SIEMENS

SINUMERIK 840D sl

Interface Specification VPLC I/O

Function Manual

Valid for

Control
SINUMERIK 840D sl

Software	*Version*
VPLC	04.04.00

05/2013

Legal information

Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

⚠ DANGER
indicates that death or severe personal injury **will** result if proper precautions are not taken.

⚠ WARNING
indicates that death or severe personal injury **may** result if proper precautions are not taken.

⚠ CAUTION
with a safety alert symbol, indicates that minor personal injury can result if proper precautions are not taken.

CAUTION
without a safety alert symbol, indicates that property damage can result if proper precautions are not taken.

NOTICE
indicates that an unintended result or situation can occur if the relevant information is not taken into account.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

Proper use of Siemens products

Note the following:

⚠ WARNING
Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be complied with. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Siemens AG
Industry Sector
Postfach 48 48
90026 NÜRNBERG
GERMANY

℗ 05/2013 Technical data subject to change

Preface

SINUMERIK documentation

The SINUMERIK documentation is organized in the following categories:

- General documentation
- User documentation
- Manufacturer/service documentation

Additional information

You can find information on the following topics under the link www.siemens.com/motioncontrol/docu:

- Ordering documentation/overview of documentation
- Additional links to download documents
- Using documentation online (find and search in manuals/information)

If you have any questions regarding the technical documentation (e.g. suggestions, corrections) then please send an e-mail to the following address: mailto:docu.motioncontrol@siemens.com

My Documentation Manager (MDM)

Under the following link you will find information to individually compile OEM-specific machine documentation based on the Siemens content: MDM www.siemens.com/mdm

Training

For information about the range of training courses, refer under:

- SITRAIN www.siemens.com/sitrain - training courses from Siemens for products, systems and solutions in automation technology
- SinuTrain www.siemens.com/sinutrain - training software for SINUMERIK

FAQs

You can find Frequently Asked Questions in the Service&Support pages Product Support www.siemens.com/automation/service&support

SINUMERIK

You can find information on SINUMERIK under the following link: www.siemens.com/sinumerik

Target group

This publication is intended for project engineers, programmers, technologists (of machine manufacturers), and system startup engineers (of systems/machines).

Benefits

The Function Manual describes the functions so that the target group is familiar with and can select them. It provides the target group with the information required to implement the functions.

Utilization phase: Planning and configuration phase, implementation phase, setup and commissioning phase

Standard version

Extensions or changes made by the machine manufacturer are documented by the machine manufacturer.

Other functions not described in this documentation might be executable in the control. This does not, however, represent an obligation to supply such functions with a new control or when servicing.

Further, for the sake of simplicity, this documentation does not contain all detailed information about all types of the product and cannot cover every conceivable case of installation, operation or maintenance.

Technical Support

You can find telephone numbers for other countries for technical support in the Internet under "Contact" www.siemens.com/automation/service&support.

SINUMERIK Internet address

http://www.siemens.com/sinumerik

Contents

1 Introduction

This document describes the interface to the VPLC (Virtual Programmable Logic Controller). Specifically, this document describes how clients may access VPLC I/O, monitor the VPLC status, and affect the VPLC mode (RUN/STOP).
A VPLC client is a software package that uses the VPLC interface. Clients may manage and/or use one or multiple VPLCs. Clients may consist of one or more processes, each interfacing to one or more VPLC instances.

The VPLC interface is delivered as a DLL (iVPLC.dll) that runs as part of the client's process, and a library (iVPLC.lib) for linking with standard Microsoft client projects. Other tool chains typically should dynamically load iVPLC.dll (LoadLibrary), and then individually address each public function (GetProcAddress) in the interface.

2 VPLC Access - LEDS & Switches

VPLC provides a functional interface that facilitates client access to VPLC operation. These interface functions are described in the following subsections.

VPLC provides an interface that facilitates simulation of PLC hardware switches and LED status indications. Using this interface, simulated switch states may be conveyed to the VPLC, and LED status indications may be queried by the client. Associated interface functions are described in the following subsections.

2.1 LEDS (status indications)

VPLC supports feedback of LED/status indications uniquely for each named VPLC and supported VPLC type. LED status indications are obtained according to the functions described in the following subsections.

2.1.1 vplc_get_leds()

```
<err> = vplc get leds(<name>,&<leds>,&<winerr>);
```

Returns the current states of the LED status indicators of the associated VPLC. where:

<**name**> (input) Data type: const char*

A pointer to a NULL terminated string specifying the name of the designated VPLC.

<**leds**> (output) Data type: **vplc_leds**

Indicates the state of each VPLC type specific status LED. Type **vplc_leds** contains the following fields:

leds_3172DP Data type: **VPLC_3172DP_LEDS**

Contains the status LED set for the VPLC 317 DP. Specifically …

run Data type: int
stop Data type: int
sf Data type: int
sf_dp Data type: int

Contains the states of the designated VPLC 317 DP status LED indicators. The possible states are the same for all LEDS. Possible states are:

LED_STATE_ON
LED_STATE_OFF
LED_FLASH_SLOW – 2Hz
LED_FLASH_FAST – 5Hz

<**winerr**> (output) Data type: int

Returns the Windows error that occurred during the VPLC shutdown, as indicated by <err>.

<**err**> (return) Data type: int
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered. The Windows error value is returned in <**winerr**>.
VPLC_NOT_STARTED	Indicates that the designated VPLC is not started.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.

2.1.2 vplc_watch_leds()

<**err**> = **vplc_watch_leds**(<**name**>,&<**leds**>,&<winerr>);
Registers a callback function with VPLC that VPLC calls each time that any LED status indicaton changes state. Note that with the registration of a watch LEDs callback function that the callback is called immediately upon its registration with the current status of the LEDs.
Where:

<**name**> (input) Data type: const char*
A pointer to a NULL terminated string specifying the name of the designated VPLC.

<**func**> (input) Data type: void (*(**func**)(<**leds**>)
A pointer to the LED status watch callback function. Note that a NULL value for <**func**> clears the most recently registered callback function. Callback parameters are:

<**leds**> (output) Data type: **vplc_leds**
Indicates the state of each VPLC type specific status LED. Type **vplc_leds** contains the following fields:

leds_3172DP Data type: **VPLC_3172DP_LEDS**
Contains the status LED set for the VPLC 317 DP. Specifically:

run	Data type: int
stop	Data type: int
sf	Data type: int
sf_dp	Data type: int

Contains the states of the designated VPLC 317 DP status LED indicators. The possible states are the same for all LEDS. Possible states are:

- **LED_STATE_ON**
- **LED_STATE_OFF**
- **LED_FLASH_SLOW** – 2Hz
- **LED_FLASH_FAST** – 5Hz

<**err**> (return) Data type: int
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered. The Windows error value is returned in <**winerr**>.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.

2.2 Hardware Switches

VPLC supports emulation of PLC hardware switches uniquely for each named VPLC and supported VPLC type. Simulated hardware switch states are conveyed to, and read from, VPLC by invoking the associated functions described in the following sections.

2.2.1 vplc_set_switch()

```
<err> = vplc_set_switch(<name>,&< v_switch >,&<winerr>);
```

Notifies the VPLC of the state of the designated switch.
See section 2.1 above.
Where:

`<name>` `(input)` `Data type: const char*`
A pointer to a NULL terminated string specifying the name of the designated VPLC.

`<v_switch>` `(input)` `Data type: const` **`vplc_switches`**
Identifies the VPLC type specific switches. Type **vplc_switches** contains the following fields:

`switches_3172DP` `Data type: enum` **`VPLC_3172DP_SWITCHES`**
Contains an enumeration of the simulated hardware switch configuration for the VPLC 317 DP. The following switch states are supported for VPLC type 3172DP:

run	Function returns immediately after switch state is set.
runp	Function returns immediately after switch state is set.
stop	Function returns immediately after switch state is set.
mres	Function does not return until after mres operation is complete.
urloeschen	Function does not return until after urloeschen operation is complete.

`<err>` `(return)` `Data type: int`
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered. The Windows error value is returned in <**winerr**>.
VPLC_NOT_STARTED	Indicates that the designated VPLC is not started.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.

Note that the return of **vplc_set_switch()** may or may not be synchronous with any subsequent VPLC response to a switch state change, e.g., the VPLC has not necessarily switched to the run state, after setting the **run** switch **ON**, upon the return of **vplc_set_switch()**. The run LED status indicator should be monitored to determine the timing of the actual state change. However, in the case of an **mres**, the **mres** operation is complete upon the return of **vplc_set_switch()**.

2.2.2 vplc_get_switch()

```
<err> = vplc_get_switch(<name>,&< v_switch >,&<winerr>);
```

Returns to VPLC the currently **ON** switch. Note that all switch states are mutually exclusive, i.e., only one can be **ON** at a given time, but also, that at least one switch state is always ON.
See section 2.1 above.

`<name>` (input) Data type: const char*
A pointer to a NULL terminated string specifying the name of the designated VPLC.

`<v_switch>` (input) Data type: vplc_switches
Returns the VPLC type specific switch that is currently set. Type **vplc_switches** contains the following fields:

switches_3172DP Data type: enum **VPLC_3172DP_SWITCHES**
Contains an enumeration of the simulated hardware switch configuration for the VPLC 317 DP. The following switch states are supported for VPLC type 3172DP:

run	Indicates that the **run** switch state is currently set.
runp	Indicates that the **runp** switch state is currently set.
stop	Indicates that the **stop** switch state is currently set.
mres	Indicates that the **mres** switch state is currently set.
urloeschen	Indicates that the **urloeschen** switch state is currently set.

`<err>` (return) Data type: int
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered. The Windows error value is returned in **<winerr>**.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.

2.2.3 vplc_watch_switch()

<err> = vplc_watch_switch(<name>,<func>,&<winerr>);

Registers a callback function with VPLC that VPLC calls each time that the run/stop/reset switch state is changed.
Note that the switch states are mutually exclusive, i.e., only one can be active at a given time, but also, that at least one switch state is always active.
See section 2.1 above.
Note also that with the registration of a watch switch callback function that the callback is called immediately upon its registration with the current state of the switch.
Where:

<**name**> (input) Data type: const char*

A pointer to a NULL terminated string specifying the name of the designated VPLC.

<**func**> (input) Data type: void *(func)(<**switch_**>)

A pointer to the switch changed callback function. Note that a NULL value for <func> clears the most recently registered callback function. Callback parameters are:

<**switch_**> (output) Data type: union vplc_switchess

Indicates the state of the VPLC run/stop/reset switch. Type vplc_switches contains the following fields:

switches_3172DP Data type: enum

Contains an enumeration of the simulated hardware switch configuration for the VPLC 317 DP. The following switches states are supported for VPLC type 3172DP:

run	Indicates that the **run** switch state is currently set.
runp	Indicates that the **runp** switch state is currently set.
stop	Indicates that the **stop** switch state is currently set.
mres	Indicates that the **mres** switch state is currently set.
urloeschen	Indicates that the **urloeschen** switch state is currently set.

<**err**> (return) Data type: int

Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered. The Windows error value is returned in <**winerr**>.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.

3 VPLC I/O Synchronization & Simulation

VPLC provides an interface that allows client application software to emulate real I/O. This requires the client simulation process to be able to synchronize with the VPLC scan. In this mode, VPLC suspends each scan at its start and notifies the client application of the scan suspended state. This allows the client application to perform any necessary I/O manipulation, as described in section 3.2 below.
After completion of the I/O manipulation, the client application may then command the VPLC to resume its scan, thus processing the affected I/O synchronously with the simulation. See figure 3-1 below to see the sequence of operations performed to accomplish the I/O simulation.

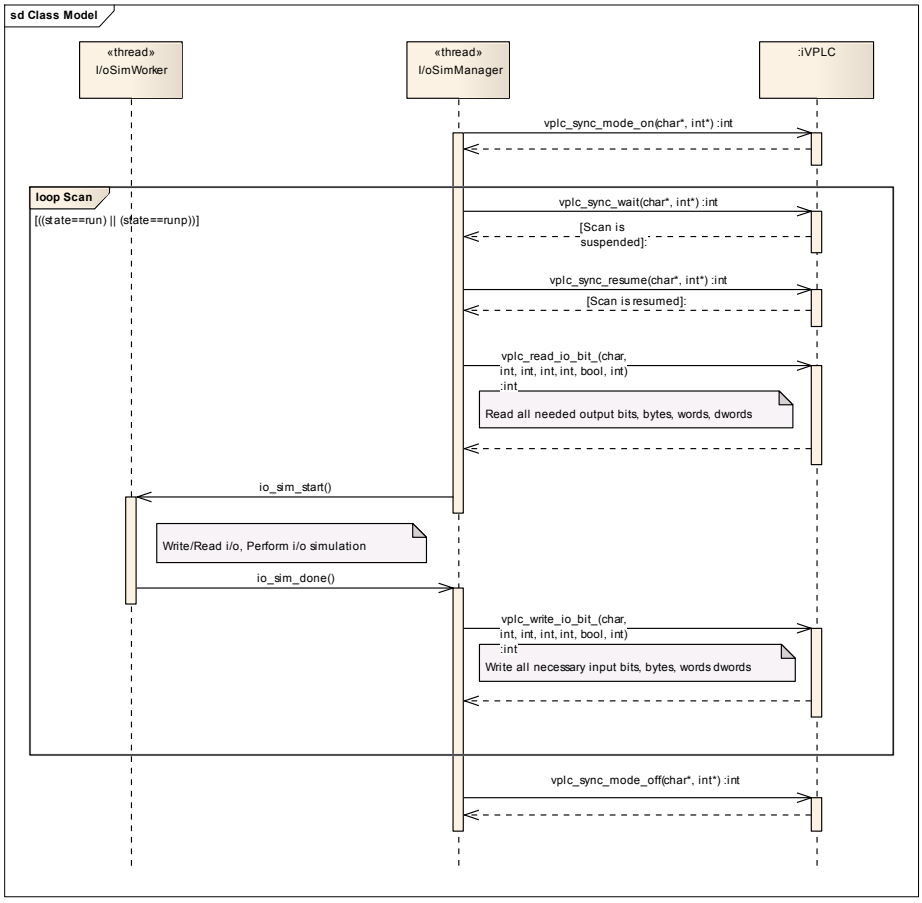

Figure 3-1: - I/O Simulation in Sync Mode Sequence of Functions

VPLC provides functions to manage I/O simulation activity. They are described as follows.

3.1 Synchronization

VPLC provides functions that allow client simulation application processes to synchronize with the VPLC scan. These functions are described in the following subsections.

3.1.1 vplc_reg_io_xchg_done()

Function **vplc_reg_io_xchg_done()** registers a client application function for callback at the completion of I/O exchange for each scan.
vplc_reg_io_xchg_done() invocation is:

```
<err> = vplc_reg_io_xchg_done(<name>,<func>,&<winerr>)
```

Registers the client application's I/O exchange done callback function for the specified VPLC instance.
Where:

```
<name>      (input)     Data type: const char*
```

A pointer to a NULL terminated string specifying the name of the designated VPLC.

```
<func>      (input)  Data type: void ((*(func))(<name>))
```

A pointer to the async mode's I/O exchange done callback function. A NULL pointer deletes the registration of any previously registered callback function. Callback function parameters are:

```
<name>      (input)           Data type: char*
```

A pointer to a NULL terminated string specifying the name of the designated VPLC.

```
<err>       (return)    Data type: int
```

Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered. The Windows error value is returned in **<winerr>**.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.

Note that the I/O exchange done callback function remains registered until the VPLC instance is terminated, or until its registration is deleted as described above.

3.1.2 vplc_sync_mode_on()

Function **vplc_sync_mode_on()** commands the VPLC to sync with the client's I/O simulation process. **vplc_sync_mode_on()** invocation is:

```
<err> = vplc_sync_mode_on(<name>,&<winerr>);
```

Commands the VPLC into sync mode.
Where:

<**name**> (input) Data type: const char*
A pointer to a NULL terminated string specifying the name of the associated VPLC.

<**winerr**> (output) Data type: int
Returns the Windows error that occurred during the operation as indicated by **<err>**.

<**err**> (return) Data type: int
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.

3.1.3 vplc_sync_wait()

Function **vplc_sync_wait()** suspends the calling process until the VPLC scan is reported by VPLC as being suspended at its start. Client software may, upon the return of **vplc_sync_wait()** perform I/O simulation and other VPLC scan synchronous activities. **vplc_sync_wait()** invocation is:

```
<err> = vplc_sync_wait(<name>,&<winerr>);
```

Commands the VPLC into sync mode.
Where:

`<name>` `(input)` `Data type: const char*`
A pointer to a NULL terminated string representing the name of the associated VPLC.

`<winerr>` `(output)` `Data type: int`
Returns the Windows error that occurred during the operation as indicated by **<err>**.

`<err>` `(return)` `Data type: int`
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered. The Windows error value is returned in <winerr>.
VPLC _SCAN_STOPPED	Indicates that the VPLC transitioned to STOP mode. Client software should either call **vplc_sync_mode_off()** if they want to terminate synchronization, or **vplc_sync_wait()** if they want to continue synchronization with the next VPLC scan, if the VPLC transitions back to RUN mode. Note that the client can cancel **vplc_sync_wait()** by calling **vplc_sync_mode_off()**.
VPLC_SHUTDOWN	Indicates that the VPLC process terminated or that the VPLC was shutdown. If the client simulation software intends to continue simulation at the next start up of VPLC, then it should call **vplc_sync_mode_off()**, **vplc_sync_mode_on()**, **followed by vplc_sync_wait()**. This will allow synchronization with the first VPLC scan upon VPLC start up.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.

3.1.4 vplc_sync_resume()

Function **vplc_sync_resume()** resumes the VPLC scan when suspended. Client software should resume the VPLC scan after performing I/O simulation and other VPLC scan synchronous activities. **vplc_sync_resume()** invocation is:

```
<err> = vplc_sync_resume(<name>,&<winerr>);
```

Commands the VPLC into sync mode.
Where:

<**name**> (input) Data type: const char*
A pointer to a NULL terminated string representing the name of the associated VPLC.

<**winerr**> (output) Data type: int
Returns the Windows error that occurred during the operation as indicated by **<err>**.

<**err**> (return) Data type: int
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_NOT_STARTED	Indicates that the designated VPLC is not started.
VPLC_SCAN_STOPPED	Indicates that the designated VPLC is not in RUN mode, and its scan cannot be resumed. To synchronize with the VPLC scan upon its transition to RUN mode, call **vplc_sync_wait()**.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.

3.1.5 vplc_sync_mode_off()

Function **vplc_sync_mode_off()** commands the VPLC to discontinue syncing with the clients I/O simulation process. **vplc_sync_mode_off()** invocation is:

```
<err> = vplc_sync_resume(<name>,&<winerr>);
```

Commands the specified VPLC to discontinue scan synchronization and resumes its scan if it is suspended. vplc_sync_mode_off() also cancels any pending vplc_sync_wait() associated with the named VPLC and client.
Where:

<**name**> (input) Data type: const char*
A pointer to a NULL terminated string representing the name of the associated VPLC.

<**winerr**> (output) Data type: int
Returns the Windows error that occurred during the operation as indicated by **<err>**.

<**err**> (return) Data type: int
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.

3.2 DP I/O Configuration Interface

VPLC provides the client application a means of obtaining notification of the presence of a DP I/O hardware configuration. VPLC provides a function that allows the client application to obtain a definition of the existing I/O configuration, and register a callback function that is called in the event of a change in the I/O configuration. VPLC also provides functions that facilitate reading and writing of VPLC I/O. These functions are specified in the following sections.

3.2.1 vplc_get_hw_config()

VPLC function **vplc_get_hw_config()** returns to the client application the current I/O hardware configuration, if one currently exists, and provides VPLC a means of notifying the client of any I/O configuration change. **vplc_get_hw_config()** invocation is:

```
<err> =
vplc_get_hw_config(<name>,&<config>,<func>,&<winerr>);
```

Returns the current DP I/O hardware configuration, and registers the associated callback function.
Where:

`<name>` `(input)` `Data type: const char*`
A pointer to a NULL terminated string representing the name of the associated VPLC instance.

`<config>` `(output)` `Data type:` **`vplc_io_desc_type*`**
A pointer to a **vplc_io_desc_type** structure that returns to the caller the DP I/O description as described in section 3.2.20 below.

`<func>` `(input)` `Data type: void ((*(func))(<name>,<*io_desc>))`
A pointer to a function that enables the VPLC to notify the client application>,<*io of a change in the DP I/O configuration. Callback function parameters are:

`<name>` `(input)` `Data type: char*`
A pointer to a NULL terminated string specifying the name of the designated VPLC.

`<io_desc>` `(input)` `Data type:` **`vplc_io_desc_type*`**
A pointer to a **vplc_io_desc_type** structure that returns to the caller the DP I/O description as described in section 3.2.20 below.

`<err>` `(return)` `Data type: int`
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_NO_SHARED_MEMORY	Indicates that no hardware configuration is present, but note that the callback function is still registered if specified.
VPLC_INV_NAME	Indicates that the name VPLC does not exist.

Function **vplc_get_hw_config()** may be called anytime after the VPLC is started.

3.2.2 vplc_test_io_state()

Function **vplc_test_io_state()** tests to see if the VPLC I/O region is currently valid, or not. The I/O region is considered invalid if the VPLC is not in RUN mode, or if a hardware configuration has not been loaded by the VPLC.
vplc_test_io_state() invocation is:

```
<err> = vplc_test_io_state(<name>,<func>,&<winerr>);
```

Determines the validity of the VPLC's I/O region and returns the appropriate status.
where:

`<name>` `(input)` `Data type: const char*`
A pointer to a NULL terminated string specifying the name of the designated VPLC.

`<err>` `(return)` `Data type: int`
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_INV_NAME	Indicates that the named VPLC does not exist
VPLC_NO_SHARED_MEMORY	Indicates that no hardware configuration is present, but note that the callback function is still registered if specified.
VPLC_IO_NOT_RDY	Indicates that the I/O is not valid at this time because the PLC is not in RUN mode.

This function is only necessary if attempting to access I/O directly through information contained in **vplc_io_desc_type**.

3.2.3 vplc_write_io_bit()

VPLC function **vplc_write_io_bit()** writes the specified state to the designated I/O location. **vplc_write_io_bit()** invocation is:

```
<err> =
vplc_write_io_bit(<name>,<area>,<adr>,<pos>,<state>,&<winerr
>);
```

Writes the specified I/O bit to the designated state.
Where:

<**name**> (input) Data type: const char*
A pointer to a NULL terminated string representing the name of the associated VPLC instance.

<**area**> (input) Data type: int
Designates the target I/O area. Valid specifications are:

VPLC_I	Specifies process image inputs.
VPLC_Q	Specifies process image outputs
VPLC_PI	Specifies peripheral inputs.
VPLC_PQ	Specifies peripheral outputs

<**adr**> (input) Data type: int
Designates the target I/O address as a byte offset from the start (byte 0) of the area. For <**area**>s **VPLC_I** and **VPLC_Q** (process image), valid addresses are: (0<=**adr**<=255). For <**area**>s **VPLC_PI** and **VPLC_PQ** (peripheral), valid addresses are: (0<=**adr**<=8191), but are dependent upon the number of I/O modules available. An out of range address specification generates a return error code.

<**pos**> (input) Data type: int
Contains the target bit position within the specified I/O <**adr**>: 0<=<**pos**><=7.

<**state**> (input) Data type: bool
Specifies the new bit state, either on (1) or off (0).

<**winerr**> (output) Data type: int
Returns the OS error that occurred during the operation as indicated by <**err**>.

<**err**> (return) Data type: int
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.
VPLC_INV_AREA	Indicates that the specified I/O <**area**> is not valid.
VPLC_INV_ADR	Indicates that the specified I/O <**adr**> exceeds the maximum range supported by the PLC, or exceeds the address range of available I/O modules.

VPLC_NO_SHARED_MEMORY	Indicates that no hardware configuration is present, but note that the callback function is still registered if specified.
VPLC_INV_POS	Indicates that the specified bit position is out of range.
VPLC_IO_NOT_RDY	Indicates that the I/O is not valid at this time because the PLC is not in RUN mode.

When a PQ (peripheral) value is forced to a particular state, if applicable, the associated Q (process image) value is also forced to that same state.

Function **vplc_write_io_bit()** should only be called while the system I/O is valid as indicated by the callback function registered by **vplc_test_io_state()**, or indicated by the **plc_in_run_mode** member of **vplc_io_desc_type** as described in section 3.2.20 below.

3.2.4 vplc_write_io_bit_()

VPLC function **vplc_write_io_bit()** writes the specified state to the designated I/O location.
vplc_write_io_bit_() is the same as **vplc_write_io_bit()** except that it specifies that I/O be addressed by slave/slot instead of by a logical address, and that the VPLC process image cannot be directly accessed.
vplc_write_io_bit_() invocation is:

```
<err> =
vplc_write_io_bit_(<name>,<area>,<slave>,<slot>,<offset>,<pos>,<state>,&<winerr>);
```

Writes the specified I/O bit to the designated state.
Where:

`<name>` (input) Data type: const char*
A pointer to a NULL terminated string representing the name of the associated VPLC instance.

`<area>` (input) Data type: int
Designates the target I/O area. Valid specifications are:
VPLC_PI Specifies peripheral inputs.
VPLC_PQ Specifies peripheral outputs.

`<slave>` (input) Data type: int
Designates I/O slave number: 1<=<**slave**><=126. An out of range address specification generates a return error code.

`<slot>` (input) Data type: int
Designates I/O slot number: 0<=<**slot**><=247. An out of range address specification generates a return error code.

`<offset>` (input) Data type: int
Designates the byte offset from the module base address. This value is dependent upon the module's I/O length. Possible values are:

0	module I/O length of 1.
0,1	module I/O length of 2.
0,1,2,3	module I/O length of 4.

<**pos**> (input) Data type: int
Contains the target bit position within the specified I/O address:
0<=<**pos**><=7.

<**state**> (input) Data type: bool
Specifies the new bit state, either on (1) or off (0).

<**winerr**> (output) Data type: int
Returns the OS error that occurred during the operation as indicated by <**err**>.

<**err**> (return) Data type: int
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_INV_NAME	Indicates that the named VPLC does not exist
VPLC_INV_AREA	Indicates that the specified I/O <area> is not valid.
VPLC_INV_SLAVE	Indicates that the specified I/O <slave> number exceeds the maximum range supported by the PLC.
VPLC_INV_SLOT	Indicates that the specified I/O <slot> number exceeds the maximum range supported by the PLC.
VPLC_NO_SHARED_MEMORY	Indicates that no hardware configuration is present, but note that the callback function is still registered if specified.
VPLC_INV_POS	Indicates that the specified bit position is out of range.
VPLC_IO_NOT_RDY	Indicates that the I/O is not valid at this time because the PLC is not in RUN mode.

When a PQ (peripheral) value is forced to a particular state, if applicable, the associated Q (process image) value is also forced to that same state.

Function **vplc_write_io_bit_()** should only be called while the system I/O is valid as indicated by the callback function registered by **vplc_test_io_state()**, or indicated by the **plc_in_run_mode** member of **vplc_io_desc_type** as described in section 3.2.20 below.

3.2.5 vplc_write_io_byte()

VPLC function **vplc_write_io_byte()** writes the specified byte value to the designated I/O location. **vplc_write_io_byte()** invocation is:

```
<err> =
vplc_write_io_byte(<name>,<area>,<adr>,<value>,&<winerr>);
```

Writes the specified byte value to the designated I/O location.
Where:

<**name**> (input) Data type: const char*
A pointer to a NULL terminated string representing the name of the associated VPLC instance.

<**area**> (input) Data type: int
Designates the target I/O area. Valid specifications are:

VPLC_I	Specifies process image inputs.
VPLC_Q	Specifies process image outputs.
VPLC_PI	Specifies peripheral inputs.
VPLC_PQ	Specifies peripheral outputs.

<**adr**> (input) Data type: int
Designates the target I/O address as a byte offset from the start (byte 0) of the area. For <**area**>s **VPLC_I** and **VPLC_Q** (process image), valid addresses are: (0<=**adr**<=255). For <**area**>s **VPLC_PI** and **VPLC_PQ** (peripheral), valid addresses are: (0<=**adr**<=8191), but are dependant upon the number of I/O modules available. An out of range address specification generates a return error code.

<**value**> (input) Data type: unsigned char
Contains the I/O byte value to be written.

<**winerr**> (output) Data type: int
Returns the OS error that occurred during the operation as indicated by <err>.

<**err**> (return) Data type: int
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.
VPLC_INV_AREA	Indicates that the specified I/O <**area**> is not valid.
VPLC_INV_ADR	Indicates that the specified I/O <**adr**> exceeds the maximum range supported by the PLC, or exceeds the address range of available I/O modules
VPLC_NO_SHARED_MEMORY	Indicates that no hardware configuration is present, but note that the callback function is still registered if specified.
VPLC_IO_NOT_RDY	Indicates that the I/O is not valid at this time because the PLC is not in RUN mode.

When a PQ (peripheral) value is forced to a particular state, if applicable, the associated Q (process image) value is also forced to that same state.

Function **vplc_write_io_byte()** should only be called while the system I/O is valid as indicated by the callback function registered by **vplc_test_io_state()**, or indicated by the **plc_in_run_mode** member of **vplc_io_desc_type** as described in section 3.2.20 below.

3.2.6 vplc_write_io_byte_()

VPLC function **vplc_write_io_byte_()** writes the specified byte value to the designated I/O location. **vplc_write_io_byte_()** is the same as **vplc_write_io_byte()** except that it specifies that I/O be addressed by slave/slot instead of by a logical address, and that the VPLC process image cannot be directly accessed. vplc_write_io_byte_() invocation is:

<**err**> =
vplc_write_io_byte_(<**name**>,<**area**>,<**slave**>,<**slot**>,<**offset**>,<**value**>,&<**winerr**>);
Writes the specified byte value to the designated I/O location.
Where:

<**name**> (input) Data type: const char*
A pointer to a NULL terminated string representing the name of the associated VPLC instance.

<**area**> (input) Data type: int
Designates the target I/O area. Valid specifications are:
VPLC_PI Specifies peripheral inputs.
VPLC_PQ Specifies peripheral outputs.

<**slave**> (input) Data type: int
Designates I/O slave number: 1<=<**slave**><=126. An out of range address specification generates a return error code.

<**slot**> (input) Data type: int
Designates I/O slot number: 0<=<**slot**><=247. An out of range address specification generates a return error code.

<**offset**> (input) Data type: int
Designates the byte offset from the module base address. This value is dependent upon the module's I/O length. Possible values are:
0 module I/O length of 1
0,1 module I/O length of 2.
0,1,2,3 module I/O length of 4.

<**value**> (input) Data type: unsigned char
Contains the I/O byte value to be written.

<**winerr**> (output) Data type: int
Returns the OS error that occurred during the operation as indicated by <err>.

<err> (return) Data type: int
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.
VPLC_INV_AREA	Indicates that the specified I/O <**area**> is not valid.
VPLC_INV_SLAVE	Indicates that the specified I/O <**slave**> number exceeds the maximum range supported by the PLC.
VPLC_INV_SLOT	Indicates that the specified I/O <**slo**t> number exceeds the maximum range supported by the PLC.
VPLC_NO_SHARED_MEMORY	Indicates that no hardware configuration is present, but note that the callback function is still registered if specified.
VPLC_IO_NOT_RDY	Indicates that the I/O is not valid at this time because the PLC is not in RUN mode.

When a PQ (peripheral) value is forced to a particular state, if applicable, the associated Q (process image) value is also forced to that same state.

Function **vplc_write_io_byte_()** should only be called while the system I/O is valid as indicated by the callback function registered by **vplc_test_io_state()**, or indicated by the **plc_in_run_mode** member of **vplc_io_desc_type** as described in section 3.2.20 below.

3.2.7 vplc_write_io_word()

VPLC function **vplc_write_io_word()** writes the specified word value to the designated I/O location. vplc_write_io_word() invocation is:

```
<err> =
vplc_write_io_word(<name>,<area>,<adr>,<value>,&<winerr>);
```

Writes the specified word value to the designated I/O location.
where:

<**name**> (input) Data type: const char*
A pointer to a NULL terminated string representing the name of the associated VPLC instance.

<**area**> (input) Data type: int
Designates the target I/O area. Valid specifications are:

VPLC_I	Specifies process image inputs.
VPLC_Q	Specifies process image outputs.
VPLC_PI	Specifies peripheral inputs.
VPLC_PQ	Specifies peripheral outputs.

<**adr**> (input) Data type: int
Designates the target I/O address as a byte offset from the start (byte 0) of the area. For <area>s **VPLC_I** and **VPLC_Q** (process image), valid addresses are: (0<=**adr**<=255). For <area>s **VPLC_PI** and **VPLC_PQ** (peripheral), valid

addresses are: (0<=**adr**<=8191), but is dependant upon the number of I/O modules available. An out of range address specification generates a return error code.

<**value**> (input) Data type: unsigned short
Contains the I/O word value to be written.

<**winerr**> (output) Data type: int
Returns the OS error that occurred during the operation as indicated by <**err**>.

<**err**> (return) Data type: int
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.
VPLC_INV_AREA	Indicates that the specified I/O <**area**> is not valid.
VPLC_INV_ADR	Indicates that the specified I/O <**adr**> exceeds the maximum range supported by the PLC, or exceeds the address range of available I/O modules.
VPLC_NO_SHARED_MEMORY	Indicates that no hardware configuration is present, but note that the callback function is still registered if specified.
VPLC_IO_NOT_RDY	Indicates that the I/O is not valid at this time because the PLC is not in RUN mode.

When a PQ (peripheral) value is forced to a particular state, if applicable, the associated Q (process image) value is also forced to that same state.

Function **vplc_write_io_word()** should only be called while the system I/O is valid as indicated by the callback function registered by **vplc_reg_io_state()**, or indicated by the **plc_in_run_mode** member of **vplc_io_desc_type** as described in section 3.2.20 below.

3.2.8 vplc_write_io_word_()

VPLC function **vplc_write_io_word_()** writes the specified word value to the designated I/O location. **vplc_write_io_word_()** is the same as **vplc_write_io_word()** except that it specifies that I/O be addressed by slave/slot instead of by a logical address, and that the VPLC process image cannot be directly accessed. **vplc_write_io_word_()** invocation is:

```
<err> =
vplc_write_io_word_(<name>,<area>,<slave>,<slot>,<offset>,<value>,&<winerr>);
```

Writes the specified word value to the designated I/O location.
Where:

`<name>` `(input)` `Data type: const char*`
A pointer to a NULL terminated string representing the name of the associated VPLC instance.

`<area>` `(input)` `Data type: int`
Designates the target I/O area. Valid specifications are:

VPLC_PI	Specifies peripheral inputs.
VPLC_PQ	Specifies peripheral outputs.

`<slave>` `(input)` `Data type: int`
Designates I/O slave number: 1<=<slave><=126. An out of range address specification generates a return error code.

`<slot>` `(input)` `Data type: int`
Designates I/O slot number: 0<=<**slot**><=247. An out of range address specification generates a return error code.

`<offset>` `(input)` `Data type: int`
Designates the byte offset from the module base address. This value is dependent upon the module's I/O length. Possible values are:

0	module I/O length of 1.
0,1	module I/O length of 2.
0,1,2,3	module I/O length of 4.

`<value>` `(input)` `Data type: unsigned short`
Contains the I/O word value to be written.
`<winerr>` `(output)` `Data type: int`
Returns the OS error that occurred during the operation as indicated by <err>.

`<err>` `(return)` `Data type: int`
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.
VPLC_INV_AREA	Indicates that the specified I/O <**area**> is not valid.
VPLC_INV_SLAVE	Indicates that the specified I/O <**slave**> number exceeds the maximum range supported by the PLC.

VPLC_INV_SLOT	Indicates that the specified I/O <**slot**> number exceeds the maximum range supported by the PLC.
VPLC_NO_SHARED_MEMORY	Indicates that no hardware configuration is present, but note that the callback function is still registered if specified.
VPLC_IO_NOT_RDY	Indicates that the I/O is not valid at this time because the PLC is not in RUN mode.

When a PQ (peripheral) value is forced to a particular state, if applicable, the associated Q (process image) value is also forced to that same state.

Function **vplc_write_io_word_()** should only be called while the system I/O is valid as indicated by the callback function registered by **vplc_reg_io_state()**, or indicated by the **plc_in_run_mode** member of **vplc_io_desc_type** as described in section 3.2.20 below.

3.2.9 vplc_write_io_dword()

VPLC function **vplc_write_io_dword()** writes the specified dword value to the designated I/O location. **vplc_write_io_dword()** invocation is:

```
<err> =
vplc_write_io_dword(<name>,<area>,<adr>,<value>,&<winerr>);
```

Writes the specified dword value to the designated I/O location.

Where:

<**name**> (input) Data type: const char*

A pointer to a NULL terminated string representing the name of the associated VPLC instance.

<**area**> (input) Data type: int

Designates the target I/O area. Valid specifications are:

VPLC_I	Specifies process image inputs.
VPLC_Q	Specifies process image outputs.
VPLC_PI	Specifies peripheral inputs.
VPLC_PQ	Specifies peripheral outputs.

<**adr**> (input) Data type: int

Designates the target I/O address as a byte offset from the start (byte 0) of the area. For <**area**>s **VPLC_I** and **VPLC_Q** (process image), valid addresses are: (0<=adr<=255). For <**area**>s **VPLC_PI** and **VPLC_PQ** (peripheral), valid addresses are: (0<=**adr**<=8191), but is dependant upon the number of I/O modules available. An out of range address specification generates a return error code.

<**value**> (input) Data type: unsigned int

Contains the I/O dword value to be written.

<**winerr**> (output) Data type: int

Returns the OS error that occurred during the operation as indicated by <**err**>.

<err> (return) Data type: int
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.
VPLC_INV_AREA	Indicates that the specified I/O <**area**> is not valid.
VPLC_INV_ADR	Indicates that the specified I/O <**adr**> exceeds the maximum range supported by the PLC, or exceeds the address range of available I/O modules.
VPLC_NO_SHARED_MEMORY	Indicates that no hardware configuration is present, but note that the callback function is still registered if specified.
VPLC_IO_NOT_RDY	Indicates that the I/O is not valid at this time because the PLC is not in RUN mode.

When a PQ (peripheral) value is forced to a particular state, if applicable, the associated Q (process image) value is also forced to that same state.

Function **vplc_write_io_dword()** should only be called while the system I/O is valid as indicated by the callback function registered by **vplc_reg_io_state()**, or indicated by the **plc_in_run_mode** member of **vplc_io_desc_type** as described in section 3.2.20 below.

3.2.10 vplc_write_io_dword_()

VPLC function **vplc_write_io_dword_()** writes the specified dword value to the designated I/O location. **vplc_write_io_dword_()** is the same as **vplc_write_io_dword()** except that it specifies that I/O be addressed by slave/slot instead of by a logical address, and that the VPLC process image cannot be directly accessed. **vplc_write_io_dword_()** invocation is:

```
<err> =
vplc_write_io_dword_(<name>,<area>,<slave>,<slot>,<value>,&<
winerr>);
```

Writes the specified dword value to the designated I/O location.
Where:

<name> (input) Data type: const char*
A pointer to a NULL terminated string representing the name of the associated VPLC instance.

<area> (input) Data type: int
Designates the target I/O area. Valid specifications are:

VPLC_PI	Specifies peripheral inputs.
VPLC_PQ	Specifies peripheral outputs.

<**slave**> (input) Data type: int
Designates I/O slave number: 1<=<slave><=126. An out of range address specification generates a return error code.

<**slot**> (input) Data type: int
Designates I/O slot number: 0<=<slot><=247. An out of range address specification generates a return error code.

<**value**> (input) Data type: unsigned int
Contains the I/O dword value to be written.

<**winerr**> (output) Data type: int
Returns the OS error that occurred during the operation as indicated by <**err**>.

<**err**> (return) Data type: int
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.
VPLC_INV_AREA	Indicates that the specified I/O <area> is not valid.
VPLC_INV_SLAVE	Indicates that the specified I/O <slave> number exceeds the maximum range supported by the PLC
VPLC_INV_SLOT	Indicates that the specified I/O <slot> number exceeds the maximum range supported by the PLC.
VPLC_NO_SHARED_MEMORY	Indicates that no hardware configuration is present, but note that the callback function is still registered if specified.
VPLC_IO_NOT_RDY	Indicates that the I/O is not valid at this time because the PLC is not in RUN mode.

When a PQ (peripheral) value is forced to a particular state, if applicable, the associated Q (process image) value is also forced to that same state.

Function **vplc_write_io_dword_()** should only be called while the system I/O is valid as indicated by the callback function registered by **vplc_reg_io_state()**, or indicated by the **plc_in_run_mode** member of **vplc_io_desc_type** as described in section 3.2.20 below.

3.2.11 vplc_read_io_bit()

VPLC function **vplc_read_io_bit()** reads the current bit state from the designated I/O location. **vplc_read_io_bit()** invocation is:

```
<err> =
vplc_read_io_bit(<name>,<area>,<adr>,<pos>,&<state>,&<winerr
>);
```

Writes the specified I/O bit to the designated state.

Where:

<**name**> (input) Data type: const char*

A pointer to a NULL terminated string representing the name of the associated VPLC instance.

<**area**> (input) Data type: int

Designates the target I/O area. Valid specifications are:

VPLC_I	Specifies process image inputs.
VPLC_Q	Specifies process image outputs.
VPLC_PI	Specifies peripheral inputs.
VPLC_PQ	Specifies peripheral outputs.

<**adr**> (input) Data type: int

Designates the target I/O address as a byte offset from the start (byte 0) of the area. For <**area**>s **VPLC_I** and **VPLC_Q** (process image), valid addresses are: (0<=**adr**<=255). For <**area**>s **VPLC_PI** and **VPLC_PQ** (peripheral), valid addresses are: (0<=**adr**<=8191), but are dependant upon the number of I/O modules available. An out of range address specification generates a return error code.

<**pos**> (input) Data type: int

Contains the target bit position within the specified I/O <adr>: 0<=<**pos**><=7.

<**state**> (input) Data type: bool

Returns the state of the designated bit address, either on (1) or off (0).

<**winerr**> (output) Data type: int

Returns the OS error that occurred during the operation as indicated by <**err**>.

<**err**> (return) Data type: int

Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.
VPLC_INV_AREA	Indicates that the specified I/O <**area**> is not valid.
VPLC_INV_ADR	Indicates that the specified I/O <**adr**> exceeds the maximum range supported by the PLC, or exceeds the address range of available I/O modules.
VPLC_INV_POS	Indicates that the specified bit position is out of range.

VPLC_NO_SHARED_MEMORY	Indicates that no hardware configuration is present, but note that the callback function is still registered if specified.
VPLC_IO_NOT_RDY	Indicates that the I/O is not valid at this time because the PLC is not in RUN mode.

Function **vplc_read_io_bit()** should only be called while the system I/O is valid as indicated by the callback function registered by **vplc_reg_io_state()**, or indicated by the **plc_in_run_mode** member of **vplc_io_desc_type** as described in section 3.2.20 below.

3.2.12 vplc_read_io_bit_()

VPLC function **vplc_read_io_bit_()** reads the current bit state from the designated I/O location. **vplc_ read_io_bit _()** is the same as **vplc_ read_io_bit_()** except that it specifies that I/O be addressed by slave/slot instead of by a logical address, and that the VPLC process image cannot be directly accessed. **vplc_read_io_bit_()** invocation is:

```
<err> =
vplc_read_io_bit_(<name>,<area>,<slave>,<slot>,<offset>,<pos>,&<state>,&<winerr>);
```

Writes the specified I/O bit to the designated state.
Where:

<**name**> (input) Data type: const char*
A pointer to a NULL terminated string representing the name of the associated VPLC instance.

<**area**> (input) Data type: int
Designates the target I/O area. Valid specifications are:

VPLC_PI	Specifies peripheral inputs.
VPLC_PQ	Specifies peripheral outputs.

<**slave**> (input) Data type: int
Designates I/O slave number: 1<=<**slave**><=126. An out of range address specification generates a return error code.

<**slot**> (input) Data type: int
Designates I/O slot number: 0<=<slot><=247. An out of range address specification generates a return error code.

<**offset**> (input) Data type: int
Designates the byte offset from the module base address. This value is dependent upon the module's I/O length. Possible values are:

0	module I/O length of 1.
0,1	module I/O length of 2.
0,1,2,3	module I/O length of 4.

<**pos**> (input) Data type: int
Contains the target bit position within the specified I/O address: 0<=<**pos**><=7.

<**state**> (input) Data type: bool
Returns the state of the designated bit address, either on (1) or off (0).

<**winerr**> (output) Data type: int
Returns the OS error that occurred during the operation as indicated by <**err**>.

<**err**> (return) Data type: int
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_INV_NAME	Indicates that the named VPLC does not exist
VPLC_INV_AREA	Indicates that the specified I/O <**area**> is not valid.
VPLC_INV_SLAVE	Indicates that the specified I/O <**slave**> number exceeds the maximum range supported by the PLC.
VPLC_INV_SLOT	Indicates that the specified I/O <**slot**> number exceeds the maximum range supported by the PLC.
VPLC_INV_POS	Indicates that the specified bit position is out of range
VPLC_NO_SHARED_MEMORY	Indicates that no hardware configuration is present, but note that the callback function is still registered if specified.
VPLC_IO_NOT_RDY	Indicates that the I/O is not valid at this time because the PLC is not in RUN mode.

Function **vplc_read_io_bit_()** should only be called while the system I/O is valid as indicated by the callback function registered by **vplc_reg_io_state()**, or indicated by the **plc_in_run_mode** member of **vplc_io_desc_type** as described in section 3.2.20 below.

3.2.13 vplc_read_io_byte()

VPLC function **vplc_read_io_byte(**) reads the designated byte value from I/O. vplc_read_io_byte() invocation is:

```
<err> =
vplc_read_io_byte(<name>,<area>,<adr>,&<value>,&<winerr>);
```

Reads the designated byte value from I/O.

Where:

<**name**> (input) Data type: const char*

A pointer to a NULL terminated string representing the name of the associated VPLC instance.

<**area**> (input) Data type: int

Designates the target I/O area. Valid specifications are:

VPLC_I	Specifies process image inputs.
VPLC_Q	Specifies process image outputs.
VPLC_PI	Specifies peripheral inputs.
VPLC_PQ	Specifies peripheral outputs.

<**adr**> (input) Data type: int

Designates the target I/O address as a byte offset from the start (byte 0) of the area. For <**area**>s **VPLC_I** and **VPLC_Q** (process image), valid addresses are: (0<=adr<=255). For <**area**>s **VPLC_PI** and **VPLC_PQ** (peripheral), valid addresses are: (0<=**adr**<=8191), but is dependant upon the number of I/O modules available. An out of range address specification generates a return error code.

<**value**> (output) Data type: unsigned char

Returns the I/O byte value.

<**winerr**> (output) Data type: int

Returns the OS error that occurred during the operation as indicated by <**err**>.

<**err**> (return) Data type: int

Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.
VPLC_INV_AREA	Indicates that the specified I/O <area> is not valid.
VPLC_INV_ADR	Indicates that the specified I/O <adr> exceeds the maximum range supported by the PLC, or exceeds the address range of available I/O modules.
VPLC_NO_SHARED_MEMORY	Indicates that no hardware configuration is present, but note that the callback function is still registered if specified.
VPLC_IO_NOT_RDY	Indicates that the I/O is not valid at this time because the PLC is not in RUN mode.

Function **vplc_read_io_byte()** should only be called while the system I/O is valid as indicated by the callback function registered by **vplc_reg_io_state()**, or indicated by the **plc_in_run_mode** member of **vplc_io_desc_type** as described in section 3.2.20 below.

3.2.14 vplc_read_io_byte_()

VPLC function **vplc_read_io_byte_()** reads the designated byte value from I/O. **vplc_read_io_byte _()** is the same as **vplc_ read_io_byte_()** except that it specifies that I/O be addressed by slave/slot instead of by a logical address, and that the VPLC process image cannot be directly accessed. **vplc_read_io_byte_()** invocation is:

```
<err> =
vplc_read_io_byte_(<name>,<area>,<slave>,<slot>,<offset>,&<
value>,&<winerr>);
```

Reads the designated byte value from I/O.

Where:

<**name**> (input) Data type: const char*

A pointer to a NULL terminated string representing the name of the associated VPLC instance.

<**area**> (input) Data type: int

Designates the target I/O area. Valid specifications are:

VPLC_PI	Specifies peripheral inputs.
VPLC_PQ	Specifies peripheral outputs.

<**slave**> (input) Data type: int

Designates I/O slave number: 1<=<**slave**><=126. An out of range address specification generates a return error code.

<**slot**> (input) Data type: int

Designates I/O slot number: 0<=<**slot**><=247. An out of range address specification generates a return error code.

<**offset**> (input) Data type: int

Designates the byte offset from the module base address. This value is dependent upon the module's I/O length. Possible values are:

0	module I/O length of 1.
0,1	module I/O length of 2.
0,1,2,3	module I/O length of 4.

<**value**> (output) Data type: unsigned char

Returns the I/O byte value.

<**winerr**> (output) Data type: int

Returns the OS error that occurred during the operation as indicated by <err>.

<**err**> (return) Data type: int

Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully
VPLC_WINERR	Indicates that a Windows error was encountered.

VPLC_INV_NAME	Indicates that the named VPLC does not exist.
VPLC_INV_AREA	Indicates that the specified I/O <area> is not valid.
VPLC_INV_SLAVE	Indicates that the specified I/O <slave> number exceeds the maximum range supported by the PLC.
VPLC_INV_SLOT	Indicates that the specified I/O <slot> number exceeds the maximum range supported by the PLC
VPLC_NO_SHARED_MEMORY	Indicates that no hardware configuration is present, but note that the callback function is still registered if specified.
VPLC_IO_NOT_RDY	Indicates that the I/O is not valid at this time because the PLC is not in RUN mode.

Function **vplc_read_io_byte_()** should only be called while the system I/O is valid as indicated by the callback function registered by **vplc_reg_io_state()**, or indicated by the **plc_in_run_mode** member of **vplc_io_desc_type** as described in section 3.2.20 below.

3.2.15 vplc_read_io_word()

VPLC function **vplc_read_io_word()** reads the designated word value from I/O. **vplc_read_io_word()** invocation is:

```
<err> =
vplc_read_io_word(<name>,<area>,<adr>,&<value>,&<winerr>);
```

Reads the designated word value from I/O.

Where:

`<name>` `(input)` `Data type: const char*`

A pointer to a NULL terminated string representing the name of the associated VPLC instance.

`<area>` `(input)` `Data type: int`

Designates the target I/O area. Valid specifications are:

VPLC_I	Specifies process image inputs.
VPLC_Q	Specifies process image outputs.
VPLC_PI	Specifies peripheral inputs.
VPLC_PQ	Specifies peripheral outputs.

`<adr>` `(input)` `Data type: int`

Designates the target I/O address as a byte offset from the start (byte 0) of the area. For <area>s **VPLC_I** and **VPLC_Q** (process image), valid addresses are: (0<=**adr**<=255). For <**area**>s **VPLC_PI** and **VPLC_PQ** (peripheral), valid addresses are: (0<=**adr**<=8191), but is dependant upon the number of I/O modules available. An out of range address specification generates a return error code.

`<value>` `(output)` `Data type: unsigned short`

Returns the I/O word value.

`<winerr>` `(output)` `Data type: int`

Returns the OS error that occurred during the operation as indicated by <**err**>.

<err> (return) Data type: int
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.
VPLC_INV_AREA	Indicates that the specified I/O <**area**> is not valid.
VPLC_INV_ADR	Indicates that the specified I/O <**adr**> exceeds the maximum range supported by the PLC, or exceeds the address range of available I/O modules.
VPLC_NO_SHARED_MEMORY	Indicates that no hardware configuration is present, but note that the callback function is still registered if specified.
VPLC_IO_NOT_RDY	Indicates that the I/O is not valid at this time because the PLC is not in RUN mode.

Function **vplc_read_io_word()** should only be called while the system I/O is valid as indicated by the callback function registered by **vplc_reg_io_state()**, or indicated by the **plc_in_run_mode** member of **vplc_io_desc_type** as described in section 3.2.20 below.

3.2.16 vplc_read_io_word_()

VPLC function **vplc_read_io_word_()** reads the designated word value from I/O. **vplc_read_io_word _()** is the same as **vplc_ read_io_word_()** except that it specifies that I/O be addressed by slave/slot instead of by a logical address, and that the VPLC process image cannot be directly accessed. **vplc_read_io_word_()** invocation is:

```
<err> =
vplc_read_io_word_(<name>,<area>,<slave>,<slot>,<offset>,&<
value>,&<winerr>);
```

Reads the designated word value from I/O.
Where:

<**name**> (input) Data type: const char*
A pointer to a NULL terminated string representing the name of the associated VPLC instance.

<**area**> (input) Data type: int
Designates the target I/O area. Valid specifications are:

VPLC_PI	Specifies peripheral inputs.
VPLC_PQ	Specifies peripheral outputs.

<**slave**> (input) Data type: int
Designates I/O slave number: 1<=<**slave**><=126. An out of range address specification generates a return error code.

<**slot**> (input) Data type: int
Designates I/O slot number: 0<=<**slot**><=247. An out of range address specification generates a return error code.

<**offset**> (input) Data type: int
Designates the byte offset from the module base address. This value is dependent upon the module's I/O length. Possible values are:

0	module I/O length of 1.
0,1	module I/O length of 2.
0,1,2,3	module I/O length of 4.

<**value**> (output) Data type: unsigned short
Returns the I/O word value.

<**winerr**> (output) Data type: int
Returns the OS error that occurred during the operation as indicated by <**err**>.

<**err**> (return) Data type: int
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.
VPLC_INV_AREA	Indicates that the specified I/O <area> is not valid.
VPLC_INV_SLAVE	Indicates that the specified I/O <slave> number exceeds the maximum range supported by the PLC.
VPLC_INV_SLOT	Indicates that the specified I/O <slot> number exceeds the maximum range supported by the PLC.
VPLC_NO_SHARED_MEMORY	Indicates that no hardware configuration is present, but note that the callback function is still registered if specified.
VPLC_IO_NOT_RDY	Indicates that the I/O is not valid at this time because the PLC is not in RUN mode.

Function **vplc_read_io_word_()** should only be called while the system I/O is valid as indicated by the callback function registered by **vplc_reg_io_state()**, or indicated by the **plc_in_run_mode** member of vplc_io_desc_type as described in section 3.2.20 below.

3.2.17 vplc_read_io_dword()

VPLC function **vplc_read_io_dword()** reads the designated dword value from I/O. **vplc_read_io_dword()** invocation is:

```
<err> =
vplc_read_io_dword(<name>,<area>,<adr>,&<value>,&<winerr>);
```

Reads the designated dword value from I/O.

Where:

<**name**> (input) Data type: const char*

A pointer to a NULL terminated string representing the name of the associated VPLC instance.

<**area**> (input) Data type: int

Designates the target I/O area. Valid specifications are:

VPLC_I	Specifies process image inputs.
VPLC_Q	Specifies process image outputs.
VPLC_PI	Specifies peripheral inputs.
VPLC_PQ	Specifies peripheral outputs.

<**adr**> (input) Data type: int

Designates the target I/O address as a byte offset from the start (byte 0) of the area. For <**area**>s **VPLC_I** and **VPLC_Q** (process image), valid addresses are: (0<=**adr**<=255). For <**area**>s **VPLC_PI** and **VPLC_PQ** (peripheral), valid addresses are: (0<=**adr**<=8191), but is dependant upon the number of I/O modules available. An out of range address specification generates a return error code.

<**value**> (output) Data type: unsigned int

Returns the I/O dword value.

<**winerr**> (output) Data type: int

Returns the OS error that occurred during the operation as indicated by <**err**>.

<**err**> (return) Data type: int

Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.
VPLC_INV_AREA	Indicates that the specified I/O <**area**> is not valid.
VPLC_INV_ADR	Indicates that the specified I/O <**adr**> exceeds the maximum range supported by the PLC, or exceeds the address range of available I/O modules.
VPLC_NO_SHARED_MEMORY	Indicates that no hardware configuration is present, but note that the callback function is still registered if specified.
VPLC_IO_NOT_RDY	Indicates that the I/O is not valid at this time because the PLC is not in RUN mode.

Function **vplc_read_io_dword()** should only be called while the system I/O is valid as indicated by the callback function registered by **vplc_reg_io_state()**, or indicated by the **plc_in_run_mode** member of **vplc_io_desc_type** as described in section 3.2.20 below.

3.2.18 vplc_read_io_dword_()

VPLC function **vplc_read_io_dword_()** reads the designated dword value from I/O. **vplc_read_io_dword _()** is the same as **vplc_ read_io_dword_()** except that it specifies that I/O be addressed by slave/slot instead of by a logical address, and that the VPLC process image cannot be directly accessed.
vplc_read_io_dword_() invocation is:

```
<err> =
vplc_read_io_dword_(<name>,<area>,<slave>,<slot>,&<value>,&
<winerr>);
```

Reads the designated dword value from I/O.
Where:

<**name**> (input) Data type: const char*
A pointer to a NULL terminated string representing the name of the associated VPLC instance.

<**area**> (input) Data type: int
Designates the target I/O area. Valid specifications are:

VPLC_PI	Specifies peripheral inputs.
VPLC_PQ	Specifies peripheral outputs.

<**slave**> (input) Data type: int
Designates I/O slave number: 1<=<**slave**><=126. An out of range address specification generates a return error code.

<**slot**> (input) Data type: int
Designates I/O slot number: 0<=<**slot**><=247. An out of range address specification generates a return error code.

<**value**> (output) Data type: unsigned int
Returns the I/O dword value.

<**winerr**> (output) Data type: int
Returns the OS error that occurred during the operation as indicated by <**err**>.

<**err**> (return) Data type: int
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully
VPLC_WINERR	Indicates that a Windows error was encountered.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.
VPLC_INV_AREA	Indicates that the specified I/O <**area**> is not valid.
VPLC_INV_SLAVE	Indicates that the specified I/O <**slave**> number exceeds the maximum range supported by the PLC.

VPLC_INV_SLOT	Indicates that the specified I/O <**slot**> number exceeds the maximum range supported by the PLC.
VPLC_NO_SHARED_MEMORY	Indicates that no hardware configuration is present, but note that the callback function is still registered if specified.
VPLC_IO_NOT_RDY	Indicates that the I/O is not valid at this time because the PLC is not in RUN mode.

Function **vplc_read_io_dword_()** should only be called while the system I/O is valid as indicated by the callback function registered by **vplc_reg_io_state()**, or indicated by the **plc_in_run_mode** member of **vplc_io_desc_type** as described in section 3.2.20 below.

3.2.19 vplc_io_terminate()

Function **vplc_io_terminate()** shuts down the VPLC I/O simulation server.
vplc_io_terminate() invocation is:

```
<err> = vplc_io_terminate(<name>,&<winerr>);
```

Terminates the VPLC I/O simulation server.
Where:

<**name**> (input) Data type: const char*
A pointer to a NULL terminated string specifying the name of the designated VPLC.

<**err**> (return) Data type: int
Returns the status of the operation. Possible return values are:

VPLC_OK	Indicates that the operation completed successfully.
VPLC_WINERR	Indicates that a Windows error was encountered. The Windows error value is returned in <**winerr**>.
VPLC_INV_NAME	Indicates that the named VPLC does not exist.

Note that this function is not actually required to terminate the I/O simulation facility because the I/O simulation server is shut down automatically when the associated VPLC client process terminates.

3.2.20 vplc_io_desc_type

Interface structure **vplc_io_desc_type** consists of the following elements:

io_desc_ver Data type: unsigned char
Indicates the version of **vplc_io_desc_type**. Valid version specifications are:
VPLC_IO_DESC_VERSION_1

reserved1 Data type: unsigned char
The user may not use this field.

dp_subsystem_count Data type: unsigned char
Indicates the number of DP masters present in the VPLC system, as specified by the downloaded Step-7 project. Specifically:
0<= **dp_subsystem_count** <= **VPLC_MAX_DP_SUBSYSTEMS**

Reserved2 Data type: unsigned char
The user may not use this field.

plc_in_run_mode Data type: unsigned char
Indicates whether I/O is valid or not.
When **TRUE**, I/O is valid and may be accessed.
When **FALSE**, I/O is invalid and direct accessing is undefined.

input_process_image_ptr Data type: void *
Pointer to the start of the inputs process image table.

output_process_image_ptr Data type: void *
Pointer to the start of the outputs process image table.

dp_subsystem [**VPLC_MAX_DP_SUBSYSTEMS**] Data type: **vplc_dp_subsystem_type.**
Describes the associated DP subsystem as defined by **vplc_dp_subsystem_type** as described in the section 3.2.21 below.

3.2.21 vplc_dp_subsystem_type

Interface structure **vplc_dp_subsystem_type** describes the associated DP master subsystem and consists of the following elements:

device_id Data type: unsigned char
Indicates the Profibus device id associated with the corresponding DP I/O subsystem. Specifically:
1<=dp subsystem id<=VPLC MAX DP SUBSYSTEM

dp_subsystem_id Data type: unsigned char
Indicates the master id of the associated DP I/O subsystem. Specifically:
1<=**dp_subsystem_id**<=**VPLC_MAX_DP_SUBSYSTEM**

module_count Data type: unsigned short
Indicates the number of I/O modules that exist in the associated DP subsystem. Specifically:
0<=**module_count**<=**VPLC_MAX_MODULES_PER_DEVICE**

input_start_ptr Data type: void *
Pointer to the start of physical I/O inputs.

inputs_len Data type: int
Byte number of physical I/O inputs.

output_start_ptr Data type: void *
Pointer to the start of physical I/O outputs.

outputs_len Data type: int
Byte number of physical I/O outputs.

module_info[VPLC_MAX_MODULES_PER_DEVICE] Data type: **vplc_module_info_type.**
Describes the associated DP I/O module as defined by **vplc_module_info_type** as described in the next section.

3.2.22 vplc_module_info_type

Interface structure **vplc_module_info_type** describes the associated DP I/O module and consists of the following elements:

io Data type: unsigned char
Indicates whether the associated I/O module is an input or an output module. Specifically:

0 indicates an input module.

1 indicates an outputs module.

mod_io_len Data type: unsigned char
Indicates the size of the I/O module. Specifically:
0 indicates no I/O.
1 indicates 1 byte (8 bits) of I/O.
2 indicates 2 bytes (16 bits) of I/O.
4 indicates 4 bytes (32 bits) of I/O.

station_adr Data type: unsigned char
Indicates the DP station address of the slave containing this I/O module. This value is limited by the associated DP I/O hardware.

slot Data type: unsigned char
Indicates the I/O module's slot number in its associated DP slave. This value is limited by the associated DP I/O hardware.

mod_type Data type: unsigned short
Indicates the Siemens DP I/O module type.

log_adr Data type: unsigned short
Indicates the logical address of the start of this modules I/O, i. e, the address of the module's I/O as seen by the PLC program. Specifically:
0<=**log_adr**<256 – **log_adr** is an index into the process image table.
256>=**log_adr**<8192 – The I/O module is not addressable via the process image table.

phy_adr_ptr Data type: void *
Pointer to the start of this module's DP I/O.