginning of a message,
- *AAA* denotes a triad of three capital letters to identify a command such as CKL for check-link function or OPL for open-link function,
- *[n_1[;n_2...]]* denotes a sequence of zero to 25 integers of the range –2,147,483,648 to 2,147,483,647 as signed integers or of the range 0 to 4,294,967,295 as unsigned integers. More specifically, each integer n may contain up to ten digits with a minus sign or an optional plus sign,
- ^ denotes an error message,
- *n_{error}* denotes an error code, which is defined as a VIP error. *VIP_NO_ERR* has the value of zero. All other errors are defined as a unique integer in hex. For examples, the error code *VIP_COMM_ERR* has a value of 0001 (HEX) or 1 (DEC).
- *<CR>* denotes the end of a message.

VIP-9-SIO messages are transmitted in ASCII codes. Each byte is an individual character. Furthermore, spaces and commas may be used to improve readability of a message. These characters are eliminated before a message is to be processed. For example, the following two messages are considered identical:

```
1. @EAC0;1;3600;60<CR>

2. @EAC  0  ;    1;  3,600  ;  60    <CR>
```

Note that not all-valid characters for VIP-9-SIO messages can be printed on a monitor's screen such as the character return **<CR>** shown above.

For a complete listing of valid characters for VIP-9-SIO messages, please, see the table in APPENDIX A.

V. VIP9-SIO MESSAGE TRANSMISSION SEQUENCE

The message transmission sequence of a message occurs in three steps and is summarized in the following table:

No.	Text-Terminal/SIO DLL (Client)	Transmission Direction	VIP-9 Command Processor (Server)
1.	Send request message.	→	Receive request message.
	e.g.: **`@EAC0;1;3600;60<CR>`**		
2.	Receive acknowledge character.	←	Send acknowledge character.
			e.g.: **`<ACK>`** or **`<NAK>`**
3.	Receive response message.	←	Send response message.
			e.g.: **`@EAC0<CR>`** `or` **`@EAC^2<CR>`**

Note that the character **`<ACK>`** (ASCII value of 6) is sent to the client as an acknowledgement after the server receives a complete request message. Likewise, the character **`<NAK>`** (ASCII value of 21) is sent instead when the server recognizes that the request message is incomplete. These characters are not printable characters. Many terminal applications are not designed to display these characters.

For a complete listing of valid characters for VIP-9-SIO messages, please, see the table in APPENDIX A.

VI. AVAILABLE VIP-9-SIO MESSAGES

This section describes 41 request messages that VIP-9-SIO recognizes and the expected responses under normal operation.

Note that the VIP-9 command processor supports both Ethernet and serial links. However, it allows just one link to remain open at a time. Switching between these two links requires that the currently opened link be explicitly closed before the other link can be opened.

For detail descriptions of the parameters used for a function, please, see the corresponding description of the function in the document *communications_interface_design.doc*.

No.	Request/Response	Description
1.	**`@AOCn`$_a$`<CR>`**	AOC: *vip_analog_offset_cal()* request and parameter: **n**$_a$ = mode number.
	`@AOC0<CR>`	*vip_analog_offset_cal()* response and the error code of VIP_NO_ERR = 0.
2.	**`@CKL<CR>`**	CKL: *vip_check_link()* request.
	`@CKL0<CR>`	*vip_check_link()* response and the error code of VIP_NO_ERR = 0.
3.	**`@CLL<CR>`**	CLL: *vip_close_link()* request.
	`@CLL0<CR>`	*vip_close_link()* response and the error code of VIP_NO_ERR = 0.
4.	**`@SDCn`$_a$`<CR>`**	SDC: *vip_dcds_enable()* request and parameter: **n**$_a$ = enable (1 for TRUE and 0 for FALSE).
	`@SDC0<CR>`	*vip_dcds_enable()* response and the error code of VIP_NO_ERR = 0.
5.	**`@EACn`$_a$`;..;n`$_d$`<CR>`**	EAC: *vip_enable_auto_cal()* request and parameters: **n**$_a$ = mode number, **n**$_b$ = enable, **n**$_c$ = minimum delay and **n**$_d$ = post exposure delay.
	`@EAC0<CR>`	*vip_enable_auto_cal()* response and the error code of VIP_NO_ERR = 0.
6.	**`@ESHn`$_a$`<CR>`**	ESH: *vip_enable_sw_handshaking()* request and parameter: **n**$_a$ = enable (1 for TRUE or 0 for FALSE).
	`@ESH0<CR>`	*vip_enable_sw_handshaking()* response and the error code of VIP_NO_ERR = 0.
7.	**`@GCPn`$_a$`<CR>`**	GCP: *vip_gain_cal_prepare()* request and parameter: **n**$_a$ = mode number.
	`@GCP0<CR>`	*vip_gain_cal_prepare()* response and the error code of VIP_NO_ERR = 0.
8.	**`@GAOn`$_a$`<CR>`**	GAO: *vip_get_analog_offset_data()* request and parameter: **n**$_a$ = mode number.

	`@GAO0;`$\mathbf{n_1}$**`;..;`**$\mathbf{n_5}$**`<CR>`**	*vip_get_analog_offset_data()* response, the error code of VIP_NO_ERR = 0 and parameters: $\mathbf{n_1}$ = target value, $\mathbf{n_2}$ = tolerance, $\mathbf{n_3}$ = median percent, $\mathbf{n_4}$ = fractional iteration delta (divide this number by 1000 to obtain the actual floating value) and $\mathbf{n_5}$ = number of iterations.

9.	**`@GAS`**$\mathbf{n_a}$**`;`**$\mathbf{n_b}$**`<CR>`**	GAS: *vip_get_analog_offset_stats()* request and parameter: $\mathbf{n_a}$ = mode number and $\mathbf{n_b}$ = number of ASIC's.
	`@GAO^4<CR>`	**This command is not supported**. *vip_get_analog_offset_stats()* response and the error code of VIP_DATA_ERR = 4 or 0004 (HEX).
10.	**`@GCS`**$\mathbf{n_a}$**`<CR>`**	GCS: *vip_get_cal_stats()* request and parameter: $\mathbf{n_a}$ = mode number.
	`@GCS0;`$\mathbf{n_1}$**`;`**$\mathbf{n_2}$**`;`**$\mathbf{n_3}$**`<CR>`**	*vip_get_cal_stats()* response, the error code of VIP_NO_ERR = 0 and parameters: $\mathbf{n_1}$ = gain median, $\mathbf{n_2}$ = gain sigma (divide this number by 1000 to obtain the actual floating value) and $\mathbf{n_3}$ = offset median.
11.	**`@GCD`**$\mathbf{n_{na}}$**`<CR>`**	GCD: *vip_get_config_data()* request and parameter: $\mathbf{n_{na}}$ = not applicable (All parameters will be ignored). This command is not currently supported.
	`@GCD^16384<CR>`	**This command is not supported**. *vip_get_config_data()* response and the error code of VIP_NOT_IMPL_ERR = 16384 or 4000 (HEX).
12.	**`@GCR<CR>`**	GCR: *vip_get_correction()* request.
	`@GCR0;`$\mathbf{n_1}$**`;..;`**$\mathbf{n_4}$**`<CR>`**	*vip_get_correction()* response, the error code of VIP_NO_ERR = 0 and parameters: $\mathbf{n_1}$ = offset calibration (1 for TRUE and 0 for FALSE), $\mathbf{n_2}$ = gain calibration (1 for TRUE and 0 for FALSE), $\mathbf{n_3}$ = defect map (1 for TRUE and 0 for FALSE) and $\mathbf{n_4}$ = line noise (1 for TRUE and 0 for FALSE).
13.	**`@GCM<CR>`**	GCM: *vip_get_current_mode()* request.
	`@GCM0;`$\mathbf{n_1}$**`<CR>`**	*vip_get_current_mode()* response, the error code of VIP_NO_ERR = 0 and $\mathbf{n_1}$ = mode number.
14.	**`@GMG`**$\mathbf{n_{na}}$**`<CR>`**	GMG: *vip_get_image()* request and parameter: $\mathbf{n_{na}}$ = not applicable (All parameters will be ignored).
	`@GMG^16384<CR>`	**This command is not supported**. *vip_get_image()* response and the error code of VIP_NOT_IMPL_ERR = 16384 or 4000 (HEX).
15.	**`@GLH`**$\mathbf{n_a}$**`<CR>`**	GLH: *vip_get_lih()* request and parameter: $\mathbf{n_a}$ = mode number.
	`@GLH^16384<CR>`	**This command is not supported**. *vip_get_lih()* response and the error code of VIP_NOT_IMPL_ERR = 16384 or 4000 (HEX).

16.	**@GMAn_a<CR>**	GMA: *vip_get_mode_acq_type()* request and parameter: **n_a** = mode number.
	@GCR0;n_1;n_2<CR>	*vip_get_mode_acq_type()* response, the error code of VIP_NO_ERR = 0 and parameters: **n_1** = mode acquisition type and **n_2** = number of frames.

17.	**@GMDn_a<CR>**	GMD: *vip_get_mode_details()* request and parameter: **n_a** = mode number.
	@GMD0;n_1;..;n_{15}<CR>	*vip_get_mode_details()* response, the error code of VIP_NO_ERR = 0 and parameters: **n_1** = acquisition type, **n_2** = frame rate, **n_3** = analog gain, **n_4** = lines per frame, **n_5** = columns per frame, **n_6** = lines per pixel, **n_7** = columns per pixel, **n_8**...**n_{15}** = 32 characters for mode description and **n_{16}**=DCDS enable status.
18.	**@GAFn_a<CR>**	GAF: *vip_get_num_acq_frames()* request and parameter: **n_a** = mode number.
	@GAF0;n_1<CR>	*vip_get_num_acq_frames()* response, the error code of VIP_NO_ERR = 0 and parameter: **n_1** = number of acquisition frames.
19.	**@GCFn_a<CR>**	GCF: *vip_get_num_cal_frames()* request and parameter: **n_a** = mode number.
	@GCF0;n_1<CR>	*vip_get_num_acq_frames()* response, the error code of VIP_NO_ERR = 0 and parameter: **n_1** = number of calibrating frames.
20.	**@GRSn_a<CR>**	GRS: *vip_get_rad_scaling()* request and parameter: **n_a** = mode number.
	@GRS0;n_1;n_2<CR>	*vip_get_rad_scaling()* response, the error code of VIP_NO_ERR = 0 and parameter: **n_1** = scaling_type and **n_2** = target_value.
21.	**@GRFn_a<CR>**	GRF: *vip_get_recursive_filter()* request and parameter: **n_a** = mode number.
	@GRF0;n_1<CR>	*vip_get_recursive_filter()* response, the error code of VIP_NO_ERR = 0 and parameter: **n_1** = buffer weight (divide this number by 1000 to obtain the actual floating value).
22.	**@GST<CR>**	GST: *vip_get_self_test_log()* request.
	@GST^16384<CR>	**This command is not supported**. *vip_get_self_test_log()* response and the error code of VIP_NOT_IMPL_ERR = 16384 or 4000 (HEX).
23.	**@GSI<CR>**	GSI: *vip_get_system_info()* request.

	@GSI0;n_1;..;n_{16}<CR>	*vip_get_system_info()* response, the error code of VIP_NO_ERR = 0 and parameters: n_1 = number of modes, n_2 = default mode number, n_3 = maximum number of lines per frame, n_4 = maximum number of columns per frame, n_5 = maximum pixel value, n_6 = has video (1 for TRUE and 0 for FALSE), $n_7 \ldots n_{14}$ = 32 characters for system description, n_{15} = startup configuration, n_{16} = number of ASIC's and n_{16} = receptor type.

24.	**@GSVn_a<CR>**	GSV: *vip_get_system_version_numbers()* request and parameter: n_a = system version number type, which is one of the following values: VIP_MOTHERBOARD_VER = 0, VIP_SYS_SW_VER = 1, VIP_GLOBAL_CTRL_VER = 2, VIP_GLOBAL_CTRL_FW_VER = 3, VIP_RECEPTOR_VER = 4, VIP_RECEPTOR_FW_VER = 5, VIP_IPS_VER = 6, VIP_VIDEO_OUT_VER = 7, VIP_VIDEO_OUT_FW_VER = 8.
	@GSV0;n_1;..;n_9<CR>	*vip_get_system_version_numbers()* response, the error code of VIP_NO_ERR = 0 and parameters: $n_1 \ldots n_9$ = eight integers containing 32 characters for the returned version number.
25.	**@GWL<CR>**	GWL: *vip_get_wl()* request.
	@GWL0;n_1;n_2;n_3<CR>	**This command is not supported**. *vip_get_wl()* response, the error code of VIP_NO_ERR = 0 and n_1 = bottom value, n_2 = top value and n_3 = mapping (1 for TRUE and 0 for FALSE).
	@GWL^16384<CR>	Current *vip_get_wl()* response and the error code of VIP_NOT_IMPL_ERR = 16384 or 4000 (HEX).
26.	**@OFCn_a<CR>**	OFC: *vip_offset_cal()* request and parameter: n_a = mode number.
	@OFC0<CR>	*vip_offset_cal()* response and the error code of VIP_NO_ERR = 0.
27.	**@OPL<CR>**	OPL: *vip_open_link()* request.
	@OPL0<CR>	*vip_open_link()* response and the error code of VIP_NO_ERR = 0.
28.	**@PCDn_{na}<CR>**	PCD: *vip_put_config_data()* request and parameter: n_{na} = not applicable (All parameters will be ignored).
	@PCD^16384<CR>	**This command is not supported**. vip*_put_config_data()* response and the error code of VIP_NOT_IMPL_ERR = 16384 or 4000 (HEX).
29.	**@PMGn_{na}<CR>**	PMG: *vip_put_image()* request and parameter: n_{na} = not applicable (All parameters will be ignored).
	@PMG^16384<CR>	**This command is not supported**. vip*_put_image()* response and the error code of VIP_NOT_IMPL_ERR = 16384 or 4000 (HEX).
30.	**@QER<CR>**	QER: *vip_query_error()* request.

	@QER0;n_1<CR>	**This command is not supported**. *vip_query_error()* response, the error code of VIP_NO_ERR = 0 and parameter: **n_1** = error mask.
	@QER^16384<CR>	Current *vip_query_error()* response and the error code is VIP_NOT_IMPL_ERR = 16384 or 4000 (HEX).

31.	**`@QPR<CR>`**	QPR: *vip_query_progress()* request.
	`@QPR0;`$\mathbf{n_1}$**`..;`**$\mathbf{n_4}$**`<CR>`**	*vip_query_progress()* response, the error code of VIP_NO_ERR = 0 and parameters: $\mathbf{n_1}$ = number of frames completed, $\mathbf{n_2}$ = acquisition or calibration complete (1 for TRUE and 0 for FALSE), $\mathbf{n_3}$ = number of pulse or X-ray on/off sequences and $\mathbf{n_4}$ = ready for the next pulse.
32.	**`@RSS<CR>`**	RSS: *vip_reset_state()* request.
	`@RSS0<CR>`	*vip_reset_state()* response and the error code of VIP_NO_ERR = 0.
33.	**`@SLM`**$\mathbf{n_a}$**`<CR>`**	SLM: *vip_select_mode()* request and parameter: $\mathbf{n_a}$ = mode number.
	`@SLM0<CR>`	*vip_select_mode()* response and the error code of VIP_NO_ERR = 0.
34.	**`@STT<CR>`**	STT: *vip_self_test()* request.
	`@STT0<CR>`	**This command is currently implemented and reserved for future revision**. *vip_self_test()* response and the error code of VIP_NO_ERR = 0.
35.	**`@SAO`**$\mathbf{n_a}$**`;..;`**$\mathbf{n_f}$**`<CR>`**	SAO: *vip_set_analog_offset_data()* request and parameters: $\mathbf{n_a}$ = mode number, $\mathbf{n_b}$ = target value, $\mathbf{n_c}$ = tolerance, $\mathbf{n_d}$ = median percent, $\mathbf{n_e}$ = fractional iteration delta and $\mathbf{n_f}$ = number of iterations.
	`@SAO0<CR>`	*vip_set_analog_offset_data()* response and the error code of VIP_NO_ERR = 0.
36.	**`@SCR`**$\mathbf{n_a}$**`;..;`**$\mathbf{n_d}$**`<CR>`**	SCR: *vip_set_correction()* request and parameters: $\mathbf{n_a}$ = offset calibration (1 for TRUE and 0 for FALSE), $\mathbf{n_b}$ = gain calibration (1 for TRUE and 0 for FALSE), $\mathbf{n_c}$ = defect map (1 for TRUE and 0 for FALSE) and $\mathbf{n_d}$ = line noise (1 for TRUE and 0 for FALSE).
	`@SCR0<CR>`	*vip_set_correction()* response and the error code of VIP_NO_ERR = 0.
37.	**`@SDB`**$\mathbf{n_a}$**`<CR>`**	SDB: *vip_set_debug()* request and parameter: $\mathbf{n_a}$ = enable (1 for TRUE and 0 for FALSE).
	`@SDB0<CR>`	*vip_set_debug()* response and the error code of VIP_NO_ERR = 0.
38.	**`@SFR`**$\mathbf{n_a}$**`;`**$\mathbf{n_b}$**`<CR>`**	SFR: *vip_set_frame_rate()* request and parameter: $\mathbf{n_a}$ = mode number and $\mathbf{n_b}$ = frame rate * 1000. For modes with image size 768 x 960, valid frame rates are 7.5, 15.0 and 30.0 frames per second (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.
	`@SFR0<CR>`	*vip_set_frame_rate()* response and the error code of VIP_NO_ERR = 0.

39.	**@SLHn$_a$;n$_b$<CR>**	SLH: *vip_set_lih()* request and parameters: **n$_a$** = mode number and **n$_b$** = lih active (1 for TRUE and 0 for FALSE).
	@SLH0<CR>	*vip_set_lih()* response and the error code of VIP_NO_ERR = 0.
40.	**@SMAn$_a$;..;n$_c$<CR>**	SMA: *vip_set_mode_acq_type()* request and parameters: **n$_a$** = mode number, **n$_b$** = acquisition type and **n$_c$** = number of frames. The acquisition type must be one of the following constants: VIP_VALID_XRAYS_N_FRAMES = 0, VIP_VALID_XRAYS_ALL_FRAMES = 1 (reserved for future use), VIP_AUTO_SENSE_N_FRAMES = 2, VIP_AUTO_SENSE_ALL_FRAMES = 3.
	@SMA0<CR>	*vip_set_mode_acq_type()* response and the error code of VIP_NO_ERR = 0.
41.	**@SAFn$_a$;n$_b$<CR>**	SAF: *vip_set_num_acq_frames()* request and parameters: **n$_a$** = mode number and **n$_b$** = number of acquisition frames. The number of acquisition frames should have one of the following values -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.
	@SAF0<CR>	*vip_set_num_acq_frames()* response and the error code of VIP_NO_ERR = 0.
42.	**@SCFn$_a$;n$_b$<CR>**	SCF: *vip_set_num_cal_frames()* request and parameters: **n$_a$** = mode number and **n$_b$** = number of calibration frames. The number of calibrating frames should have a value in 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.
	@SCF0<CR>	*vip_set_num_cal_frames()* response and the error code of VIP_NO_ERR = 0.
43.	**@SRSn$_a$;n$_b$;n$_c$<CR>**	SRS: *vip_set_rad_scaling()* request and parameters: **n$_a$** = mode number, **n$_b$** = scaling type and **n$_c$** = target value. The scaling type must be one of the following constants: VIP_RAD_SCALE_NONE = 0, VIP_RAD_SCALE_UP = 1, VIP_RAD_SCALE_DOWN = 2 and VIP_RAD_SCALE_BOTH = 3.
	@SRS0<CR>	*vip_set_rad_scaling()* response and the error code of VIP_NO_ERR = 0.

44.	**@SRFn_a;n_b<CR>**	SRF: *vip_set_recursive_filter()* request and parameters: $\mathbf{n_a}$ = mode number and $\mathbf{n_b}$ = buffer weight.
	@SRF0<CR>	*vip_set_recursive_filter()* response and the error code of VIP_NO_ERR = 0.
45.	**@SWLn_a;n_b;n_c<CR>**	SWL: *vip_set_wl()* request and parameters: $\mathbf{n_a}$ = bottom value, $\mathbf{n_b}$ = top value and $\mathbf{n_c}$ = mapping. The parameter mapping must have one the following constants: VIP_WL_MAPPING_LINEAR = 0, VIP_WL_MAPPING_NORM_ATAN = 1 and VIP_WL_MAPPING_CUSTOM = 2.
	@SWL0<CR>	**This command is not supported**. *vip_set_wl()* response and the error code of VIP_NO_ERR = 0.
	@SWL^16384<CR>	Current *vip_get_wl()* response and the error code of VIP_NOT_IMPL_ERR = 16384 or 4000 (HEX).
46.	**@SHSn_a;n_b<CR>**	SHS: *vip_sw_handshaking()* request and parameters: $\mathbf{n_a}$ = signal type and $\mathbf{n_b}$ = active (1 for TRUE and 0 for FALSE).
	@SHS0<CR>	*vip_sw_handshaking()* response and the error code of VIP_NO_ERR = 0.

VII. APPENDICES

Appendix A - Valid Characters for VIP-9-SIO Messages

No.	Character	Dec	Hex	Description	Notes
1.	<NUL>	0	00	Null – takes up time without transmitting content.	<NUL> is also identified as <CTRL-@>.
2.	<ACK>	6	06	Message received complete.	<ACK> is also identified as <CTRL-F>.
3.	<CR>	13	0D	Last character in every message.	It also identified as <End of Message>
4.	<NAK>	21	15	Unrecognized message fragment received.	<NAK> is also identified as <CTRL-U>.
5.	<SPACE>	32	20	Space character – use for readability only, ignored.	
6.	+	43	2B	Plus sign – optional to indicate positive values.	
7.	,	44	2C	Comma – ignored, for readability.	
8.	-	45	2D	Minus sign – to indicate negative values.	
9.	0 – 9	48-57	30 - 39	Digits – represent values.	
10.	;	59	3B	Semicolon – separates values in multiple-value commands.	
11.	@	64	40	First character in all formatted messages.	It also identified as <New Message>.
12.	A-Z	65 – 90	41 – 5A	Used in triads to identify commands.	
13.	^	94	5E	Caret – error code indicator.	