```
// ************************************************************
// Setpoint, temperature controller
// The setpoint is used by the temperature controller. For OPC UA
communication, use 0x71/vSPT instead.
// ************************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fGetvSP();
// parameter 1: new value in °C with an accuracy of 0.01 °C (an accuracy
of 0.001 °C is possible when DV E-grade is activated)
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fSetvSP(float fValue);

// ************************************************************
// Internal temperature
// The current temperature of the thermal fluid which is flowing to the
application.
// ************************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fGetvTI();

// ************************************************************
// Return temperature
// The current temperature of the thermal fluid which is flowing back to
the temperature control device.
// ************************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fGetvTR();

// ************************************************************
// Pump pressure (absolute)
// Absolute pump pressure at the pressure sensor in the thermal fluid
circuit of the thermostat.
// ************************************************************
// return value: actual value in 1 mbar
int iGetvpP();

// ************************************************************
// Current power
// The currently resulting power of the thermostat in watts.
// ************************************************************
// return value: actual value in 1 W
int iGetvPow();

// ************************************************************
// Error message
// The number of the error message that appeared first.
// ************************************************************
// return value: actual value in 1
int iGetvError();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvError(int value);

// ************************************************************
// Warning message
// The number of the most recent warning message to appear.
// ************************************************************
// return value: actual value in 1
int iGetvWarn();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvWarn(int value);

// ************************************************************
// Process temperature (Lemosa)
// The current process temperature.
// ************************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fGetvTE();

// ************************************************************
// Actual value setting, Internal temperature
// With this, an alternative measured value outside the thermostat can be
specified for the internal controller.
// ************************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fGetvIntMove();
// parameter 1: new value in °C with an accuracy of 0.01 °C (an accuracy
of 0.001 °C is possible when DV E-grade is activated)
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fSetvIntMove(float fValue);

// ************************************************************
// Setting, Process temperature
// With process control, the process measured value can be specified for
temperature control with this.
// ************************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fGetvExtMove();
// parameter 1: new value in °C with an accuracy of 0.01 °C (an accuracy
of 0.001 °C is possible when DV E-grade is activated)
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fSetvExtMove(float fValue);

// ***********************************************************
// Status of the thermostat
// Query current thermostat status. The returned value is a bit field. The
individual bits must be considered independently of each other.
// ***********************************************************
// return value: actual value of the bit field
int iGetvStatus1();
// Bit 0: TemperatureControlOn
// Temperature control operating mode: 1: active / 0: inactive
bool bGetvStatus1_Bit0_TemperatureControlOn();
// Bit 1: Circulation
// Circulation operating mode: 1: active / 0: inactive
bool bGetvStatus1_Bit1_Circulation();
// Bit 2: CompressorOn
// Refrigerator compressor: 1: switched on / 0: switched off
bool bGetvStatus1_Bit2_CompressorOn();
// Bit 3: TemperatureControlMode
// Temperature control mode “Process control”: 1: active / 0: inactive
bool bGetvStatus1_Bit3_TemperatureControlMode();
// Bit 4: PumpOn
// Circulating pump: 1: switched on / 0: switched off
bool bGetvStatus1_Bit4_PumpOn();
// Bit 5: CoolingPowerAvailable
// Cooling power available: 1: available / 0: not available
bool bGetvStatus1_Bit5_CoolingPowerAvailable();
// Bit 6: KeyLock
// Key lock: 1: active / 0: inactive
bool bGetvStatus1_Bit6_KeyLock();
// Bit 7: PIDParameterMode
// PID parameter set, temperature controller : 1: Automatic mode / 0:
Expert mode
bool bGetvStatus1_Bit7_PIDParameterMode();
// Bit 8: Error
// Error: 1: Error occurred / 0: no error
bool bGetvStatus1_Bit8_Error();
// Bit 9: Warning
// Warning: 1: New warning occurred / 0: No new warning
bool bGetvStatus1_Bit9_Warning();
// Bit 10: IntMoveActive
// Mode for setting the internal temperature (see Address 8): 1: active /
0: inactive
bool bGetvStatus1_Bit10_IntMoveActive();
// Bit 11: ExtMoveActive
// Mode for setting the external temperature (see Address 9): 1: active /
```

```
0: inactive
bool bGetvStatus1_Bit11_ExtMoveActive();
// Bit 12: DVEgrade
// DV E-grade: 1: activated / 0: not activated
bool bGetvStatus1_Bit12_DVEgrade();
// Bit 14: Restart
// Restart electronics / Power failure (*): 1: No new start / 0: New start
bool bGetvStatus1_Bit14_Restart();
// Bit 15: FreezeProtectionActive
// Freeze protection (not available on all devices): 1: active / 0: inactive
bool bGetvStatus1_Bit15_FreezeProtectionActive();

// ***********************************************************
// Control blow-down valve
// Specify blow-down valve position and query current position.
// ***********************************************************
// return value: actual value in 1
int iGetvBDPos();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvBDPos(int value);

// ***********************************************************
// Release blown-down valve heating
// Release heating for blow down valve.
// ***********************************************************
// return value: actual value in 1
int iGetvBDHeat();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvBDHeat(int value);

// ***********************************************************
// Fill level
// Fill level of the thermostat.
// ***********************************************************
// return value: actual value in % with an accuracy of 0.1 %
float fGetvNiv();

// ***********************************************************
// PID Parameter, automatic temperature controller.
// Set and query currently used temperature controller parameter set.
// ***********************************************************
// return value: actual value in 1
int iGetvAutoPID();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvAutoPID(int value);
```

```
// **********************************************************
// Temperature control mode
// Set and query the temperature control mode of the thermostat.
// **********************************************************
// return value: actual value in 1
int iGetvTmpMode();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvTmpMode(int value);

// **********************************************************
// Temperature control
// Start or stop temperature control of the thermostat or query current
status.
// **********************************************************
// return value: actual value in 1
int iGetvTmpActive();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvTmpActive(int value);

// **********************************************************
// Compressor operating mode
// The compressor operating mode can be set and queried.
// **********************************************************
// return value: actual value in 1
int iGetvCompAuto();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvCompAuto(int value);

// **********************************************************
// Circulation
// Start or stop thermostat circulation or query current status.
// **********************************************************
// return value: actual value in 1
int iGetvCircActive();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvCircActive(int value);

// **********************************************************
// Operating lock
// Activate, deactivate or query operating lock at Pilot.
// **********************************************************
// return value: actual value of the bit field
int iGetvKeyLock();
// parameter 1: new value of the bit field
// return value: actual value of the bit field
```

```
int iSetvKeyLock(int value);
// Bit 0: Lock
// Operating lock active or inactive. 0: Operating lock inactive. 1:
Operating lock active. Manual operation of the thermostat via Pilot is not
possible.
bool bGetvKeyLock_Bit0_Lock();
bool bSetvKeyLock_Bit0_Lock(bool value);
// Bit 1: WatchdogBehavior
// Watchdog behavior. 0: Watchdog inactive. 1: Activate Watchdog for 30 s.
If Bit 1 is not reset within 30 s, the operating lock is automatically
cancelled. This can be used to permit manual operation again if
communication with the thermostat is interrupted for any reason.
bool bGetvKeyLock_Bit1_WatchdogBehavior();
bool bSetvKeyLock_Bit1_WatchdogBehavior(bool value);

// ***********************************************************
// Internal temperature actual value setting mode
// Activate, deactivate or query operating mode for setting the internal
temperature.
// ***********************************************************
// return value: actual value of the bit field
int iGetvCITM();
// parameter 1: new value of the bit field
// return value: actual value of the bit field
int iSetvCITM(int value);
// Bit 0: ModeActive
// Operating mode active or inactive. 0: Mode not active. 1: Mode active.
Instead of the internal temperature, the temperature controller uses for
control the current content of Variable 0x08 (vIntMove) to calculate the
control variable. At the same time, when this bit is set for the first
time, a Watchdog is activated which reacts after 30 s. By regularly
setting the Variable 0x08 (vIntMove), it is possible to prevent the
Watchdog from being triggered.
bool bGetvCITM_Bit0_ModeActive();
bool bSetvCITM_Bit0_ModeActive(bool value);
// Bit 1: WatchdogBehavior
// Watchdog behavior. Defines the behavior of the thermostat if a data
communication problem is determined. 0: The thermostat issues the warning
-2129, deactivates the actual value setting and continues to operate
temperature control by using the thermostat's internal sensor. 1: The
thermostat issues the error message -328 and changes to standby mode
(temperature control is stopped).
bool bGetvCITM_Bit1_WatchdogBehavior();
bool bSetvCITM_Bit1_WatchdogBehavior(bool value);

// ***********************************************************
// Process temperature actual value setting mode
// Activate, deactivate or query operating mode for setting the process
temperature.
```

```
// ************************************************************
// return value: actual value of the bit field
int iGetvCETM();
// parameter 1: new value of the bit field
// return value: actual value of the bit field
int iSetvCETM(int value);
// Bit 0: ModeActive
// Operating mode active or inactive. 0: Mode not active. 1: Mode active.
Instead of the process temperature, the temperature controller uses for
control the current content of Variable 0x09 (vExtMove) to calculate the
control variable. At the same time, when this bit is set for the first
time, a Watchdog is activated which reacts after 30 s. By regularly
setting the Variable 0x09 (vExtMove) it is possible to prevent the
Watchdog from being triggered.
bool bGetvCETM_Bit0_ModeActive();
bool bSetvCETM_Bit0_ModeActive(bool value);
// Bit 1: WatchdogBehavior
// Watchdog behavior. Defines the behavior of the thermostat if a data
communication problem is determined. 0: The thermostat issues the warning
-2130, deactivates the actual value setting, changes from process to
internal control (see vTmpMode, Variable 19) and continues to operate
temperature control. 1: The thermostat issues the error message -329 and
changes to standby mode (temperature control is stopped).
bool bGetvCETM_Bit1_WatchdogBehavior();
bool bSetvCETM_Bit1_WatchdogBehavior(bool value);

// ************************************************************
// Freeze protection
// Activate, deactivate or query freeze protection of the thermostat.
// ************************************************************
// return value: actual value in 1
int iGetvICE();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvICE(int value);

// ************************************************************
// Serial Number
// Query the serial number of the thermostat.
// ************************************************************
// return value: serial number of the machine
int iGetvSNR();

// ************************************************************
// Kp of the internal controller
// Change and query amplification of the PID controller with internal
regulation.
// ************************************************************
// return value: actual value in 1
```

```
int iGetvKpInt();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvKpInt(int value);

// *********************************************************
// Tn of the internal controller
// Change and query reset time of the PID controller with internal
regulation.
// *********************************************************
// return value: actual value in s with an accuracy of 0.1 s
float fGetvTnInt();
// parameter 1: new value in s with an accuracy of 0.1 s
// return value: actual value in s
float fSetvTnInt(float fValue);

// *********************************************************
// Tv of the internal controller
// Change and query hold-back time of the PID controller with internal
regulation.
// *********************************************************
// return value: actual value in s with an accuracy of 0.1 s
float fGetvTvInt();
// parameter 1: new value in s with an accuracy of 0.1 s
// return value: actual value in s
float fSetvTvInt(float fValue);

// *********************************************************
// Kp of the jacket controller
// Change and query amplification of the PID controller with jacket control.
// *********************************************************
// return value: actual value in 1
int iGetvKpJack();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvKpJack(int value);

// *********************************************************
// Tn of the jacket controller
// Change and query reset time of the PID controller with jacket control.
// *********************************************************
// return value: actual value in s with an accuracy of 0.1 s
float fGetvTnJack();
// parameter 1: new value in s with an accuracy of 0.1 s
// return value: actual value in s
float fSetvTnJack(float fValue);

// *********************************************************
// Tv of the jacket controller
// Change and query hold-back time of the PID controller with jacket
control.
// ***********************************************************
// return value: actual value in s with an accuracy of 0.1 s
float fGetvTvJack();
// parameter 1: new value in s with an accuracy of 0.1 s
// return value: actual value in s
float fSetvTvJack(float fValue);

// ***********************************************************
// Kp of the process controller
// Change and query amplification of the PID controller with process
control.
// ***********************************************************
// return value: actual value in  with an accuracy of 0.01
float fGetvKpProc();
// parameter 1: new value in  with an accuracy of 0.01
// return value: actual value in
float fSetvKpProc(float fValue);

// ***********************************************************
// Tn of the process controller
// Change and query reset time of the PID controller with process control.
// ***********************************************************
// return value: actual value in s with an accuracy of 0.1 s
float fGetvTnProc();
// parameter 1: new value in s with an accuracy of 0.1 s
// return value: actual value in s
float fSetvTnProc(float fValue);

// ***********************************************************
// Tv of the process controller
// Change and query hold-back time of the PID controller with process
control
// ***********************************************************
// return value: actual value in s with an accuracy of 0.1 s
float fGetvTvProc();
// parameter 1: new value in s with an accuracy of 0.1 s
// return value: actual value in s
float fSetvTvProc(float fValue);

// ***********************************************************
// Pump speed
// Query current pump speed.
// ***********************************************************
// return value: actual value in 1 1/min
int iGetvnP();

// ***********************************************************
// Cooling water temperature
// Current measured value of cooling water entry temperature.
// ************************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fGetvTKwin();

// ************************************************************
// Cooling water pressure
// Current measured value of cooling water pressure.
// ************************************************************
// return value: actual value in 1 mbar
int iGetvpKw();

// ************************************************************
// Power supply conditions
// With some temperature control devices, the user must inform the
thermostat of the power supply values. This is normally done using a menu
entry. This manual setting is recreated here. These variables must be used
with the devices in question. Entering this information in devices that do
not require it is ignored.
// ************************************************************
// return value: actual value of the bit field
int iGetvPowCon();
// parameter 1: new value of the bit field
// return value: actual value of the bit field
int iSetvPowCon(int value);
int iGetvPowCon_Bit0to2_Voltage();
int iSetvPowCon_Bit0to2_Voltage(int value);
int iGetvPowCon_Bit8to9_MaximumCurrent();
int iSetvPowCon_Bit8to9_MaximumCurrent(int value);

// ************************************************************
// Minimum setpoint
// This minimum setpoint is used by the temperature controller.
// ************************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fGetvMinSP();
// parameter 1: new value in °C with an accuracy of 0.01 °C (an accuracy
of 0.001 °C is possible when DV E-grade is activated)
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fSetvMinSP(float fValue);

// ************************************************************
// Maximum setpoint
// This maximum setpoint is used by the temperature controller.
// ************************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an accuracy of 0.001 °C is possible when DV E-grade is activated)
float fGetvMaxSP();
// parameter 1: new value in °C with an accuracy of 0.01 °C (an accuracy of 0.001 °C is possible when DV E-grade is activated)
// return value: actual value in °C with an accuracy of 0.01 °C (an accuracy of 0.001 °C is possible when DV E-grade is activated)
float fSetvMaxSP(float fValue);

// **********************************************************
// Upper level limit
// Setting the fill level upper limit.
// **********************************************************
// return value: actual value in % with an accuracy of 0.1 %
float fGetvNivHi();
// parameter 1: new value in % with an accuracy of 0.1 %
// return value: actual value in %
float fSetvNivHi(float fValue);

// **********************************************************
// Lower level limit
// Setting the fill level lower limit.
// **********************************************************
// return value: actual value in % with an accuracy of 0.1 %
float fGetvNivLo();
// parameter 1: new value in % with an accuracy of 0.1 %
// return value: actual value in %
float fSetvNivLo(float fValue);

// **********************************************************
// Setting level output
// Setting the switching direction for level contacts (optional facility). These are two digital outputs which can be set optionally as opener or closer. One output reports an over-level, the other an under-level.
// **********************************************************
// return value: actual value of the bit field
int iGetvNivCont();
// parameter 1: new value of the bit field
// return value: actual value of the bit field
int iSetvNivCont(int value);
// Bit 0: UnderLevelCloser
// 0: Under-level output as opener. 1: Under-level output as closer.
bool bGetvNivCont_Bit0_UnderLevelCloser();
bool bSetvNivCont_Bit0_UnderLevelCloser(bool value);
// Bit 1: OverLevelCloser
// 0: Over-level output as opener. 1: Over-level output as closer.
bool bGetvNivCont_Bit1_OverLevelCloser();
bool bSetvNivCont_Bit1_OverLevelCloser(bool value);
```

```
// ***********************************************************
// Process temperature
// The actual value of the temperature used by the process controller.
// ***********************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fGetvTProc();

// ***********************************************************
// Status of the thermostat
// Query current thermostat status. The returned value is a bit field. The
individual bits must be considered independently of each other.
// ***********************************************************
// return value: actual value of the bit field
int iGetvStatus2();
// Bit 0: CurrentController
// Control: 1: The process controller currently specifies the control
parameter / 0: The internal controller currently specifies the control
parameter.
bool bGetvStatus2_Bit0_CurrentController();
int iGetvStatus2_Bit1to4_CoolingController();
// Bit 5: DripTrayFull
// The drip tray is full and needs to be emptied (pump seal leak) (only
with special equipment).
bool bGetvStatus2_Bit5_DripTrayFull();
// Bit 6: VPCReferenceOK
// The VPC has completed its reference point approach and you can start
the pump. Message -4137 is displayed on the Pilot ONE® if an attempt is
made to start the pump while bit 6 returns the value 0.
bool bGetvStatus2_Bit6_VPCReferenceOK();
// Bit 7: AirPurgeActive
// Air purge operating mode: 1: active / 0: inactive
bool bGetvStatus2_Bit7_AirPurgeActive();
// Bit 8: AirPurgeFinished
// Air purge process status: 1: finished / 0: in progress
bool bGetvStatus2_Bit8_AirPurgeFinished();
// Bit 9: AirPurgeFeedback
// Air purge feedback: 1: Thermostat signals when air purge is successful
/ 0: no feedback
bool bGetvStatus2_Bit9_AirPurgeFeedback();

// ***********************************************************
// Disturbance feedforward
// With foreseeable load changes, the thermostat can be informed, that it
should adjust its power by the current value.
// ***********************************************************
// return value: actual value in 1 W
int iGetvDistFeed();
// parameter 1: new value in 1 W
// return value: actual value in 1 W
int iSetvDistFeed(int value);

// ***********************************************************
// Pressure in return (absolute)
// Absolute pressure in the return flow of the thermal fluid circuit of
the thermostat.
// ***********************************************************
// return value: actual value in 1 mbar
int iGetvpPin();

// ***********************************************************
// Status Blow-Down
// Status of the optional blow-down device.
// ***********************************************************
// return value: actual value of the bit field
int iGetvBlDwn();
// parameter 1: new value of the bit field
// return value: actual value of the bit field
int iSetvBlDwn(int value);
// Bit 0: CompressedAirOK
// Compressed air OK
bool bGetvBlDwn_Bit0_CompressedAirOK();
// Bit 1: NoOvertemperature
// Overtemperature status OK
bool bGetvBlDwn_Bit1_NoOvertemperature();
// Bit 2: StandbyMode
// Operating mode OK (Standby)
bool bGetvBlDwn_Bit2_StandbyMode();
// Bit 3: TemperatureOK
// Internal temperature is OK (< 70°C)
bool bGetvBlDwn_Bit3_TemperatureOK();
// Bit 4: PumpPressureOK
// Pressure in hydraulic circuit (pump pressure) is OK (< 300mbar).
bool bGetvBlDwn_Bit4_PumpPressureOK();
// Bit 5: ValvePositionOK
// Valve position OK.
bool bGetvBlDwn_Bit5_ValvePositionOK();
// Bit 8: BlowDownActive
// Switch solenoid valve or blow-down process on / off.
bool bGetvBlDwn_Bit8_BlowDownActive();
bool bSetvBlDwn_Bit8_BlowDownActive(bool value);

// ***********************************************************
// Watchdog (fault)
// Watchdog timer for monitoring data communication. Generates a fault
after timeout.
// ***********************************************************
// return value: actual value in 1 s
```

```
int iGetvWD1();
// parameter 1: new value in 1 s
// return value: actual value in 1 s
int iSetvWD1(int value);

// ***********************************************************
// Watchdog (2nd setpoint)
// Watchdog timer for monitoring data communication. Switches to the 2nd
setpoint after timeout.
// ***********************************************************
// return value: actual value in 1 s
int iGetvWD2();
// parameter 1: new value in 1 s
// return value: actual value in 1 s
int iSetvWD2(int value);

// ***********************************************************
// 2nd setpoint
// Setting of the 2nd setpoint.
// ***********************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fGetvSP2();
// parameter 1: new value in °C with an accuracy of 0.01 °C (an accuracy
of 0.001 °C is possible when DV E-grade is activated)
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fSetvSP2(float fValue);

// ***********************************************************
// PMA mode
// The so-called PMA mode allows to override the thermostat's temperature
controller directly specify the desired performance (cooling / heating
capacity).
// ***********************************************************
// return value: actual value in 1
int iGetvPMAMode();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvPMAMode(int value);

// ***********************************************************
// PMA power specified
// Power specification as percentage in PMA mode. Negative values for
cooling capacity, positive values for heating capacity.
// ***********************************************************
// return value: actual value in % with an accuracy of 0.1 %
float fGetvPMA();
// parameter 1: new value in % with an accuracy of 0.1 %
// return value: actual value in %
float fSetvPMA(float fValue);

// ***********************************************************
// Setpoint pump speed
// Set and query the current pump speed setpoint.
// ***********************************************************
// return value: actual value in 1 1/min
int iGetvnPSet();
// parameter 1: new value in 1 1/min
// return value: actual value in 1 1/min
int iSetvnPSet(int value);

// ***********************************************************
// Setpoint pump pressure
// Set and query the current pump pressure setpoint.
// ***********************************************************
// return value: actual value in 1 mbar
int iGetvpPSet();
// parameter 1: new value in 1 mbar
// return value: actual value in 1 mbar
int iSetvpPSet(int value);

// ***********************************************************
// VPC bypass operating mode
// Set and query the operating mode of the VPC bypass.
// ***********************************************************
// return value: actual value in 1
int iGetvVPCMode();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvVPCMode(int value);

// ***********************************************************
// VPC bypass target position
// Specify VPC bypass position or query current setting.
// ***********************************************************
// return value: actual value in % with an accuracy of 0.1 %
float fGetvDesVPCPos();
// parameter 1: new value in % with an accuracy of 0.1 %
// return value: actual value in %
float fSetvDesVPCPos(float fValue);

// ***********************************************************
// Cooling water outflow temperature
// Current measured value of the cooling water outflow temperature.
// ***********************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
```

```
float fGetvTKwOut();

// ***********************************************************
// Thermal fluid volume flow
// Current measured value of the thermal fluid volume flow.
// ***********************************************************
// return value: actual value in l/min with an accuracy of 0.1 l/min (an accuracy of 0.001 l/min is possible when DV E-grade is activated)
float fGetvFluidFlow();

// ***********************************************************
// Setpoint thermal fluid volume flow
// Current setpoint of the thermal fluid volume flow.
// ***********************************************************
// return value: actual value in l/min with an accuracy of 0.1 l/min (an accuracy of 0.001 l/min is possible when DV E-grade is activated)
float fGetvFluidFlowSet();
// parameter 1: new value in l/min with an accuracy of 0.1 l/min (an accuracy of 0.001 l/min is possible when DV E-grade is activated)
// return value: actual value in l/min with an accuracy of 0.1 l/min (an accuracy of 0.001 l/min is possible when DV E-grade is activated)
float fSetvFluidFlowSet(float fValue);

// ***********************************************************
// Setpoint delta-T control
// This setpoint is used by the temperature controller and limits the temperature difference between then internal and the process temperature.
// ***********************************************************
// return value: actual value in K with an accuracy of 0.01 K (an accuracy of 0.001 K is possible when DV E-grade is activated)
float fGetvDeltaT();
// parameter 1: new value in K with an accuracy of 0.01 K (an accuracy of 0.001 K is possible when DV E-grade is activated)
// return value: actual value in K with an accuracy of 0.01 K (an accuracy of 0.001 K is possible when DV E-grade is activated)
float fSetvDeltaT(float fValue);

// ***********************************************************
// Alarm limit delta-T
// Set and query the alarm limit for the difference between the internal and the process temperature.
// ***********************************************************
// return value: actual value in K with an accuracy of 0.01 K (an accuracy of 0.001 K is possible when DV E-grade is activated)
float fGetvDeltaTAlarm();
// parameter 1: new value in K with an accuracy of 0.01 K (an accuracy of 0.001 K is possible when DV E-grade is activated)
// return value: actual value in K with an accuracy of 0.01 K (an accuracy of 0.001 K is possible when DV E-grade is activated)
```

```
float fSetvDeltaTAlarm(float fValue);

// ************************************************************
// Upper alarm limit, internal temperature
// Set and query upper alarm limit for the internal temperature.
// ************************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fGetvTIAlarmHi();
// parameter 1: new value in °C with an accuracy of 0.01 °C (an accuracy
of 0.001 °C is possible when DV E-grade is activated)
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fSetvTIAlarmHi(float fValue);

// ************************************************************
// Lower alarm limit, internal temperature
// Set and query lower alarm limit for the internal temperature.
// ************************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fGetvTIAlarmLo();
// parameter 1: new value in °C with an accuracy of 0.01 °C (an accuracy
of 0.001 °C is possible when DV E-grade is activated)
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fSetvTIAlarmLo(float fValue);

// ************************************************************
// Upper alarm limit, process temperature
// Set and query upper alarm limit for the process temperature.
// ************************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fGetvTEAlarmHi();
// parameter 1: new value in °C with an accuracy of 0.01 °C (an accuracy
of 0.001 °C is possible when DV E-grade is activated)
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fSetvTEAlarmHi(float fValue);

// ************************************************************
// Lower alarm limit, process temperature
// Set and quer lower alarm limit for the process temperature.
// ************************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fGetvTEAlarmLo();
// parameter 1: new value in °C with an accuracy of 0.01 °C (an accuracy
of 0.001 °C is possible when DV E-grade is activated)
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fSetvTEAlarmLo(float fValue);

// ***********************************************************
// Setting of the heating overtemperature protection
// Query the current setting of the overtemperature protection trigger
value of the heating system(s).
// ***********************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fGetvOTHeater();

// ***********************************************************
// Setting of the expansion vessel overtemperature protection.
// Query the current setting of the overtemperature protection trigger
value of the expansion vessel.
// ***********************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fGetvOTExpVessel();

// ***********************************************************
// Starting the temperature control program
// Starting the thermoregulation or query the identifier of the current
thermoregulation program.
// ***********************************************************
// return value: actual value in 1
int iGetvProgramStart();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvProgramStart(int value);

// ***********************************************************
// Specify ramp duration
// Specify ramp duration in seconds and query ramp duration after ramp
start.
// ***********************************************************
// return value: actual value in 1 s
int iGetvRampDuration();
// parameter 1: new value in 1 s
// return value: actual value in 1 s
int iSetvRampDuration(int value);

// ***********************************************************
// Start ramp
// Specifies the end-setpoint and starts a setpoint ramp whose duration
was previously specified using command iSetvRampDuration. The end-setpoint
```

```
of a running setpoint ramp can be queried.
// **********************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fGetvRampStart();
// parameter 1: new value in °C with an accuracy of 0.01 °C (an accuracy
of 0.001 °C is possible when DV E-grade is activated)
// return value: actual value in °C with an accuracy of 0.01 °C (an
accuracy of 0.001 °C is possible when DV E-grade is activated)
float fSetvRampStart(float fValue);

// **********************************************************
// Specify the blow-down mode
// Specify or query the blow-down mode.
// **********************************************************
// return value: actual value in 1
int iGetvBlowDownPos();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvBlowDownPos(int value);

// **********************************************************
// Days left until the next maintenance
// Query the number of days left, before the next maintenance message is
displayed.
// **********************************************************
// return value: actual value in 1 d
int iGetvMaintenanceDays();

// **********************************************************
// Create Service Package
// Saves the service package onto the USB stick.
// **********************************************************
// return value: actual value in 1
int iGetvServicePackage();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvServicePackage(int value);

// **********************************************************
// Change program status
// Changes or queries the current status of a running thermoregulation.
// **********************************************************
// return value: actual value in 1
int iGetvProgramState();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvProgramState(int value);
```

```
// ***********************************************************
// vBoostActive
// vBoostActive
// ***********************************************************
// return value: actual value in 1
int iGetvBoostActive();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvBoostActive(int value);

// ***********************************************************
// vPMAMaster
// vPMAMaster
// ***********************************************************
// return value: actual value in % with an accuracy of 0.1 %
float fGetvPMAMaster();
// parameter 1: new value in % with an accuracy of 0.1 %
// return value: actual value in %
float fSetvPMAMaster(float fValue);

// ***********************************************************
// vpVPC
// vpVPC
// ***********************************************************
// return value: actual value in 1 mbar
int iGetvpVPC();
// parameter 1: new value in 1 mbar
// return value: actual value in 1 mbar
int iSetvpVPC(int value);

// ***********************************************************
// Thermal fluid volume flow actual value setting mode
// Activate, deactivate or query operating mode for setting the thermal
fluid volume flow.
// ***********************************************************
// return value: actual value in 1
int iGetvTFlowMode();

// ***********************************************************
// Actual value setting, Thermal fluid volume flow
// With this, an alternative measured value outside the thermostat can be
specified for the thermal fluid volume flow.
// ***********************************************************
// return value: actual value in l/min with an accuracy of 0.1 l/min (an
accuracy of 0.001 l/min is possible when DV E-grade is activated)
float fGetvTFlowVal();

// ***********************************************************
// Pump control mode
// Set and query the pump control mode of the thermostat.
// ***********************************************************
// return value: actual value in 1
int iGetvPumpCtrlMode();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvPumpCtrlMode(int value);

// ***********************************************************
// Set POKO mode
// Set and query the mode of the floating contact POKO.
// ***********************************************************
// return value: actual value in 1
int iGetvPoKoExtMode();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvPoKoExtMode(int value);

// ***********************************************************
// Change POKO state
// Set and query the state of the floating contact POKO.
// ***********************************************************
// return value: actual value in 1
int iGetvPoKoState();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvPoKoState(int value);

// ***********************************************************
// Current power (high bytes)
// The currently resulting power of the thermostat in watts. Use this
together with command \vPow\.
// ***********************************************************
// return value: actual value in 1 W
int iGetvPowHi();

// ***********************************************************
// Air purge
// Start or stop air purge or query current status.
// ***********************************************************
// return value: actual value in 1
int iGetvAirPurge();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvAirPurge(int value);

// ***********************************************************
// Draining
// Start or stop draining or query current status.
// ***********************************************************
// return value: actual value in 1
int iGetvDrain();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvDrain(int value);

// ***********************************************************
// Setpoint, temperature controller
// The setpoint is used by the temperature controller.
// ***********************************************************
// return value: actual value in °C with an accuracy of 0.01 °C (an accuracy of 0.001 °C is possible when DV E-grade is activated)
float fGetvSPT();
// parameter 1: new value in °C with an accuracy of 0.01 °C (an accuracy of 0.001 °C is possible when DV E-grade is activated)
// return value: actual value in °C with an accuracy of 0.01 °C (an accuracy of 0.001 °C is possible when DV E-grade is activated)
float fSetvSPT(float fValue);

// ***********************************************************
// VPC bypass current position
// Query current position.
// ***********************************************************
// return value: actual value in % with an accuracy of 0.1 %
float fGetvCurVPCPos();
// parameter 1: new value in % with an accuracy of 0.1 %
// return value: actual value in %
float fSetvCurVPCPos(float fValue);

// ***********************************************************
// Message
// The number of the most recent message (alarm, warning or message) to appear.
// ***********************************************************
// return value: actual value in 1
int iGetvMes();
// parameter 1: new value in 1
// return value: actual value in 1
int iSetvMes(int value);

// ***********************************************************
// Disturbance feedforward of VPC flow control
// With foreseeable changes, the thermostat can be informed, that it should adjust the position of the VPC by the current value.
// ***********************************************************
// return value: actual value in % with an accuracy of 0.01 %
float fGetvDistFeedVPC();
// parameter 1: new value in % with an accuracy of 0.01 %
// return value: actual value in %
float fSetvDistFeedVPC(float fValue);
//[[[end]]]

// ***********************************************************
// Connect to Pilot ONE over Ethernet
// ***********************************************************
// parameter 1: IP address of the Pilot ONE
// return value: true if connection is OK, otherwise false
bool bConnect(string IPaddress);

// ***********************************************************
// Connection already established?
// True if connected, otherwise false.
// ***********************************************************
// return value: true if connection is OK, otherwise false
bool bConnected();

// ***********************************************************
// Release connection
// ***********************************************************
void Dispose();
```