

VTC Software API Document

TABLE OF CONTENTS

1. Purpose.....	3
2. Scope	3
3. Definitions, Acronyms and Abbreviations.....	3
4. References	3
5. Middleware.....	4
5.1. Overview.....	4
6. API Commands	5
6.1. System Info/Status	5
6.1.1. <i>GetFWVersion</i>	5
6.1.2. <i>GetModbusID</i>	6
6.1.3. <i>GetHardwareID</i>	6
6.1.4. <i>GetSystemStatus</i>	7
6.1.5. <i>GetDriverStatus</i>	8
6.1.6. <i>GetLEDStatus</i>	9
6.2. Operation Mode	10
6.2.1. <i>ControlChannel</i>	10
6.2.2. <i>SetTargetVoltage</i>	11
6.2.3. <i>GetTargetVoltage</i>	11
6.2.4. <i>SetTargetCurrent</i>	12
6.2.5. <i>GetTargetCurrent</i>	12
6.2.6. <i>SetTargetTemp</i>	13
6.2.7. <i>GetTargetTemp</i>	14
6.2.8. <i>GetTargetRamp</i>	14
6.2.9. <i>GetActualVoltage</i>	15
6.2.10. <i>GetActualCurrent</i>	15
6.2.11. <i>GetActualTemp</i>	16
6.2.12. <i>SetThermalProfile</i>	16
6.2.13. <i>ExecuteThermalProfile</i>	17
6.2.14. <i>GetThermalProfile</i>	17
6.2.15. <i>SetAutoOffTimer</i>	18
6.3. Editing a Thermal Profile	19
6.3.1. <i>TPReset</i>	19
6.3.2. <i>TPSetName</i>	20
6.3.3. <i>TPSetHCTemp</i>	20
6.3.4. <i>TPSetStepData</i>	21
6.3.5. <i>TPSetRepeat</i>	22
6.3.6. <i>TPSaveToEEPROM</i>	23
6.4. Thermal Profile Info	24
6.4.1. <i>TPGetHCTemp</i>	24
6.4.2. <i>TPGetStepData</i>	24
6.4.3. <i>TPGetAllSteps</i>	25
6.4.4. <i>TPGetInfo</i>	26
6.4.5. <i>TPGetStatus</i>	26

6.5.	Miscellaneous	28
6.5.1.	<i>Reset</i>	28
6.5.2.	<i>SetDataMonitoring</i>	28
6.5.3.	<i>ControlFan</i>	29
6.5.4.	<i>PIDGetParams</i>	29
6.5.5.	<i>PIDSetParams</i>	30
6.5.6.	<i>PIDSaveToEEPROM</i>	31
6.5.7.	<i>SetData</i>	31
6.5.8.	<i>GetData</i>	32
6.5.9.	<i>GetACR</i>	33
6.6.	Error List	33
7.	Revision History	35
8.	Legal Notices	36

1. PURPOSE

The purpose of this document is to elaborate upon all SW API supported by VTC firmware.

2. SCOPE

This document provides the details on the SW API supported by VTC firmware.

3. DEFINITIONS, ACRONYMS AND ABBREVIATIONS

Word or Acronyms	Definition
FW	Firmware
SW	Software
API	Application Programming Interface
VTC	VBMB TEC Controller
VMM	VBMB Module Manager

4. REFERENCES

Document Description	Document No.
VTC Hardware User Guide	VTC-UM-0004
VTC Software User Manual	VTC-UM-0006
VMM User Manual	VMM-UM-0002

5. MIDDLEWARE

5.1. OVERVIEW

The VTC communicates via USB serial connection. Middleware APIs are provided to assist end users in developing applications for the VTC. These APIs shorten development time by acting as an intermediary between the top-level application and the low level VTC firmware.

There are two methods of calling the APIs:

1. VMM Executable

The VBMB Module Manager (VMM) hosts a local server on port 58288 and listens to incoming messages from clients. A client can be any application developed by the end user to communicate with this server. The client can send the API commands (Section 6) to the VMM to trigger any VTC operation.

2. VBMB Python Library

Encapsulates the VTC APIs into a python module. Ideal for bring-up and testing of VTC applications directly over the command line.

The details of all APIs are listed in Section 6 of this manual. Refer to the VMM User manual for more info on using the VMM.

6. API COMMANDS

This section provides a detailed description and syntax for all VTC APIs.

All APIs share the same name irrespective of whether they are called via the VMM or the Python Library.

- When called via the VMM, the API commands are non-blocking and shall respond with an appropriate VMM status code
- When using the Python Library to control the VTC, all calls are blocking by default and context switching is left up to the end users.

This section provides information on all VTC APIs and their respective send/receive formats when called through the VMM and as a Python module.

For all python library syntax, it is assumed that the VTC module was instantiated as `vtc = VTC.VTC()`

6.1. SYSTEM INFO/STATUS

This section contains API's that query general info or status from the VTC.

6.1.1. GetFWVersion

Read VTC firmware version and return a hex-string.

E.g.: If firmware version is 0x0361, value returned shall be "0361"

Prototype: `GetFWVersion(void)`

Response: `GetFWVersion(string status,string version)`

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.
version	Firmware version of VTC as a hex string. Will only be returned if status is E00

In the python library, this command can be invoked as shown below.

```
ret = vtc.GetFWVersion().retval
```

If successful, ret shall contain the firmware version as a hex-string else None.

6.1.2. GetModbusID

Read the VTC modbus ID a.k.a. node ID and return a hex-string.
E.g.: If ID is 197 (0xC5), value returned would be "C5"

Prototype: GetModbusID(void)

Response: GetModbusID(string status,string id)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.
id	Modbus ID of VTC as a hex string. Will only be returned if status is E00

In the python library, this command can be invoked as shown below.

```
ret = vtc.GetModbusID().retval
```

If successful, ret shall contain the modbus ID as a hex-string else None.

6.1.3. GetHardwareID

Read VTC hardware ID and return an integer.

E.g.: If hardware ID is 20(0x14), value returned would be 20

Prototype: GetHardwareID(void)

Response: GetHardwareID(string status,int32 id)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.
id	Hardware id of the VTC as an integer. Will only be returned if status is E00

In the python library, this command can be invoked as shown below.

```
ret = vtc.GetHardwareID().retval
```

If successful, ret shall contain the hardware ID as an integer else None.

6.1.4. GetSystemStatus

Read the VTC system status bits and return a 16-bit hex string.

E.g.: If system status is 0xEF3F, value returned would be "EF3F".

Prototype: `GetSystemStatus(void)`

Response: `GetSystemStatus(string status,string sys)`

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.
sys	16-bit data representing the hardware status: bit[0-5] = channel 0-5 voltage sensor. bit[8] = TEC driver DRV8323 for channel A, B, C bit[9] = TEC driver DRV8323 for channel D, E, F bit[10] = Analog sensor ADS131 bit[11] = EEPROM status bit[12] = Atleast 1 TEC channel has an overheating problem. bit[13] = VDD status bit[14] = MOSFET driver status bit[15] = MOSFET gate enable Will only be returned if status is E00

In the python library, this command can be invoked as shown below.

```
ret = vtc.GetSystemStatus().retval
```

If successful, ret shall contain the system status as a 16-bit hex string else None.

6.1.5. GetDriverStatus

Read the TEC driver status bits and return a 32-bit hex string.

E.g.: If driver status is 0xFFFFFFFF, value returned would be "FFFFFFFF".

Prototype: GetDriverStatus(int32 channel)

Parameter	Description	Value
channel	VTC channel group	VTC channel group is 123 or 456

Response: GetDriverStatus(string status, string driver)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.
driver	32-bit data representing the driver status: bit[26] = FAULT: Logic OR of FAULT status registers bit[25] = VDS_OCP: Indicates VDS monitor overcurrent fault condition bit[24] = GDF: Indicates gate drive fault condition bit[23] = UVLO: Indicates under voltage lockout fault condition bit[22] = OTSD: Indicates over temperature shutdown bit[21] = VDS_HA: Indicates VDS overcurrent fault on the A high-side MOSFET bit[20] = VDS_LA: Indicates VDS overcurrent fault on the A low-side MOSFET bit[19] = VDS_HB: Indicates VDS overcurrent fault on the B high-side MOSFET bit[18] = VDS_LB: Indicates VDS overcurrent fault on the B low-side MOSFET bit[17] = VDS_HC: Indicates VDS overcurrent fault on the C high-side MOSFET bit[16] = VDS_LC: Indicates VDS overcurrent fault on the C low-side MOSFET bit[10] = SA_OC: Indicates overcurrent on phase A sense amplifier bit[9] = SB_OC: Indicates overcurrent on phase B sense amplifier bit[8] = SC_OC: Indicates overcurrent on phase C sense amplifier bit[7] = OTW: Indicates over temperature warning bit[6] = CPUV: Indicates charge pump under voltage fault condition bit[5] = VGS_HA: Indicates gate drive fault on the A high-side MOSFET bit[4] = VGS_LA: Indicates gate drive fault on the A low-side MOSFET bit[3] = VGS_HB: Indicates gate drive fault on the B high-side MOSFET bit[2] = VGS_LB: Indicates gate drive fault on the B low-side MOSFET bit[1] = VGS_HC: Indicates gate drive fault on the C high-side MOSFET bit[0] = VGS_LC: Indicates gate drive fault on the C low-side MOSFET The bit default value is 0b. Return value 1b means fault condition. Will only be returned if status is E00

In the python library, this command can be invoked as shown below.

```
ret = vtc.GetDriverStatus(123).retval
```

If successful, ret shall contain the driver status as a 32-bit hex string else None.

6.1.6. GetLEDStatus

Read the LED status and return a 16-bit hex string.

E.g.: If the LED status is 0x0001, value returned would be "0001".

Prototype: GetLEDStatus(void)

Response: GetLEDStatus(string status,string led)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.
led	16-bit data representing the LED status: bit[0-1] = channel 1 bit[2-3] = channel 2 bit[4-5] = channel 3 bit[6-7] = channel 4 bit[8-9] = channel 5 bit[10-11] = channel 6 0b00: Off 0b01: Running (Blue) 0b10: Stable (Green) 0b11 Warning (Red) Will only be returned if status is E00

In the python library, this command can be invoked as shown below.

```
ret = vtc.GetLEDStatus().retval
```

If successful, ret shall contain the LED status as a 16-bit hex string else None.

6.2. OPERATION MODE

Commands to interact with the VTC channel(s). Each VTC channel can be turned on/off individually and there are 4 operation modes available:

- Constant Voltage i.e., hold a specified voltage setpoint.
- Constant Current i.e., hold a specified current setpoint.
- Constant Temperature i.e., hold a specified temperature.
- Thermal profile.

6.2.1. ControlChannel

Enable/disable a specified channel. Disabling a channel does not clear any previous settings on the channel.

E.g.: If a setpoint was assigned to the channel before disabling, the setpoint shall remain active. When the channel is re-enabled, the VTC will start ramping up/down to the setpoint once again without any user inputs.

Prototype: `ControlChannel(int32 channel,int32 state)`

Parameter	Description	Value
channel	VTC channel number	1-6 = VTC channel 1 - 6
state	Channel state to be set	0 = Disable 1 = Enable

Response: `ControlChannel(string status,None)`

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.

In the python library, this command can be invoked as shown below.

```
ret = vtc.ControlChannel(channel = <channel>,
                        state = <state>).status
```

<channel> = 1 – 6 for each channel respectively

<state> = True/False to enable/disable the channel respectively

If successful, ret shall be True else False.

6.2.2. SetTargetVoltage

Set the target voltage for Constant Voltage mode on a specified channel and execute the setpoint. Setting a 0 voltage will turn off the channel.

Prototype: SetTargetVoltage(int32 channel, float voltage)

Parameter	Description	Value
channel	VTC channel number	1-6 = VTC channel 1-6 0 = Group control
voltage	Voltage setpoint. Set to 0 to turn off the VTC channel	Negative value = Heating Positive value = Cooling Can take any value from max heating voltage to max cooling voltage ¹

Response: SetTargetVoltage(string status, None)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.

In the python library, this command can be invoked as shown below.

```
ret = vtc.SetTargetVoltage(channel = <channel>,
                           voltage = <voltage>).status
```

<channel> and <voltage> data types are same as in VMM. If successful, ret shall return True else False.

6.2.3. GetTargetVoltage

Get the latest voltage setpoint.

Prototype: GetTargetVoltage(int32 channel)

Parameter	Description	Value
channel	VTC channel number	1-6 = VTC channel 1-6

Response: GetTargetVoltage(string status, float voltage)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.
voltage	The target voltage as a floating-point value. Will only be returned if status is E00

In the python library, this command can be invoked as shown below.

¹ Max cooling and heating voltages are configured by the user via the VTC Utility. Default value for max cooling voltage is +V_{in} and max heating voltage is -V_{in}

```
ret = vtc.GetTargetVoltage(channel = <channel>).retval
```

<channel> data types are same as in VMM. If successful, ret shall return the target voltage as a floating-point value else None.

6.2.4. SetTargetCurrent

Set the target current for Constant Current mode on a specified channel and execute the setpoint. Setting a 0 current will turn off the channel.

Prototype: SetTargetCurrent(int32 channel,float current)

Parameter	Description	Value
channel	VTC channel number	1-6 = VTC channel 1-6 0 = Group control
current	Current setpoint. Set to 0 to turn off the VTC channel	Negative value = Heating Positive value = Cooling Can take any value from $-I_{max}$ to $+I_{max}^2$

Response: SetTargetCurrent(string status,None)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.

In the python library, this command can be invoked as shown below.

```
ret = vtc.SetTargerCurrent(channel = <channel>,  
                           current = <current>).status
```

<channel> and <current> data types are same as in VMM. If successful, ret shall return True else False.

6.2.5. GetTargetCurrent

Get the latest current setpoint.

Prototype: GetTargetCurrent(int32 channel)

Parameter	Description	Value
channel	VTC channel number	1-6 = VTC channel 1-6

Response: GetTargetCurrent(string status,float current)

² I_{max} is configured by user through the VTC Utility

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.
current	The target current as a floating-point value. Will only be returned if status is E00.

In the python library, this command can be invoked as shown below.

```
ret = vtc.GetTargetCurrent(channel = <channel>).retval
```

<channel> data types are same as VMM. If successful ret shall return the target current as a floating-point value else None.

6.2.6. SetTargetTemp

Set the target temperature for Constant Temperature mode on a specified channel and execute the setpoint.

Prototype:

```
SetTargetTemp(int32 channel,float temp,float ramp)
```

Parameter	Description	Value
channel	VTC channel number	1-6 = VTC channel 1-6 0 = Group control
temp	Temperature setpoint. Set to 0 to turn off the VTC channel	Can take any value from -327 °C to +327 °C ³
ramp	Ramp rate for the VTC in °C/s	Can take any value from 0.001 to 65 °C/s

Response: SetTargetTemp(string status,None)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.

In the python library, this command can be invoked as shown below.

```
ret = vtc.SetTargetTemp(channel = <channel>,  
                        temp = <temp>,ramp = <ramp>).status
```

<channel>, <temp> and <ramp> data types are same as VMM. If successful, ret shall return True else False.

³ This is theoretical temperature limit user can set, actual operating range is dependent on thermal block.

6.2.7. GetTargetTemp

Get the latest temperature setpoint.

Prototype: GetTargetTemp(int32 channel)

Parameter	Description	Value
channel	VTC channel number	1-6 = VTC channel 1-6

Response: GetTargetTemp(string status,float temp)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.
temp	The target temperature as a floating-point value. Will only be returned if status is E00.

In the python library, this command can be invoked as shown below.

```
ret = vtc.GetTargetTemp(channel = <channel>).retval
```

<channel> data types are same as VMM. If successful, ret shall return the target temperature as a floating-point value else None.

6.2.8. GetTargetRamp

Get the ramp-rate for specified VTC channel in °C/s.

Prototype: GetTargetRamp(int32 channel)

Parameter	Description	Value
channel	VTC channel number	1-6 = VTC channel 1-6

Response: GetTargetRamp(string status,float ramp)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.
ramp	The channel ramp-rate as a floating-point value. Will only be returned if status is E00.

In the python library, this command can be invoked as shown below.

```
ret = vtc.GetTargetRamp(channel = <channel>).retval
```

<channel> data types are same as VMM. If successful, ret shall return the target channel ramp-rate as a floating-point value else None.

6.2.9. GetActualVoltage

Get the actual voltage of specified VTC channel. This command should be used for live monitoring the VTC channel.

Prototype: GetActualVoltage(int32 channel)

Parameter	Description	Value
channel	VTC channel number	1-6 = VTC channel 1-6

Response: GetActualVoltage(string status,float voltage)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.
voltage	The channel voltage as a floating-point value. Will only be returned if status is E00

In the python library, this command can be invoked as shown below.

```
ret = vtc.GetActualVoltage(channel = <channel>).retval
```

<channel> data types are same as VMM. If successful, ret shall return the channel voltage as a floating-point value else None.

6.2.10. GetActualCurrent

Get the actual current of the specified VTC channel. This command should be used for live monitoring the VTC channel.

Prototype: GetActualCurrent(int32 channel)

Parameter	Description	Value
channel	VTC channel number	1-6 = VTC channel 1-6

Response: GetActualCurrent(string status,float current)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.
current	The target current as a floating-point value. Will only be returned if status is E00.

In the python library, this command can be invoked as shown below.

```
ret = vtc.GetActualCurrent(channel = <channel>).retval
```

<channel> data types are same as VMM. If successful, ret shall return the channel current as a floating-point value else None.

6.2.11. GetActualTemp

Get the actual temperature of specified VTC channel. This command should be used for live monitoring the VTC channel. This command will report the temperature of the sensor configured as the “Main Sensor” for that channel. Refer to the User Manual for more info about configuring channel sensors via the VTC Utility.

Prototype: GetActualTemp(int32 channel)

Parameter	Description	Value
channel	VTC channel number	1-6 = VTC channel 1-6

Response: GetActualTemp(string status,float temp)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.
current	The target current as a floating-point value. Will only be returned if status is E00.

In the python library, this command can be invoked as shown below.

```
ret = vtc.GetActualTemp(channel = <channel>).retval
```

<channel> data types are same as VMM. If successful, ret shall return the channel temperature as a floating-point value else None.

6.2.12. SetThermalProfile

Assign the specified channel to the specified profile. Users can have multiple channels assigned to the same profile. However, this command will not run the profile and users need to make a call to ExecuteThermalProfile() to run the profile.

Prototype: SetThermalProfile(int32 channel,int32 profile)

Parameter	Description	Value
channel	VTC channel number	1-6 = VTC channel 1-6
profile	VTC Thermal Profile number	1-16 = VTC Thermal Profile 1-16

Response: SetThermalProfile(string status,None)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1: List of error codes for more info on status codes.

In the python library, this command can be invoked as shown below.

```
ret = vtc.SetThermalProfile(channel = <channel>,  
                             profile = <profile>).status
```

<channel> and <profile> data types are same as VMM. If successful, ret shall return True else False.

6.2.13. ExecuteThermalProfile

This command allows users to start, pause and stop execution of the selected thermal profile on all assigned channels.

Prototype: ExecuteThermalProfile(int32 profile,int32 action)

Parameter	Description	Value
profile	VTC Thermal Profile number	1-16 = VTC channel 1-16
action	Action to be taken on specified profile	255 = Stop profile execution 254 = Pause profile execution 0 – 253 = Start profile execution after 0 – 253 second delay

Response: ExecuteThermalProfile(string status,None)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.

In the python library, this command can be invoked as shown below.

```
ret = vtc.ExecuteThermalProfile(profile = <profile>,
                                action = <action>).status
```

<profile> and <action> data types are same as VMM. If successful, ret shall return True else False.

6.2.14. GetThermalProfile

Get the profile number to which the specified channel is assigned.

Prototype: GetThermalProfile(int32 channel)

Parameter	Description	Value
channel	VTC channel number	1-6 = VTC channel 1-6

Response: GetThermalProfile(string status,int32 profile)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.
profile	The VTC Thermal Profile number as an integer from 1 - 16

In the python library, this command can be invoked as shown below.

```
ret = vtc.GetThermalProfile(channel = <channel>).retval
```

<channel> data types are same as VMM. If successful, ret shall return the VTC Thermal profile to which the channel is assigned else None.

6.2.15. SetAutoOffTimer

Set the auto off timer on specified channel. Setting 0 means no timeout.

User need to send this command continuously before timeout reached if the timeout is not 0. Otherwise the VTC will turn off the channel after the timeout reached.

Prototype: SetAutoOffTimer(int32 channel,int32 timeout)

Parameter	Description	Value
channel	VTC channel number	1-6 = VTC channel 1-6 0 = Group control
timeout	Auto off timer timeout based on the 0.1s unit.	0 = no timeout

Response: SetAutoOffTimer(string status,None)

Parameter	Description
status	String representing the return status of this command. If status is E00: status ok. Refer to Table 6.1 for more info on status codes.

In the python library, this command can be invoked as shown below.

```
ret = vtc.SetAutoOffTimer(channel = <channel>,  
                           timeout = <timeout>).status
```

<channel> and <timeout> data types are same as in VMM. If successful, ret shall return True else False.

6.3. EDITING A THERMAL PROFILE

User can have a maximum of 16 thermal profiles saved in the VTC board. Each profile is identified by a unique profile number, ranging from 1 to 16.

Refer to the following steps to create a new thermal profile.

- Step 1) *Reset profile* – This will reset the data saved under this profile number in the memory.
- Step 2) *Set new profile name* – Define a new profile name, maximum 30 characters.
- Step 3) *Set heater cover temperature* (if applicable)
- Step 4) *Set profile step* – Define thermal profile details for each step. Step range is from 0 to 26. Maximum number of steps is 27.
- Step 5) *Set profile repeat step* – Define repeating steps and number of cycles.

Note:

Created thermal profiles 1-16 are stored in the RAM. Users can save these profiles to EEPROM using the `TPSaveToEEPROM()` command. Unsaved profiles will be lost upon VTC reset/power cycle. Since the EEPROM has a limited number of write cycles, users should avoid frequent writes to the EEPROM by ensuring that the saved profiles will not have any further changes. It is recommended to use the VTC Utility to create and test the thermal profiles before saving them to EEPROM.

6.3.1. TPReset

Clear the contents of specified thermal profile.

Prototype: `TPReset(int32 profile)`

Parameter	Description	Value
profile	VTC Thermal Profile number	1-16 = VTC Thermal Profile 1-16

Response: `TPReset(string status, None)`

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.

In the python library, this command can be invoked as shown below.

```
ret = vtc.TPReset(profile = <profile>).status
```

<profile> data types are same as in VMM. If successful, ret shall return True else False.

6.3.2. TPSetName

Set the name of the specified thermal profile.

Prototype: TPSetName(int32 profile,string name)

Parameter	Description	Value
profile	VTC Thermal Profile number	1-16 = VTC Thermal Profile 1-16
name	Name to be assigned to the VTC thermal profile	Max. name length is 30 ASCII characters. Cannot contain special characters

Response: TPSetName(string status,None)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.

In the python library, this command can be invoked as shown below.

```
ret = vtc.TPSetName(profile = <profile>,name = <name>).status
```

<profile> and <name> data types are same as in VMM. If successful, ret shall return True else False.

6.3.3. TPSetHCTemp

Set the heater cover temperature of the specified profile.

Prototype: TPSetHCTemp(int32 profile,int32 temp)

Parameter	Description	Value
profile	VTC Thermal Profile number	1-16 = VTC Thermal Profile 1-16
temp	Heater cover temperature in °C	Must be a positive integer

Response: TPSetHCTemp(string status,None)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.

In the python library, this command can be invoked as shown below.

```
ret = vtc.TPSetHCTemp(profile = <profile>,temp = <temp>).retval
```

<profile> and <temp> data types are same as in VMM. If successful, ret shall return True else False.

6.3.4. TPSetStepData

Set the step data parameters for specified step number of the specified VTC thermal profile.

Prototype:

```
TPSetStepData(int32 profile, int32 step, float tm0,
               float tm1, float ramp, float duration)
```

Parameter	Description	Value
profile	VTC Thermal Profile number	1-16 = VTC Thermal Profile 1-16
step	Profile step number	Integer between 0 – 26
tm0	Temperature 0 value of the step in °C.	Floating point number upto 2 decimal places. Can take any value from -327 °C to +327 °C ⁴ Used in gradient control ⁵
tm1	Temperature 1 value of the step in °C.	
ramp	Ramp rate to be used in gradient control in °C/s.	Floating point number upto 3 decimal places. Can take any value from 0.001 to 65 °C/s
duration	Amount of time (in seconds) taken to maintain the temperature specified in the step	0: Skip the step > 30000: Hold forever. Precise to the nearest 0.5 seconds.

Response: TPSetStepData(string status, None)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.

In the python library, this command can be invoked as shown below.

```
ret = vtc.TPSetStepData(profile = <profile>,
                        step = <step>, tm0 = <tm0>, tm1 = <tm1>,
                        ramp = <ramp>, duration = <dur>).status
```

<profile>, <step>, <tm0>, <tm1>, <ramp>, <dur> data types are same as in VMM.
If successful, ret shall return True else False.

⁴ This is theoretical temperature limit user can set. Actual operating range depends on thermal block.

⁵ If gradient control is not required, tm0 = tm1

6.3.5. TPSetRepeat

Set a repeat for specified set of steps in the specified profile.

E.g.: If a profile has 5 steps and repeat is set to 0, order of execution would be

Step 0 → Step 1 → Step 2 → Step 3 → Step 4

However, in the same scenario if repeat = 2 is set for Step 2 – Step 3, order of execution would be

Step 0 → Step 1 → Step 2 → Step 3 → Step 2 → Step 3 → Step 2 → Step 3 → Step 4

Prototype:

TPSetRepeat(int32 profile,int32 start,int32 end,int32 repeat)

Parameter	Description	Value
profile	VTC Thermal Profile number	1-16 = VTC Thermal Profile 1-16
start	The first step to be repeated	Positive integer between 0 - 26 This value should not exceed the number of steps in the profile
end	The last step to be repeated	
repeat	Number of additional times to repeat.	Must be a positive integer. 0 if no repeat is required

Response: TPSetRepeat(string status,None)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.

In the python library, this command can be invoked as shown below.

```
ret = vtc.TPSetRepeat(profile = <profile>,
                      start = <start>,end = <end>,
                      repeat = <repeat>).status
```

<profile>, <start>, <end>, <repeat> data types are same as in VMM. If successful, ret shall return True else False.

6.3.6. TPSToEEPROM

Save the specified thermal profile to EEPROM.

Prototype: TPSToEEPROM(int32 profile)

Parameter	Description	Value
profile	VTC Thermal Profile number	1-16 = VTC Thermal Profile 1-16

Response: TPSToEEPROM(string status, None)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.

In the python library, this command can be invoked as shown below.

```
ret = vtc.TPSToEEPROM(profile = <profile>).status
```

<profile> data types are same as in VMM. If successful, ret shall return True else False.

6.4. THERMAL PROFILE INFO

Users can use the following commands to retrieve the info about steps and other parameters in a VTC thermal profile.

6.4.1. TPGetHCTemp

Get the heater cover temperature of the specified profile.

Prototype: TPGetHCTemp(int32 profile)

Parameter	Description	Value
profile	VTC Thermal Profile number	1-16 = VTC Thermal Profile 1-16

Response: TPGetHCTemp(string status,int32 temp)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.
temp	Heater cover temperature as an integer in °C

In the python library, this command can be invoked as shown below.

```
ret = vtc.TPGetHCTemp(profile = <profile>).retval
```

<profile> data types are same as in VMM. If successful, ret shall return the HC temperature as an integer else None.

6.4.2. TPGetStepData

Get the data for the specified step number from the specified thermal profile.

Prototype: TPGetStepData(int32 profile,int32 step)

Parameter	Description	Value
profile	VTC Thermal Profile number	1-16 = VTC Thermal Profile 1-16
step	Profile step number	Positive integer between 0 – 26

Response: TPGetHCTemp(string status,dict StepData)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.

Parameter	Description
StepData	Dictionary of the form <pre>{ "TM0": <TM0 value in °C or None>, "TM1": <TM1 value in °C or None>, "Ramp": <Ramp rate in °C/s or None> "Duration": <Duration in s or None> }</pre>

In the python library, this command can be invoked as shown below.

```
ret = vtc.TPGetStepData(profile = <profile>,
                        step = <step>).retval
```

<profile> and <step> data types are same as in VMM. If successful, ret shall contain a dictionary as shown above else None.

6.4.3. TPGetAllSteps

Get the data for all steps in a specified thermal profile.

Prototype: TPGetAllSteps(int32 profile)

Parameter	Description	Value
profile	VTC Thermal Profile number	1-16 = VTC Thermal Profile 1-16

Response: TPGetAllSteps(string status,dict AllSteps)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.
AllSteps	Dictionary of the form <pre>{ "Step number from 0-26": <StepData dictionary for each step> }</pre>

In the python library, this command can be invoked as shown below.

```
ret = vtc.TPGetAllSteps(profile = <profile>).retval
```

<profile> data types are same as in VMM. If successful, ret shall return a dictionary as shown above else None.

6.4.4. TPGetInfo

Get all info for a specified thermal profile.

Prototype: TPGetInfo(int32 profile)

Parameter	Description	Value
profile	VTC Thermal Profile number	1-16 = VTC Thermal Profile 1-16

Response: TPGetInfo(string status,dict TPInfo)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.
TPInfo	Dictionary of the form: { "Name": <profile name as a string>, "HC": <profile HC temperature as an integer in °C>, "RStart": <Step number of first repeat step as an integer>, "REnd": <Step number of last repeat step as an integer>, "RCount": <Repeat count as an integer> "Profile": <AllSteps dictionary> }

In the python library, this command can be invoked as shown below.

```
ret = vtc.TPGetInfo(profile = <profile>).retval
```

<profile> data types are same as VMM. If successful, ret shall contain a dictionary as shown above else None.

6.4.5. TPGetStatus

Get the status of the specified thermal profile.

Prototype: TPGetStatus(int32 profile)

Parameter	Description	Value
profile	VTC Thermal Profile number	1-16 = VTC Thermal Profile 1-16

Response: TPGetStatus(string status,dict TPStatus)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.

Parameter	Description
TPStatus	Dictionary of the form: <pre>{ "Active": <Integer indicating whether profile is currently active. 0 = Inactive; 1 = Active>, "Time": <Floating point value indicating the time elapsed (in secs) since profile started execution>, "Step": <Integer indicating the step number being executed>, "Cycle": <Integer indicating the number of repeat cycles completed if any>, "TM0": <Floating point value indicating the TM0 temperature setpoint for the currently executing step> "TM1": <Floating point value indicating the TM1 temperature setpoint for the currently executing step> }</pre>

In the python library, this command can be invoked as shown below.

```
ret = vtc.TPGetStatus(profile = <profile>).retval
```

<profile> data types are same as VMM. If successful, ret shall contain a dictionary as shown above else None. TPStatus["Active"] shall be a boolean value instead of an integer when the command is invoked via the python library.

6.5. MISCELLANEOUS

Commands to control miscellaneous behaviour of the VTC.

6.5.1. Reset

Trigger a VTC system reset. This command shall return after a 5 second delay to allow enough time for the VTC to reset.

Prototype: Reset(void)

Response: Reset(string status, None)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.

In the python library, this command can be invoked as shown below.

```
ret = vtc.Reset()
```

If successful, ret shall be True else False.

6.5.2. SetDataMonitoring

VTC hardware status can be sent to user application for system data monitoring during operation. This command allows the user to configure the system data monitoring interval and its content.

Prototype: SetDataMonitoring(int32 data_mask, int32 interval)

Parameter	Description	Value
data_mask	16-bit data mask to enable/disable certain field in data monitoring 0: Disable mask 1: Enable mask	Bit 0: TEC1 info Bit 1: TEC2 info Bit 2: TEC3 info Bit 3: TEC4 info Bit 4: TEC5 info Bit 5: TEC6 info Bit 6: System Temperature Bit 7: System power Bit 8: Analog sensor A0-A8, Digital sensor D0-D7 temperature Bit 9: Analog sensor A0-A8 resistance Bit 10: TEC crossover info
interval	Data reporting interval in ms.	The data reporting interval is dependent on the PID control loop interval. Software will round the requested interval to the nearest possible setting.

Response: SetDataMonitoring(string status, None)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.

In the python library, this command can be invoked as shown below.

```
ret = vtc.SetDataMonitoring(data_mask = <data_mask>,
                           interval = <interval>).status
```

<data_mask> and <interval> have same data types as in VMM. If successful, ret shall be True else False.

6.5.3. ControlFan

Control the speed of the specified VTC fan.

Prototype: ControlFan(int32 fan_id, int32 pwm)

Parameter	Description	Value
fan_id	VTC fan number	1-3 = VTC Fan 1-3
pwm	The PWM of the specified fan in %	The PWM is rounded to the nearest tens. Can be a non-negative integer between 0-100 A value of 0 will turn off the fan

Response: ControlFan(string status, None)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.

In the python library, this command can be invoked as shown below.

```
ret = vtc.ControlFan(fan_id = <fan_id>, pwm = <pwm>).status
```

<fan_id> and <pwm> data types are same as in VMM. If successful, ret shall be True else False.

6.5.4. PIDGetParams

Read the K_p , T_i and T_d values for the VTC PID loop on the specified channel.

Prototype: PIDGetParams(int32 channel, int32 param_id)

Parameter	Description	Value
channel	VTC channel number	1-6 = VTC channel 1-6
param_id	Selects what PID param to read	1 = Read the K_p 2 = Read the T_i 3 = Read the T_d

Response: PIDGetParams(string status,float value)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.
value	The requested PID param as a floating-point integer. Will only be returned if status is E00

In the python library, this command can be invoked as shown below.

```
ret = vtc.PIDGetParams(channel = <channel>,
                        param_id = <param_id>).retval
```

<channel> and <param_id> data types are same as in VMM. If successful, ret shall contain the param value as a floating-point number else None.

6.5.5. PIDSetParams

Write the K_p , T_i , T_d values for the VTC PID loop on the specified channel. Edited parameters will take effect immediately but will not be saved to EEPROM.

Prototype:

PIDSetParams(int32 channel,int32 param_id,float value)

Parameter	Description	Value
channel	VTC channel number	1-6 = VTC channel 1-6
param_id	Selects what PID param to write	1 = Read the K_p 2 = Read the T_i 3 = Read the T_d
value	The value of the param to edit	

Response: PIDSetParams(string status,None)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.

In the python library, this command can be invoked as shown below.

```
ret = vtc.PIDSetParams(channel = <channel>,
                        param_id = <param_id>,
                        value = <value>).status
```

<channel>, <param_id> and <value> data types are same as in VMM. If successful, ret shall be True else False.

6.5.6. PIDSaveToEEPROM

Save the PID parameters for the specified VTC channel to the EEPROM. Once saved, these values will be loaded automatically when the VTC is powered up.

Prototype: PIDSaveToEEPROM(int32 channel)

Parameter	Description	Value
channel	VTC channel number	1-6 = VTC channel 1-6

Response: PIDSaveToEEPROM(string status, None)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.

In the python library, this command can be invoked as shown below.

```
ret = vtc.PIDSaveToEEPROM(channel = <channel>).status
```

<channel> data type is same as in VMM. If successful, ret shall be True else False.

6.5.7. SetData

Set the object data of the VTC.

Prototype: SetData(string object, string value)

Parameter	Description	Value
object	Object number.	Object number always 5 number (0-9,A-F) 0xTSSNN T - Data Type 0: Default return 16bit decimal -32768 to +32768 1: Hex return 16bit hex 0xFFFF. 2: Long return long decimal value -2147483648 to 2147483647 3: Float return number value 1.175494351 E-38 to 3.402823466 E+38 4: String return txt string like "xxxx" 5: Block return whole segment data, divide to 8 segments SS - Segment code (segment number 0x00-0x4F), each segment size 256 bytes (128 words). NN - Offset inside segment. Each step is 2 bytes. (0x00 -0x7F)
value	Object value.	The object value to be set according to the data type.

Response:

SetData(string status, string object, string error)

Parameter	Description
status	String representing the return status of this command. If status is E00: status ok. Refer to Table 6.1: List of error codes for more info on status codes.
object	Object number.

Parameter	Description
error	String representing the firmware error code. If command was successful, 0x0000 shall be returned. Refer to Error! Reference source not found. for more info.

Example 1: Set the data of segment 0x30 with offset 0x16.

Transmit : SetData(0x03016,0)

Receive : SetData(E00,0x03016,0x0000)

In the python library, this command can be invoked as shown below

```
ret = vtc.SetData(object = <object>, value= <value>)
```

<object> <value> data types are same as in VMM.

ret shall be a VBMBReturnStruct() object. Refer to VMM-UM-0002 for more info.

6.5.8. GetData

Get the object data of the VTC.

Prototype: GetData(string object)

Parameter	Description	Description
object	Object number.	Object number always 5 number (0-9,A-F) 0xTSSNN T - Data Type 0: Default return 16bit decimal -32768 to +32768 1: Hex return 16bit hex 0xFFFF. 2: Long return long decimal value -2147483648 to 2147483647 3: Float return number value 1.175494351 E-38 to 3.402823466 E+38 4: String return txt string like "xxxx" 5: Block return whole segment data, divide to 8 segments SS - Segment code (segment number 0x00-0x4F), each segment size 256 bytes (128 words). NN - Offset inside segment. Each step is 2 bytes. (0x00-0x7F)

Response:

GetData(string status, string object, string error, string data)

Parameter	Description
status	String representing the return status of this command. If status is E00: status ok. Refer to Table 6.1: List of error codes for more info on status codes.
object	Object number.
error	String representing the firmware error code. If command was successful, 0x0000 shall be returned. Refer to Error! Reference source not found. for more info.
data	Object data.

Example 1: Get the data of segment 0x02 with offset 0x00.

Transmit : GetData(0x30200)

Receive : GetData(E00,0x30200,0x0000,25.194490)

In the python library, this command can be invoked as shown below

```
ret = vtc.GetData(object = <object>).retval
```

<object> data types are same as in VMM.

ret shall be a VBMBReturnStruct() object. Refer to VMM-UM-0002 for more info.

6.5.9. GetACR

Get the ambient temperature and all channel's TEC AC Resistance (ACR) values according to the module type.

Note: It is important to note here that TEC ACR always to be compared at the same ambient temperature because it depends on the ambient temperature. TEC ACR grows with ambient temperature roughly by 0.5% per 1°C.

Prototype: GetACR(void)

Response: GetACR(string status, float temp, float ACR...)

Parameter	Description
status	String representing the return status of this command. If status id E00: status ok. Refer to Table 6.1 for more info on status codes.
temp	Ambient temperature
ACR	The TEC ACR values

Example 1: Get the ambient temperature and all channel's ACR values for the VTC1503.

Transmit : GetACR()

Receive : GetACR(E00,24.613712,2.000000,2.000000,2.000000)

In the python library, this command is not support.

6.6. ERROR LIST

Table 6.1 contains the various error codes seen in replies from the VMM. These error codes are not applicable to the Python Library

Error Code	Description
E00	Status OK
E02	Command cancelled by user.
E03	Command cancelled as took too long.
E04	Invalid command.
E06	Invalid no. of parameters.
E07	Invalid parameters.
E08	Failed to execute.
E09	Interlock safeguard.
E10	Commands are disabled.

Table 6.1: List of error codes

7. REVISION HISTORY

Version	Date Released	Initials	Changes
0.0.1	17 May 2023	LJR	Initial Release
0.0.2	14 Sept 2023	KY	Updated the APIs based on PyLib 0.7.2
0.0.3	29 Sept 2023	KY	Updated the APIs based on PyLib 0.7.3
0.0.4	06 Aug 2025	KY	Updated the APIs based on VMM v0.8.9

8. LEGAL NOTICES

The signed agreement between Purchaser and VBMB will govern the sale and purchase of Venture Biotech Modules Business Pte. Ltd.'s ("VBMB") products ("Products"). If no agreement has been concluded, VBMB's terms and conditions of supply will apply.

Testing and other quality control techniques are used to the extent that VBMB deems necessary to support its warranty.

Except where required by law, specific testing of all parameters of each Product is not necessarily performed.

Purchaser must provide adequate design and operating safeguards to minimize inherent or procedural and technical risks associated with Purchaser products and applications. Purchaser is solely responsible for its selection and use of VBMB Products. VBMB assumes no liability for applications assistance, Purchaser product design or any incompatibility of the Product with Purchaser product.

Products supplied by VBMB are not designed, intended, or authorized for use in life support, life sustaining, medical systems or devices, aircraft navigation, nuclear, or other applications, including, but not limited to, public transportation operating systems, in which the failure of such Products could reasonably be expected to result in personal injury, loss of life or severe property or environmental damage. Purchaser acknowledges that use of VBMB's Products in such product applications is understood to be fully at the risk of Purchaser and that Purchaser is responsible for verification and validation of the suitability of VBMB's Products in such applications. Purchaser agrees that VBMB is not and shall not be liable, in whole or in part, for any claim or damage arising from use in such applications. Purchaser agrees to indemnify, defend, and hold VBMB harmless from and against all claims, damages, losses, costs, expenses, and liabilities arising out of or in connection with any such use or application.

VBMB retains all rights to all proprietary intellectual property in the Products and associated manufacturing processes and has the right to file for and obtain intellectual property protection for same.