AlgalCommand Script Programming Manual

Contents

[Introduction 1](#_Toc358209369)

[Event-Based Scripting 2](#_Toc358209370)

[Simple Scripts 5](#_Toc358209371)

[Script Equivalents to Manual Controls 6](#_Toc358209372)

[Sinusoidal Day-Night Cycle 6](#_Toc358209373)

[Temperature Fluctuations 6](#_Toc358209374)

[pH Control by Gas Injection 6](#_Toc358209375)

[Turbidostat 7](#_Toc358209376)

[Function Reference 8](#_Toc358209377)

[PBR 8](#_Toc358209378)

[Simulation 10](#_Toc358209379)

[Timer 10](#_Toc358209380)

[Time 10](#_Toc358209381)

[Math 11](#_Toc358209382)

[GUI 11](#_Toc358209383)

[ScheduleReader 12](#_Toc358209384)

# Introduction

AlgalCommand allows the user operate the ePBRs in non-standard modes of operation by creating JavaScript files that specify the desired behavior of the ePBR.

**WARNING: It is possible for scripts to access the filesystem of the computer. Take extra care when loading scripts not written by you or a trusted source.**

# Event-Based Scripting

Scripts for the ePBRs are event-based, with functions in the script being called in response to events, such as timing events and data received events. The reason that the scripts are not linear is that a linear script would not easily be able to make decisions based on real-time data from the ePBR and that the ePBRs may generate data independently of the script (such as from clicking “Measure Now” in the manual controls tab). The event-based structure ensures that the script will always be able to intercept data, even if the trigger for that measurement was external to the script or the ePBR has lost connection at the time of the request for data and comes back online at a later time.

Taking a measurement in a script has two components. First, the appropriate measure function is called (such as PBR.measureOD()). Calling the measurement functions will send a command to the ePBR to take that measurement and send back the data at some time in the near future. Usually the data comes back to AlgalCommand within a second of asking for a measurement, but network traffic could introduce a delay. When the data comes back from the ePBR, the data is passed back to the script through one of the data event functions. If the ePBR is not online, the functions to ask for measurements will not fail (so the script will continue to run), but no data will return and thus the data event functions will not be triggered until the ePBR is back online.

All code in the script will be called in response to some event. For code that is not triggered by data or user interaction, use timers to generate timing events. Use Timer.setTimer(X, Y) to generate time events at a periodic interval of Y seconds and then place the repeating code in the timeEvent(X) function. The timeEvent(X) function’s argument will be the number of the timer that was created.

The following figure shows all the events and corresponding functions for scripts.

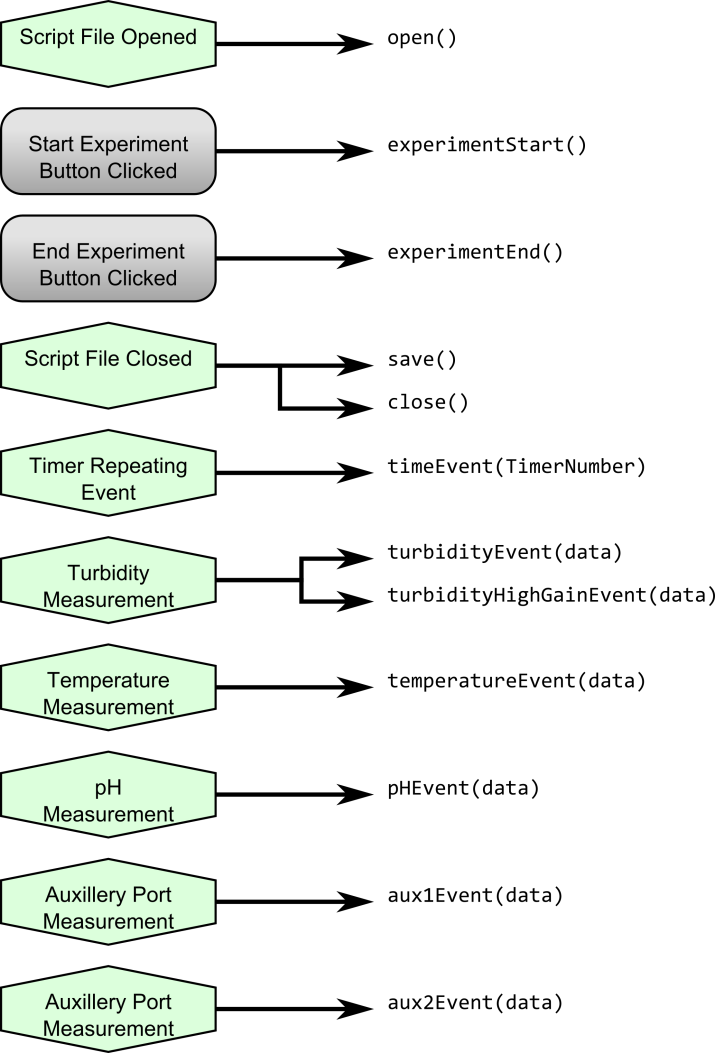


Figure 1: A list of all of the events that can be intercepted by a script and the corresponding functions that those events call.

In a typical script, periodic events are described in the timeEvent(X) function and their timers are created in the open() function. Most scripts will have environment parameter updates as one periodic event and request for measurements as another periodic event. The following figure shows how such a script works:

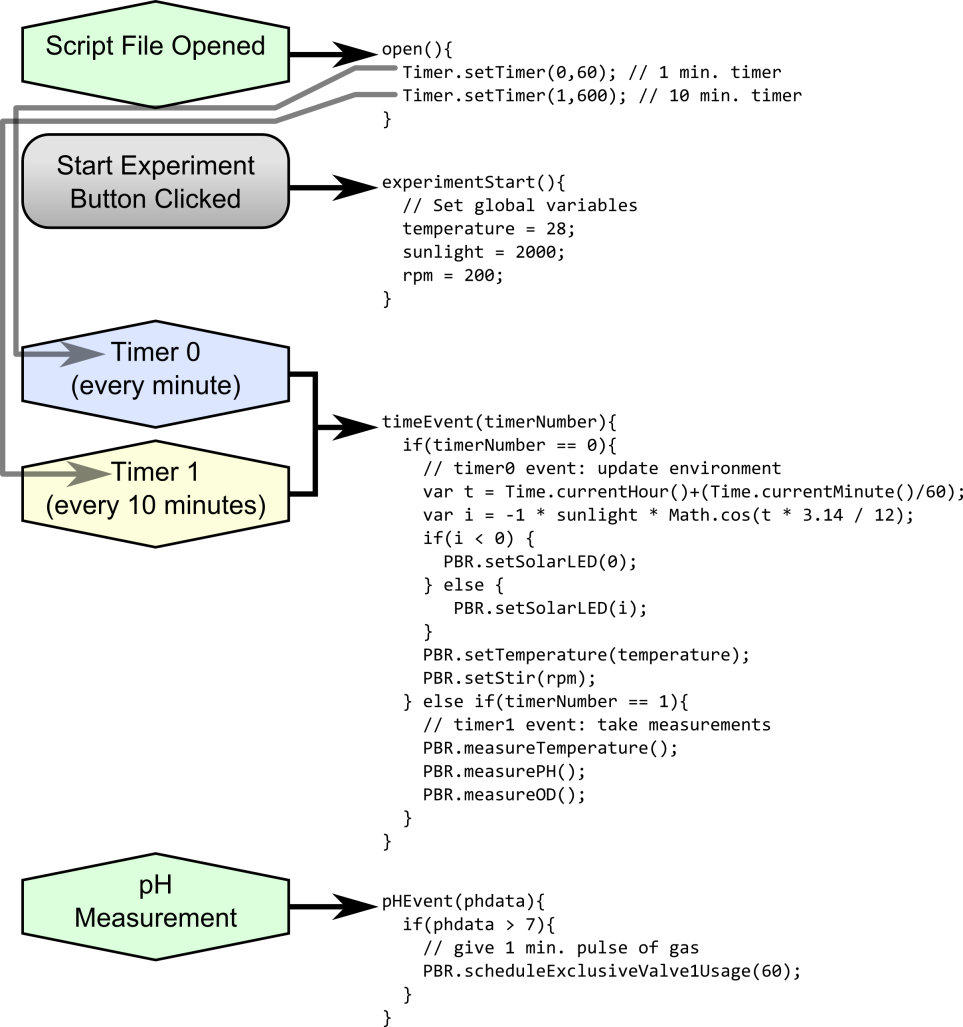


Figure 2: Structure of a typical script file. The open() function is called when the script file is loaded into AlgalCommand. This function creates two repeating timers, which cause a timing event at different intervals (1 minutes for timer 0 and 10 minutes for timer 1). Global variables in the script have their values reset when the user clicks on the “Start Experiment” button. The periodic timers call the timeEvent(…) function and pass in the numerical ID of the triggering timer. This script updates the environment on timer 0 (every minute) and takes measurements on timer 1 (every 10 minutes). When the ePBR returns pH data, the pHEvent(…) function is called, which triggers a dose of gas injection if the pH was measured to be above 7.

# Simple Scripts

The following is a simple script that you may use as a template for creating more advanced scripts. It uses 3 timers to update the light and temperature of the ePBR every minute, take measurements every 10 minutes, and give 2 minutes of sparging every hour. The functions to handle received data are blank because this script does not make decisions based on real time data (the measurements are automatically logged, so no scripting is required to save the data).

// global variables and default values

// (values of global variables are saved when script is

// closed and opened again, so the values should be

// reset in the experimentStart() function).

var peakIntensity = 0;

var temperature = 20;

// called when this file is loaded

function open() {

Timer.setTimer(1, 1 \* 60); // 1 min. interval

Timer.setTimer(2,10 \* 60); // 10 min. interval

Timer.setTimer(3,60 \* 60); // 1 hr interval

PBR.setStir(200);

}

// called when the "Start Experiment" button is clicked

function experimentStart() {

peakIntensity = 2000;

temperature = 28;

}

// called when the "End Experiment" button is clicked

function experimentEnd() {

peakIntensity = 0;

PBR.stopTemperatureControl();

}

function timeEvent(timerID){

if(timerID == 1){

var daylength = 16;

var intensity = peakIntensity \* Simulation.sinusoidalDayFactor(daylength);

PBR.setSolarLED(intensity);

PBR.setTemperature(temperature);

} else if(timerID == 2){

PBR.measureOD();

PBR.measureTemperature();

PBR.measurePH();

} else if(timerID == 3){

PBR.scheduleExclusiveValve1Usage(120,10); // 10s purge, 2min. sparge

}

}

// called when the script file is closed (e.g. when AlgalCommand is closed)

function save() {

//

}

// called when the script file is closed (e.g. when AlgalCommand is closed)

function close() {

//

}

// called when OD data is received

function turbidityEvent(opticalDensity) {

//

}

// called when OD data is received

function turbidityHighGainEvent(opticalDensityHG){

//

}

// called when temperature data is received

function temperatureEvent(degreesC){

//

}

// called when pH data is received

function pHEvent(pH){

//

}

// called when aux ports are measured (this feature is not used by default)

function aux1Event(voltage){

//

}

// called when aux ports are measured (this feature is not used by default)

function aux2Event(voltage){

//

}

# Script Equivalents to Manual Controls

### Sinusoidal Day-Night Cycle

The Simulation.sinusoidalDayFactor(dayLength) function uses the same formula as the manual controls for calculating the sinusoidal approximation for a day-night cycle. The following code is an example that uses this function.

function timeEvent(timerID){

if(timerID == 1){

var daylength = 16;

var intensity = peakIntensity \* Simulation.sinusoidalDayFactor(daylength);

PBR.setSolarLED(intensity);

}

}

### Temperature Fluctuations

The temperature fluctuation control in the manual controls is a very simplistic sine wave to emulate a pond getting warmer during the day and cooler at night. You may wish to phase-shift this behavior. The following code implements the same logic as the manual control with a phaseShift variable to shift the phase of the sine wave.

function timeEvent(timerID){

if(timerID == 1){

var tempAve = 20;

var tempAmp = 5;

var phaseShift = 0;

var timeOfDay = Simulation.currentTimeOfDayInHours() - phaseShift;

var temperature = tempAve - tempAmp \* Math.sin(timeOfDay \* Math.PI / 12);

PBR.setTemperature(temperature);

}

}

### pH Control by Gas Injection

There are several ways to implement pH control by gas injection and the default control may not be suitable for your gas system. In the manual controls, pH control follows simple logic: when the pH is greater than [target + tolerance], the gas valve is turned on until the pH is below [target - tolerance]. This logic ignores the state of other ePBRs, so it does not prevent multiple ePBRs from sparging at the same time. This manual controls are equivalent to this script code:

function pHEvent(pH){

var target = 7.5;

var tolerance = 0.75;

if (pH > (target + tolerance)){

PBR.setValve1(true);

} else if (pH < (target - tolerance)){

PBR.setValve1(false);

}

}

If you wish to administer the sparges as 1 minute doses and did not want more than one ePBR to sparge at the same time, use this code instead:

function pHEvent(pH){

var target = 7.5;

if (pH > target){

PBR.scheduleExclusiveValve1Usage(60);

}

}

### Turbidostat

Running a turbidostat requires that you either blank the vessel or otherwise use some reference point. The following code illustrates a turbidostat that tries to maintain the optical density at the time that the “Start Experiment” button was last clicked. If the culture is thick, move the code from turbidityEvent(...) to turbidityHighGainEvent(...).

var setThreshold = false;

var odTarget = 100;

PBR.setAux2(false);

function experimentStart() {

setThreshold = true;

PBR.measureOD();

}

function turbidityEvent(opticalDensity) {

if (setThreshold == true){

odTarget = opticalDensity;

setThreshold = false;

} else {

if (opticalDensity > odTarget){

PBR.triggerPump(1);

}

}

}

# Function Reference

Below is a list of functions provided by AlgalCommand for scripting. The scripting engine also allows the use of standard Java objects (e.g. java.lang.Thread) in the script file, but this is not recommended.

## PBR

**PBR.flashLED()**

Blinks the solar LED of the ePBR several times.

**PBR.logCustomData(measurementName, dataValue, unitOfMeasurement)**

Logs custom data as if it was generated by the ePBR. The data will be graphed in the ePBR control window and

**PBR.measureOD()**

Tells the ePBR to take a turbidity measurement.

**PBR.measureODNow()**

Tells the ePBR to override the rolling average and take an immediate turbidity measurement.

**PBR.measurePH()**

Tells the ePBR to take a pH measurement

**PBR.measureTemperature()**

Tells the ePBR to take a temperature measurement

**PBR.measureTransmission()**

Tells the PBR to take an optical density measurement and return the raw voltage.

**PBR.measureTransmissionNow()**

Tells the PBR to override the rolling average and take an optical density measurement and return the raw voltage.

**PBR.scheduleExclusiveValve1Usage(secondsOn)**

Causes the ePBR to receive a pulse of gas through valve 1 at some time in the future, where the duration of the pulse is equal to the provided number. AlgalCommand will coordinate the operation of the gas valves so that if multiple ePBRs use this function at the same time, the gas will be delivered to only 1 ePBR at a time, roughly in the order that they called this function.

**PBR.scheduleExclusiveValve1Usage(secondsOn, purgeTime)**

Same as PBR.scheduleExclusiveValve1Usage(secondsOn), except that valve 2 will be opened for a pulse immediately before opening valve 1. Use this function to depressurize the gas system if pressure builds up in your gas system while all ePBRs are not using the gas.

**PBR.scheduleExclusiveValve2Usage(secondsOn)**

Causes the ePBR to receive a pulse of gas through valve 2 at some time in the future, where the duration of the pulse is equal to the provided number. AlgalCommand will coordinate the operation of the gas valves so that if multiple ePBRs use this function at the same time, the gas will be delivered to only 1 ePBR at a time, roughly in the order that they called this function.

**PBR.setAux1(true|false)**

Call PBR.setAux1(true) to turn on and PBR.setAux1(false) the power to the aux1 (light heater) port. Aux ports are **on** by default.

**PBR.setAux2(true|false)**

Call PBR.setAux2(true) to turn on and PBR.setAux2(false) the power to the aux2 (aux) port. Aux ports are **on** by default.

**PBR.setSolarLED(intensity)**

Sets the light intensity of the illumination LED

**PBR.setSolarLEDVoltage(voltage)**

Sets the raw voltage setting on the control circuit for the LED. *This is not the voltage output to the LED.* The LED is on a current-regulating circuit whose current is proportional to the control voltage.

**PBR.setStir(rpm)**

Sets the speed of the stirring motor.

**PBR.setTemperature(degreesC)**

Sets the temperature control target for the ePBR.

**PBR.setValve1(true|false)**

Call PBR.setValve1(true) to open gas valve 1 and PBR.setValve1(false) to close it.

**PBR.setValve2(true|false)**

Call PBR.setValve2(true) to open gas valve 2 and PBR.setValve1(false) to close it.

**PBR.stopTemperatureControl()**

Tells the PBR to stop regulating the temperature

**PBR.triggerPump(secondsOn)**

Tells the PBR to trigger a dose of dilution media from the turbidostat pump. The trigger signal will be on for the given number of seconds (1 second is good for edge-triggered pumps).

## Simulation

**Simulation.currentTimeOfDayInHours()**

returns the time of day in hours from 0 to 24

**Simulation.sinusoidalDayFactor(dayLength)**

Returns a number from 0 to 1 specifying what the light intensity should be based on the time of day and day length using a sinusoidal approximation (same calculation as the sinusoidal day setting in the control window). Multiply this number by the peak light intensity to replicate the behavior of the manual controls.

**Simulation.sinusoidalDayFactor(timeInHours, dayLength)**

Same as sinusoidalDayFactor(dayLength), but uses the provided number as the time of day instead of the system clock.

## Timer

**Timer.setTimer(idNumber, interval)**

Sets up a timer to trigger the timeEvent(id) function at a specified interval. The interval is specified in seconds.

**Timer.removeTimer(idNumber)**

Deletes a timer that was previously created with Timer.setTimer(idNumber, interval).

**Timer.removeAllTimers()**

Deletes all timers.

## Time

**Time.currentDayOfMonth()**

Returns the current day of the month

**Time.currentDayOfYear()**

Returns the current day of the year

**Time.currentHour()**

Returns the current hour in the day

**Time.currentMinute()**

Returns the current minute

**Time.currentMonth()**

Returns the current month of the year

**Time.currentSecond()**

Returns the current second

**Time.currentTimeInDays()**

Returns the raw system time (time since 1970), converted to days

**Time.currentTimeInHours()**

Returns the raw system time (time since 1970), converted to hours

**Time.currentTimeInMinutes()**

Returns the raw system time (time since 1970), converted to minutes

**Time.currentTimeInYears()**

Returns the raw system time (time since 1970), converted to years

**Time.currentYear()**

Returns the current year

**Time.getTimeStamp()**

Returns a text representation of the current time

## Math

See <http://www.w3schools.com/jsref/jsref_obj_math.asp>

## GUI

**GUI.getInput(name, initialValue)**

Launches a pop-up asking for an input value and returns either the new value (if the user clicked OK) or the initial value (if the user clicked cancel). For example:

useSpecial = GUI.getInput(“Use special function?”, false);

setting = GUI.getInput(“What value to set?”, 1500);

Only supports Strings, Numbers, and booleans.

## ScheduleReader

The ScheduleReader class is provided for parsing tab-delimited files describing the environmental settings of the ePBR over time. The first line of the schedule text file contains the headers for the columns (required). One of the columns must be TIME, which is formatted as hour:minute:second. The hour portion is hours from start of experiment, not time of day. The schedule does not automatically repeat itself. Typical usage in script:

function open() {

Timer.setTimer(1, 60);

}

function experimentStart() {

startTime = ScheduleReader.currentTime();

scheduleFile = ""+ScheduleReader.askForFile("Choose schedule file");

}

function timeEvent(timerID){

if(timerID == 1){

if(scheduleFile != null && startTime != null){

if(schedule == null){

schedule = new ScheduleReader(scheduleFile);

}

schedule.setPBR(PBR,ScheduleReader.timeSince(startTime));

// use “schedule.setPBR(PBR, ScheduleReader.timeOfDayInMilliseconds());”

// for repeating 24hr schedule

}

}

}

Typical contents of schedule file:

TIME LINT TEMP STIR

0:00:00 0 23.4 0

4:00:00 0 22.4 0

8:00:00 414.3 25.3 200

12:00:00 1696 27.3 200

16:00:00 1017 27.4 200

20:00:00 0 25.7 0

24:00:00 0 24.4 0

28:00:00 0 22.6 0

32:00:00 603.2 25.2 200

36:00:00 1572 27.2 200

40:00:00 941.4 26.8 200

44:00:00 0 25.4 0

48:00:00 0 24.8 0

To make the schedule repeat every 24 hours, replace schedule.setPBR(PBR,ScheduleReader.timeSince(startTime)); with schedule.setPBR(PBR, ScheduleReader.timeOfDayInMilliseconds());

**ScheduleReader.askForFile()**

Shows a pop-up asking the user to select a file.

**ScheduleReader.currentTime()**

Returns the current time, expressed in computer standard millisecond time (milliseconds since midnight on Jan. 1st, 1970).

**\_\_\_\_\_.setPBR(PBR, mstime)**

Sets the environmental properties of the given PBR using interpolated values from the loaded schedule file. The variable mstime refers to the time since the start of the experiment, in milliseconds. This function can only be called on a variable that was assigned the value of a new ScheduleReader instance:

schedule = new ScheduleReader(scheduleFile);

…

schedule.setPBR(PBR,ScheduleReader.timeSince(startTime));

**ScheduleReader.timeOfDayInMilliseconds()**

Returns the number of milliseconds since midnight.

**ScheduleReader.timeSince(initialTime)**

Returns the amount of time since the given time point, in milliseconds.