Программа для ЭВМ

«Программный Комплекс Автоматизированное Рабочее Место «Умная Среда MTM IoT»

Исходный текст программы

Правообладатель: Общество с ограниченной ответственностью Инжиниринговая Компания «Велес»

Автор:

Игнатьев Валерий Викторович

```
std::map<std::string, std::map<int, time_t>> *getDefaultAstroEvents(time_t currentTime, double lon, double lat) {
// получаем три даты для выборки
char *tmpStr = new char[11];
std::tm tmpTm = {0};
std::map<int, std::string> actualDates;
localtime_r(&currentTime, &tmpTm);
// вчера
tmpTm.tm_mday--;
std::strftime(tmpStr, 11, "%F", &tmpTm);
actualDates.insert({0, tmpStr});
// сегодня
tmpTm.tm_mday++;
std::strftime(tmpStr, 11, "%F", &tmpTm);
actualDates.insert({1, tmpStr});
// завтра
tmpTm.tm_mday++;
std::strftime(tmpStr, 11, "%F", &tmpTm);
actualDates.insert({2, tmpStr});
auto *defaultAstroEvents = new std::map<std::string, std::map<int, time_t>>;
// зарание строим массив с расчитанными значениями, которые далее будем переопределять данными полученными из базы
localtime_r(&currentTime, &tmpTm);
// вчерашняя дата
tmpTm.tm_mday--;
time_t tmpTime = std::mktime(&tmpTm);
auto it = actualDates.begin();
while (it != actualDates.end()) {
auto second = it->second;
double r, s;
double twb, twe;
localtime_r(&tmpTime, &tmpTm);
sun_rise_set(tmpTm.tm_year + 1900, tmpTm.tm_mon + 1, tmpTm.tm_mday, lon, lat, &r, &s);
fillTimeStruct(r, &tmpTm);
```

```
(*defaultAstroEvents)[second][0] = mktime(&tmpTm);
fillTimeStruct(s, &tmpTm);
(*defaultAstroEvents)[second][1] = mktime(&tmpTm);
civil_twilight(tmpTm.tm_year + 1900, tmpTm.tm_mon + 1, tmpTm.tm_mday, lon, lat, &twb,
&twe);
fillTimeStruct(twb, &tmpTm);
(*defaultAstroEvents)[second][2] = mktime(&tmpTm);
fillTimeStruct(twe, &tmpTm);
(*defaultAstroEvents)[second][3] = mktime(&tmpTm);
tmpTm.tm_mday++;
tmpTime = std::mktime(&tmpTm);
it++;
}
it = actualDates.begin();
auto next = actualDates.begin();
next++;
while (next != actualDates.end()) {
auto prevDay = it->second;
auto currDay = next->second;
uint16_t nightLenght;
uint16_t twilightLength;
// длительность ночи восход сегодня - закат вчера
nightLenght = (*defaultAstroEvents)[currDay][0] - (*defaultAstroEvents)[prevDay][1];
// длительность ночи со вчера на сегодня
(*defaultAstroEvents)[prevDay][4] = nightLenght;
// длительность вчерашних сумерек конец сумерек - закат
twilightLength = (*defaultAstroEvents)[prevDay][3] - (*defaultAstroEvents)[prevDay][1];
// длительность вчерашних сумерек
(*defaultAstroEvents)[prevDay][5] = twilightLength;
it++;
next++;
}
return defaultAstroEvents;
}
void checkAstroEvents(time_t currentTime, double lon, double lat, DBase *dBase, int32_t
```

```
threadId) {
struct tm ctm = {0};
struct tm tmp_tm = {0};
double rise, set;
double twilightStart, twilightEnd;
int rs;
int civ;
uint64_t sunRiseTime;
uint64_t sunSetTime;
uint64_t twilightStartTime;
uint64_t twilightEndTime;
uint64_t twilightLength;
uint64_t nightLength;
uint64_t calcNightLength;
double nightRate;
mtm_cmd_action action = {0};
MYSQL_RES *res;
MYSQL_ROW row;
std::string query;
localtime_r(&currentTime, &ctm);
rs = sun_rise_set(ctm.tm_year + 1900, ctm.tm_mon + 1, ctm.tm_mday, lon, lat, &rise, &set);
bool isTimeAboveSunSet;
bool isTimeLessSunSet;
bool isTimeAboveSunRise;
bool isTimeLessSunRise;
civ = civil_twilight(ctm.tm_year + 1900, ctm.tm_mon + 1, ctm.tm_mday, lon, lat,
&twilightStart,
&twilightEnd);
bool isTimeAboveTwilightStart;
bool isTimeLessTwilightStart;
bool isTimeAboveTwilightEnd;
bool isTimeLessTwilightEnd;
localtime_r(&currentTime, &tmp_tm);
// расчитываем длительность сумерек по реальным данным восхода и начала сумерек
fillTimeStruct(rise, &tmp_tm);
```

```
sunRiseTime = mktime(&tmp_tm);
fillTimeStruct(twilightStart, &tmp_tm);
twilightStartTime = mktime(&tmp_tm);
twilightLength = sunRiseTime - twilightStartTime;
// расчитываем реальную длительность ночи, с сумерками
fillTimeStruct(set, &tmp_tm);
sunSetTime = mktime(&tmp_tm);
nightLength = 86400 - (sunSetTime - sunRiseTime);
// пытаемся получить данные из календаря
sunRiseTime = 0;
sunSetTime = 0;
query.append(
"SELECT unix_timestamp(nct.date) AS time, type FROM node_control AS nct WHERE
DATE(nct.date)=CURRENT_DATE()");
res = dBase->sqlexec(query.data());
if (res != nullptr) {
dBase->makeFieldsList(res);
while ((row = mysql_fetch_row(res)) != nullptr) {
if (std::stoi(row[dBase->getFieldIndex("type")]) == 0) {
sunRiseTime = std::stoull(row[dBase->getFieldIndex("time")]);
} else if (std::stoi(row[dBase->getFieldIndex("type")]) == 1) {
sunSetTime = std::stoull(row[dBase->getFieldIndex("time")]);
}
}
mysql_free_result(res);
}
if (rs == 0 && civ == 0) {
if (sunRiseTime == 0) {
fillTimeStruct(rise, &tmp_tm);
sunRiseTime = mktime(&tmp_tm);
}
if (sunSetTime == 0) {
fillTimeStruct(set, &tmp_tm);
sunSetTime = mktime(&tmp_tm);
}
```

```
// расчитываем коэффициент как отношение расчитаной длительности ночи к реальной
calcNightLength = 86400 - (sunSetTime - sunRiseTime);
nightRate = (double) calcNightLength / nightLength;
// расчитываем время начала/конца сумерек относительно рассвета/заката (которые возможно получили из календаря)
// устанавливая их длительность пропорционально изменившейся длительности ночи
twilightStartTime = sunRiseTime - (uint64_t)(twilightLength * nightRate);
twilightEndTime = sunSetTime + (uint64_t)(twilightLength * nightRate);
action.header.type = MTM_CMD_TYPE_ACTION;
action.header.protoVersion = MTM_VERSION_0;
action.device = MTM_DEVICE_LIGHT;
isTimeAboveSunSet = currentTime >= sunSetTime;
isTimeLessSunSet = currentTime < sunSetTime;
isTimeAboveSunRise = currentTime >= sunRiseTime;
isTimeLessSunRise = currentTime < sunRiseTime;
isTimeAboveTwilightStart = currentTime >= twilightStartTime;
isTimeLessTwilightStart = currentTime < twilightStartTime;
isTimeAboveTwilightEnd = currentTime >= twilightEndTime;
isTimeLessTwilightEnd = currentTime < twilightEndTime;
if ((isTimeAboveSunSet && isTimeLessTwilightEnd) && (!isSunSet || !isSunInit)) {
isSunInit = true;
isSunSet = true;
isTwilightEnd = false;
isTwilightStart = false;
isSunRise = false;
// включаем контактор
switchContactor(true, MBEE_API_DIGITAL_LINE7);
char message[1024];
sprintf(message, "Наступил закат, включаем реле контактора.");
kernel->log.ulogw(LOG_LEVEL_ERROR, "[%s] %s", TAG, message);
AddDeviceRegister(dBase, (char *) coordinatorUuid.data(), message);
// даём задержку для того чтоб стартанули модули в светильниках
// т.к. неизвестно, питаются они через контактор или всё время под напряжением
sleep(5);
// зажигаем светильники
```

```
ssize_t rc;
// rc = switchAllLight(100);
// if (rc == -1) {
// kernel->log.ulogw(LOG_LEVEL_ERROR, "[%s] ERROR write to port", TAG);
// // останавливаем поток с целью его последующего автоматического запуска и
инициализации
// mtmZigbeeStopThread(dBase, threadId);
// AddDeviceRegister(dBase, (char *) coordinatorUuid.data(),
// (char *) "Ошибка записи в порт координатора");
// return;
// }
// передаём команду "астро событие" "закат"
action.data = (0x02 << 8 | 0x01); // NOLINT(hicpp-signed-bitwise)
rc = send_mtm_cmd(coordinatorFd, 0xFFFF, &action, kernel);
if (rc == -1) {
lostZBCoordinator(dBase, threadId, &coordinatorUuid);
return;
}
if (kernel->isDebug) {
kernel->log.ulogw(LOG_LEVEL_INFO, "[%s] rc=%ld", TAG, rc);
kernel->log.ulogw(LOG_LEVEL_INFO, "[%s] закат", TAG);
}
} else if ((isTimeAboveTwilightEnd || isTimeLessTwilightStart) && (!isTwilightEnd || !
isSunInit)) {
isSunInit = true;
isSunSet = false;
isTwilightEnd = true;
isTwilightStart = false;
isSunRise = false;
// включаем контактор
switchContactor(true, MBEE_API_DIGITAL_LINE7);
char message[1024];
sprintf(message, "Наступил конец сумерек, включаем реле контактора.");
kernel->log.ulogw(LOG_LEVEL_ERROR, "[%s] %s", TAG, message);
// AddDeviceRegister(dBase, (char *) coordinatorUuid.data(), message);
```

```
// даём задержку для того чтоб стартанули модули в светильниках
// т.к. неизвестно, питаются они через контактор или всё время под напряжением
sleep(5);
// передаём команду "астро событие" "конец сумерек"
action.data = (0x01 << 8 | 0x00); // NOLINT(hicpp-signed-bitwise)
ssize_t rc = send_mtm_cmd(coordinatorFd, 0xFFFF, &action, kernel);
if (rc == -1) {
lostZBCoordinator(dBase, threadId, &coordinatorUuid);
return;
}
if (kernel->isDebug) {
kernel->log.ulogw(LOG_LEVEL_INFO, "[%s] rc=%ld", TAG, rc);
kernel->log.ulogw(LOG_LEVEL_INFO, "[%s] конец сумерек", TAG);
}
} else if ((isTimeAboveTwilightStart && isTimeLessSunRise) && (!isTwilightStart || !
isSunInit)) {
isSunInit = true;
isSunSet = false;
isTwilightEnd = false;
isTwilightStart = true;
isSunRise = false;
// включаем контактор
switchContactor(true, MBEE_API_DIGITAL_LINE7);
char message[1024];
sprintf(message, "Наступило начало сумерек, включаем реле контактора.");
kernel->log.ulogw(LOG_LEVEL_ERROR, "[%s] %s", TAG, message);
// AddDeviceRegister(dBase, (char *) coordinatorUuid.data(), message);
// даём задержку для того чтоб стартанули модули в светильниках
// т.к. неизвестно, питаются они через контактор или всё время под напряжением
sleep(5);
// передаём команду "астро событие" "начало сумерек"
action.data = (0x03 << 8 | 0x00); // NOLINT(hicpp-signed-bitwise)
ssize_t rc = send_mtm_cmd(coordinatorFd, 0xFFFF, &action, kernel);
if (rc == -1) {
lostZBCoordinator(dBase, threadId, &coordinatorUuid);
```

```
return;
}
if (kernel->isDebug) {
kernel->log.ulogw(LOG_LEVEL_INFO, "[%s] rc=%ld", TAG, rc);
kernel->log.ulogw(LOG_LEVEL_INFO, "[%s] начало сумерек", TAG);
}
} else if ((isTimeAboveSunRise && isTimeLessSunSet) && (!isSunRise || !isSunInit)) {
isSunInit = true;
isSunSet = false;
isTwilightEnd = false;
isTwilightStart = false;
isSunRise = true;
// выключаем контактор, гасим светильники, отправляем команду "восход"
switchContactor(false, MBEE_API_DIGITAL_LINE7);
char message[1024];
sprintf(message, "Наступил восход, выключаем реле контактора.");
kernel->log.ulogw(LOG_LEVEL_ERROR, "[%s] %s", TAG, message);
AddDeviceRegister(dBase, (char *) coordinatorUuid.data(), message);
// на всякий случай, если светильники всегда под напряжением
switchAllLight(0);
// передаём команду "астро событие" "восход"
action.data = (0x00 << 8 | 0x00); // NOLINT(hicpp-signed-bitwise)
ssize_t rc = send_mtm_cmd(coordinatorFd, 0xFFFF, &action, kernel);
if (rc == -1) {
lostZBCoordinator(dBase, threadId, &coordinatorUuid);
return;
}
if (kernel->isDebug) {
kernel->log.ulogw(LOG_LEVEL_INFO, "[%s] rc=%ld", TAG, rc);
kernel->log.ulogw(LOG_LEVEL_INFO, "[%s] восход", TAG);
}
} else {
// ситуация когда мы не достигли условий переключения состояния светильников
// такого не должно происходить
}
```

```
} else {
}
}
void mtmZigbeePktListener(DBase *dBase, int32_t threadId) {
bool run = true;
int64_t count;
uint32_t i = 0;
uint8_t data;
uint8_t seek[1024];
//---
bool isSof = false;
bool isFrameLen = false;
uint8_t frameLen = 0;
bool isCommand = false;
uint16_t commandByteCount = 0;
bool isFrameData = false;
uint8_t frameDataByteCount = 0;
uint8_t fcs;
time_t currentTime, heartBeatTime = 0, syncTimeTime = 0, checkSensorTime = 0,
checkAstroTime = 0,
checkOutPacket = 0, checkCoordinatorTime = 0, checkLinkState = 0;
struct tm *localTime;
struct zb_pkt_item {
// zigbee_frame frame;
void *pkt;
uint32_t len;
SLIST_ENTRY(zb_pkt_item) items;
};
// struct zb_queue *zb_queue_ptr;
SLIST_HEAD(zb_queue, zb_pkt_item)
zb_queue_head = SLIST_HEAD_INITIALIZER(zb_queue_head);
SLIST_INIT(&zb_queue_head);
// zb_queue_ptr = (struct zb_queue *) (&zb_queue_head);
struct zb_pkt_item *zb_item;
mtmZigbeeSetRun(true);
```

```
while (run) {
count = read(coordinatorFd, &data, 1);
if (count > 0) {
// printf("data: %02X\n", data);
// TODO: сделать вложенные if
// начинаем разбор
if (!isSof && data == SOF) {
i = 0;
isSof = true;
seek[i++] = data;
if (kernel->isDebug) {
// kernel->log.ulogw(LOG_LEVEL_INFO, "[%s] found SOF", TAG);
}
} else if (!isFrameLen) {
isFrameLen = true;
seek[i++] = frameLen = data;
if (kernel->isDebug) {
// kernel->log.ulogw(LOG_LEVEL_INFO, "[%s] found frame len", TAG);
}
} else if (!isCommand) {
commandByteCount++;
seek[i++] = data;
if (commandByteCount == 2) {
commandByteCount = 0;
isCommand = true;
if (kernel->isDebug) {
// kernel->log.ulogw(LOG_LEVEL_INFO, "[%s] found command", TAG);
}
}
} else if (!isFrameData && frameDataByteCount < frameLen) {
seek[i++] = data;
frameDataByteCount++;
if (frameDataByteCount == frameLen) {
isFrameData = true;
frameDataByteCount = 0;
```

```
if (kernel->isDebug) {
// kernel->log.ulogw(LOG_LEVEL_INFO, "[%s] found frame data", TAG);
}
}
} else {
// нашли контрольную сумму
seek[i++] = data;
if (kernel->isDebug) {
// kernel->log.ulogw(LOG_LEVEL_INFO, "[%s] found FCS", TAG);
}
// пакет вроде как разобран
// нужно проверить контрольную сумму фрейма
fcs = compute_fcs(seek, i);
if (fcs == seek[i - 1]) {
if (kernel->isDebug) {
// kernel->log.ulogw(LOG_LEVEL_INFO, "[%s] frame good", TAG);
}
// складываем полученный пакет в список
zb_item = (struct zb_pkt_item *) malloc(sizeof(struct zb_pkt_item));
zb_item->len = i;
zb_item->pkt = malloc(zb_item->len);
memcpy(zb_item->pkt, seek, zb_item->len);
SLIST_INSERT_HEAD(&zb_queue_head, zb_item, items);
} else {
if (kernel->isDebug) {
// kernel->log.ulogw(LOG_LEVEL_ERROR, "[%s] frame bad", TAG);
}
// вероятно то что попадает в порт с модуля zigbee уже проверено им самим
// как проверить это предположение? попробовать послать порченый пакет.
// либо он не будет отправлен, либо не попадёт в порт т.к. порченый, либо попадёт
в порт мне на обработку
// считаем что такое не возможно - проверить
}
// сбрасываем состояние алгоритма разбора пакета zigbee
isSof = false;
```

```
isFrameLen = false;
isCommand = false;
isFrameData = false;
i = 0;
}
} else {
// есть свободное время, разбираем список полученных пакетов
while (!SLIST_EMPTY(&zb_queue_head)) {
if (kernel->isDebug) {
kernel->log.ulogw(LOG_LEVEL_INFO, "[%s] processing zb packet...", TAG);
}
zb_item = SLIST_FIRST(&zb_queue_head);
mtmZigbeeProcessInPacket((uint8_t *) zb_item->pkt, zb_item->len);
SLIST_REMOVE_HEAD(&zb_queue_head, items);
free(zb_item->pkt);
free(zb_item);
}
// проверяем, не отключили ли запуск потока, если да, остановить выполнение
// обновляем значение c_time в таблице thread раз в 5 секунд
currentTime = time(nullptr);
if (currentTime - heartBeatTime >= 5) {
heartBeatTime = currentTime;
char query[512] = {0};
MYSQL_RES *res;
MYSQL_ROW row;
my_ulonglong nRows;
int isWork = 0;
sprintf(query, "SELECT * FROM threads WHERE _id = %d", threadId);
res = mtmZigbeeDBase->sqlexec(query);
if (res) {
nRows = mysql_num_rows(res);
if (nRows == 1) {
mtmZigbeeDBase->makeFieldsList(res);
row = mysql_fetch_row(res);
if (row != nullptr) {
```

```
isWork = std::stoi(row[mtmZigbeeDBase->getFieldIndex("work")]);
} else {
// ошибка получения записи из базы, останавливаем поток
kernel->log.ulogw(LOG_LEVEL_ERROR, "[%s] Read thread record get null",
TAG);
kernel->log.ulogw(LOG_LEVEL_ERROR, "[%s] Stopping thread", TAG);
mysql_free_result(res);
return;
}
} else {
// записи о потоке нет, либо их больше одной, останавливаем поток
kernel->log.ulogw(LOG_LEVEL_ERROR, "[%s] Thread record not single, or not
exists", TAG);
kernel->log.ulogw(LOG_LEVEL_ERROR, "[%s] Stopping thread", TAG);
mysql_free_result(res);
return;
}
mysql_free_result(res);
if (isWork == 1) {
// обновляем статус
UpdateThreads(*mtmZigbeeDBase, threadId, 0, 1, nullptr);
} else {
// поток "остановили"
sprintf(query, "UPDATE threads SET status=%d,
changedAt=FROM_UNIXTIME(%lu) WHERE _id=%d", 0,
currentTime, threadId);
res = mtmZigbeeDBase->sqlexec(query);
mysql_free_result(res);
kernel->log.ulogw(LOG_LEVEL_ERROR, "[%s] Thread stopped from GUI",
TAG);
kernel->log.ulogw(LOG_LEVEL_ERROR, "[%s] Stopping thread", TAG);
return;
}
}
}
```

```
// рассылаем пакет с текущим "временем" раз в 10 секунд
currentTime = time(nullptr);
if (currentTime - syncTimeTime >= 10) {
// В "ручном" режиме пакет со времменем не рассылаем, т.к. в нём передаётся
уровень диммирования для
// каждой группы. При этом какое бы значение мы не установили по умолчанию,
оно "затрёт" установленное
// вручную оператором, что для демонстрационного режима неприемлемо.
if (!manualMode(dBase)) {
syncTimeTime = currentTime;
mtm_cmd_current_time current_time;
current_time.header.type = MTM_CMD_TYPE_CURRENT_TIME;
current_time.header.protoVersion = MTM_VERSION_0;
localTime = localtime(&currentTime);
current_time.time = localTime->tm_hour * 60 + localTime->tm_min;
for (int idx = 0; idx < 16; idx++) {
current_time.brightLevel[idx] = lightGroupBright[idx];
}
ssize_t rc = send_mtm_cmd(coordinatorFd, 0xFFFF, &current_time, kernel);
if (rc == -1) {
lostZBCoordinator(dBase, threadId, &coordinatorUuid);
return;
}
if (kernel->isDebug) {
kernel->log.ulogw(LOG_LEVEL_INFO, "[%s] Written %ld bytes.", TAG, rc);
}
}
}
// опрашиваем датчики на локальном координаторе
currentTime = time(nullptr);
if (currentTime - checkSensorTime >= 10) {
checkSensorTime = currentTime;
zigbee_mt_cmd_af_data_request req = {0};
req.dst_addr = 0x0000;
req.sep = 0xE8;
```

```
req.dep = 0xE8;
req.cid = MBEE_API_LOCAL_IOSTATUS_CLUSTER;
ssize_t rc = send_zb_cmd(coordinatorFd, AF_DATA_REQUEST, &req, kernel);
if (rc == -1) {
lostZBCoordinator(dBase, threadId, &coordinatorUuid);
return;
}
if (kernel->isDebug) {
kernel->log.ulogw(LOG_LEVEL_INFO, "[%s] rc=%ld", TAG, rc);
}
req = {0};
req.dst_addr = 0x0000;
req.sep = 0xE8;
req.dep = 0xE8;
req.cid = MBEE_API_GET_TEMP_CLUSTER;
rc = send_zb_cmd(coordinatorFd, AF_DATA_REQUEST, &req, kernel);
if (rc == -1) {
lostZBCoordinator(dBase, threadId, &coordinatorUuid);
return;
}
if (kernel->isDebug) {
kernel->log.ulogw(LOG_LEVEL_INFO, "[%s] rc=%ld", TAG, rc);
}
}
// получаем версию модуля, по полученному ответу понимаем что модуль работает
currentTime = time(nullptr);
if (currentTime - checkCoordinatorTime >= 15) {
if (!isCheckCoordinatorRespond) {
// координатор не ответил
kernel->log.ulogw(LOG_LEVEL_ERROR, "[%s] ERROR Coordinator not answer for
request module version",
TAG);
// останавливаем поток с целью его последующего автоматического запуска и
инициализации
mtmZigbeeStopThread(mtmZigbeeDBase, threadId);
```

```
AddDeviceRegister(mtmZigbeeDBase, (char *) coordinatorUuid.data(),
(char *) "Координатор не ответил на запрос");
return;
}
// сбрасываем флаг полученного ответа от координатора
isCheckCoordinatorRespond = false;
checkCoordinatorTime = currentTime;
zigbee_mt_cmd_af_data_request req = {0};
req.dst_addr = 0x0000;
req.sep = 0xE8;
req.dep = 0xE8;
req.cid = 0x0100;
ssize_t rc = send_zb_cmd(coordinatorFd, AF_DATA_REQUEST, &req, kernel);
if (rc == -1) {
lostZBCoordinator(dBase, threadId, &coordinatorUuid);
return;
}
if (kernel->isDebug) {
kernel->log.ulogw(LOG_LEVEL_INFO, "[%s] rc=%ld", TAG, rc);
}
}
// проверка на наступление астрономических событий
currentTime = time(nullptr);
if (currentTime - checkAstroTime > 60) {
// костыль для демонстрационных целей, т.е. когда флаг установлен, ни какого
автоматического
// управления светильниками не происходит. только ручной режим.
if (!manualMode(dBase)) {
double lon = 0, lat = 0;
checkAstroTime = currentTime;
MYSQL_RES *res = mtmZigbeeDBase->sqlexec("SELECT * FROM node LIMIT
1");
if (res) {
MYSQL_ROW row = mysql_fetch_row(res);
mtmZigbeeDBase->makeFieldsList(res);
```

```
if (row) {
lon = strtod(row[mtmZigbeeDBase->getFieldIndex("longitude")], nullptr);
lat = strtod(row[mtmZigbeeDBase->getFieldIndex("latitude")], nullptr);
}
mysql_free_result(res);
}
// управление контактором, рассылка пакетов светильникам
checkAstroEvents(currentTime, lon, lat, dBase, threadId);
// рассылка пакетов светильникам по параметрам заданным в программах
checkLightProgram(mtmZigbeeDBase, currentTime, lon, lat, threadId);
}
}
currentTime = time(nullptr);
if (currentTime - checkLinkState > 10) {
checkLinkState = currentTime;
mtmCheckLinkState(mtmZigbeeDBase);
}
currentTime = time(nullptr);
if (currentTime - checkOutPacket > 2) {
checkOutPacket = currentTime;
mtmZigbeeProcessOutPacket(threadId);
}
run = mtmZigbeeGetRun();
usleep(10000);
}
}
}
class MtmDevLightStatus extends MtmPktHeader
{
const MAX_SENSORS = 16;
public $mac;
public $alert;
public $data;
public function rules()
{
```

```
return [
['mac', 'string', 'length' => [16]],
['alert', 'integer', 'min' => 0x00, 'max' => 0xffff],
['data', 'checkDataSize'],
];
}
public function checkDataSize($attr, $param)
{
$statusValues = $this->attributes[$attr];
if (!is_array($statusValues)) {
$this->addError($attr, 'Должен быть список элементов ');
return;
}
$count = count($statusValues);
if ($count == 0 || $count > 16) {
$this->addError($attr, 'Список элементов должен быть больше 0 и меньше 17');
return;
}
}
public function loadBase64Data($data)
{
$this->loadBinaryData(base64_decode($data));
}
public function loadBinaryData($data)
{
$dataLen = strlen($data);
// таким нехитрым способом определяем сколько на самом деле двух байтовых значений
статусов датчиков
// пришло в пакете со светильника (12 = 1 тип + 1 версия + 8 mac + 2 alert)
$sensorsCount = $dataLen - 12;
if ($sensorsCount % 2 != 0) {
$this->addError('sensors_count', 'Не чётное значение байт статусов датчиков.');
return false;
} else {
$sensorsCount = $sensorsCount / 2;
```

```
if ($sensorsCount >= self::MAX_SENSORS) {
$this->addError('sensors_count', 'Количество статусов датчиков больше ' .
self::MAX_SENSORS);
return false;
}
}
$this->type = ord($data[0]);
$this->protoVersion = ord($data[1]);
$this->mac =
self::i2h(ord($data[9])) .
self::i2h(ord($data[8])) .
self::i2h(ord($data[7])) .
self::i2h(ord($data[6])) .
self::i2h(ord($data[5])) .
self::i2h(ord($data[4])) .
self::i2h(ord($data[3])) .
self::i2h(ord($data[2]));
$this->alert = ord($data[10]) | (ord($data[11]) << 8);
for ($i = 0; $i < $sensorsCount; $i++) {
$this->data[$i] = ord($data[$i * 2 + 12]) | (ord($data[$i * 2 + 13]) << 8);
}
return $this->validate();
}
public static function i2h($int)
{
return $int < 16 ? '0' . dechex($int) : dechex($int);
}
}
class MtmServerAmqpWorker extends Worker
{
const ROUTE_TO_LSERVER = 'routeLServer';
const EXCHANGE = 'light';
const QUERY_LSERVER = 'queryLServer';
public $active = true;
public $maxProcesses = 1;
```

```
public $delay = 60;
public $run = true;
/** @var AMQPStreamConnection */
private $connection;
/** @var AMQPChannel $channel */
private $channel;
public function handler($signo)
{
$this->log('call handler... ' . $signo);
switch ($signo) {
case SIGTERM:
case SIGINT:
$this->run = false;
break;
}
}
public function init()
{
$this->logFile = '@console/runtime/daemon/logs/mtm_server_amqp_worker.log';
parent::init();
$params = Yii::$app->params;
if (!isset($params['amqpServer']['host']) ||
!isset($params['amqpServer']['port']) ||
!isset($params['amqpServer']['user']) ||
!isset($params['amqpServer']['password'])) {
$this->log('Не задана конфигурация сервера сообщений и шкафа.');
$this->run = false;
return;
}
try {
$this->connection = new AMQPStreamConnection($params['amqpServer']['host'],
$params['amqpServer']['port'],
$params['amqpServer']['user'],
$params['amqpServer']['password']);
$this->channel = $this->connection->channel();
```

```
$this->channel->exchange_declare(self::EXCHANGE, 'direct', false, true, false);
$this->channel->queue_declare(self::QUERY_LSERVER, false, true, false, false);
$this->channel->queue_bind(self::QUERY_LSERVER, self::EXCHANGE,
self::ROUTE_TO_LSERVER);
$this->channel->basic_consume(self::QUERY_LSERVER, '', false, false, false, false,
[&$this, 'callback']);
} catch (Exception $e) {
$this->log($e->getMessage());
$this->log('init not complete');
$this->run = false;
return;
}
pcntl_signal(SIGTERM, [&$this, 'handler']);
pcntl_signal(SIGINT, [&$this, 'handler']);
$this->log('init complete');
}
public function run()
{
$checkNodes = 0;
$checkNodesRate = 30;
$this->log('run...');
while ($this->run) {
// $this->log('tick...');
// TODO: придумать механизм который позволит выбирать все сообщения в очереди, а
не по одному с задержкой в секунду
try {
if (count($this->channel->callbacks)) {
// $this->log('wait for message...');
$this->channel->wait(null, true);
// $this->log('end wait...');
}
} catch (ErrorException $e) {
$this->log($e->getMessage());
} catch (AMQPTimeoutException $e) {
$this->log($e->getMessage());
```

```
} catch (Exception $e) {
$this->log($e->getMessage());
return;
}
// изменяем статус шкафа если от координатора давно не поступали данные
// это не верно, т.к. шкаф может быть доступен, но все потоки в том числе и
координатора на нём остановлены
// пока сделаю так
$linkTimeOut = 60;
$currentTime = time();
if ($checkNodes + $checkNodesRate < $currentTime) {
$checkNodes = $currentTime;
// для всех шкафов от которых не было пакетов состояния координатора более
$timeOut секунд,
// а статус был "В порядке", устанавливаем статус "Нет связи"
$db = Yii::$app->db;
// выбираем все шкафы которые будут менять статус с WORK на NOT_LINK
$params = [
':deviceType' => DeviceType::DEVICE_ZB_COORDINATOR,
':timeOut' => $linkTimeOut,
':workUuid' => DeviceStatus::WORK,
':measureType' => MeasureType::COORD_DIGI1,
];
$command = $db->createCommand("
SELECT nt.uuid as nodeUuid, dt.uuid as deviceUuid, nt.address as nodeAddr,
dt.deviceTypeUuid,
dt.address as devAddr, nt.oid
FROM node AS nt
LEFT JOIN device AS dt ON dt.nodeUuid=nt.uuid
LEFT JOIN sensor_channel AS sct ON sct.deviceUuid=dt.uuid
LEFT JOIN measure AS mt ON mt.sensorChannelUuid=sct.uuid
WHERE dt.deviceTypeUuid=:deviceType
AND nt.deviceStatusUuid=:workUuid
AND sct.measureTypeUuid=:measureType
```

```
AND (timestampdiff(second, mt.changedAt, current_timestamp()) > :timeOut OR mt.changedAt
IS
NULL)
GROUP BY dt.uuid", $params);
//ORDER BY mt.changedAt DESC", $params);
$result = $command->query()->readAll();
// $this->log('sel query: ' . $command->rawSql);
// создаём записи в логах о смене статуса, составляем список для изменения статуса
$uuid2Update = [];
foreach ($result as $device) {
$uuid2Update[] = $device['nodeUuid'];
$address = $device['deviceTypeUuid'] ==
DeviceType::DEVICE_ZB_COORDINATOR ? $device['nodeAddr'] : $device['devAddr'];
$rc = MainFunctions::deviceRegister($device['deviceUuid'], "Устройство изменило
статус на 'Нет связи' (" . $address . ")", $device['oid']);
$this->log('MainFunctions::deviceRegister: ' . $rc);
}
// изменяем статус
$params = [
':noLinkUuid' => DeviceStatus::NOT_LINK,
];
$inParam = [];
$inParamSql = $db->getQueryBuilder()->buildCondition(['IN', 'nt.uuid', $uuid2Update],
$inParam);
$params = array_merge($params, $inParam);
$command = $db->createCommand("
UPDATE node AS nt SET nt.deviceStatusUuid=:noLinkUuid, changedAt=current_timestamp()
WHERE $inParamSql", $params);
// $this->log('upd query: ' . $command->rawSql);
$command->execute();
// для всех шкафов от которых были получены пакеты со статусом координатора
менее 30 секунд назад,
// а статус был "Нет связи", устанавливаем статус "В порядке"
$params = [
':timeOut' => $linkTimeOut,
```

```
':noLinkUuid' => DeviceStatus::NOT_LINK,
':deviceType' => DeviceType::DEVICE_ZB_COORDINATOR,
':measureType' => MeasureType::COORD_DIGI1,
];
$command = $db->createCommand("
SELECT nt.uuid as nodeUuid, dt.uuid as deviceUuid, nt.address as nodeAddr,
dt.deviceTypeUuid,
dt.address as devAddr, nt.oid
FROM node AS nt
LEFT JOIN device AS dt ON dt.nodeUuid=nt.uuid
LEFT JOIN sensor_channel AS sct ON sct.deviceUuid=dt.uuid
LEFT JOIN measure AS mt ON mt.sensorChannelUuid=sct.uuid
WHERE dt.deviceTypeUuid=:deviceType
AND nt.deviceStatusUuid=:noLinkUuid
AND sct.measureTypeUuid=:measureType
AND (timestampdiff(second, mt.changedAt, current_timestamp()) < :timeOut)
GROUP BY dt.uuid", $params);
//ORDER BY mt.changedAt DESC ", $params);
// $this->log('upd query: ' . $command->rawSql);
$result = $command->query()->readAll();
// создаём записи в логах о смене статуса, составляем список для изменения статуса
$uuid2Update = [];
foreach ($result as $device) {
$uuid2Update[] = $device['nodeUuid'];
$address = $device['deviceTypeUuid'] ==
DeviceType::DEVICE_ZB_COORDINATOR ? $device['nodeAddr'] : $device['devAddr'];
$rc = MainFunctions::deviceRegister($device['deviceUuid'], "Устройство изменило
статус на 'В порядке' (" . $address . ")", $device['oid']);
$this->log('MainFunctions::deviceRegister: ' . $rc);
}
$params = [
':workUuid' => DeviceStatus::WORK,
];
$inParam = [];
$inParamSql = $db->getQueryBuilder()->buildCondition(['IN', 'nt.uuid', $uuid2Update],
```

```
$inParam);
$params = array_merge($params, $inParam);
$command = $db->createCommand("
UPDATE node AS nt SET nt.deviceStatusUuid=:workUuid, changedAt=current_timestamp()
WHERE $inParamSql", $params);
// $this->log('upd query: ' . $command->rawSql);
$command->execute();
// для всех шкафов у которых нет координаторов и каналов измерения для них,
ставим нет связи
$params = [
':workUuid' => DeviceStatus::WORK,
':noLinkUuid' => DeviceStatus::NOT_LINK,
':deviceType' => DeviceType::DEVICE_ZB_COORDINATOR,
];
$command = $db->createCommand("UPDATE node AS nt SET
nt.deviceStatusUuid=:noLinkUuid
WHERE nt.uuid NOT IN (
SELECT dt.nodeUuid FROM device AS dt
LEFT JOIN sensor_channel AS sct ON sct.deviceUuid=dt.uuid
WHERE dt.deviceTypeUuid=:deviceType
GROUP BY dt.uuid
)
AND nt.deviceStatusUuid=:workUuid", $params);
// $this->log('upd query: ' . $command->rawSql);
$command->execute();
}
pcntl_signal_dispatch();
sleep(1);
}
if ($this->connection != null) {
$this->channel->close();
$this->connection->close();
}
$this->log('finish...');
}
```

```
public function callback($msg)
{
$content = json_decode($msg->body);
$type = $content->type;
switch ($type) {
default:
break;
}
}
}
class DeviceProgramController extends Controller
{
public function behaviors()
{
return [
'access' => [
'class' => AccessControl::class,
'rules' => [
[
'allow' => true,
'roles' => ['@'],
],
],
],
'verbs' => [
'class' => VerbFilter::class,
'actions' => [
'delete' => ['POST'],
],
],
];
}
public function actionIndex()
{
$dataProvider = new ActiveDataProvider([
```

```
'query' => DeviceProgram::find(),
]);
return $this->render('index', [
'dataProvider' => $dataProvider,
]);
}
public function actionView($id)
{
return $this->render('view', [
'model' => $this->findModel($id),
]);
}
protected function findModel($id)
{
if (($model = DeviceProgram::findOne($id)) !== null) {
return $model;
}
throw new NotFoundHttpException('The requested page does not exist.');
}
public function actionCreate()
{
$model = new DeviceProgram();
$model->oid = User::getOid(Yii::$app->user->identity);
if ($model->load(Yii::$app->request->post()) && $model->save()) {
return $this->redirect(['view', 'id' => $model->_id]);
}
return $this->render('create', [
'model' => $model,
]);
}
public function actionUpdate($id)
{
$model = $this->findModel($id);
if ($model->load(Yii::$app->request->post()) && $model->save()) {
MainFunctions::register("Изменена программа работы: '{$model->title}'");
```

```
return $this->redirect(['view', 'id' => $model->_id]);
}
return $this->render('update', [
'model' => $model,
]);
}
public function actionDelete($id)
{
$model = $this->findModel($id);
$used = DeviceConfig::find()->where(['parameter' =>
DeviceConfig::PARAM_LIGHT_PROGRAM, 'value' => $model->title])->all();
if (count($used) > 0) {
Yii::$app->session->setFlash('error', '<h3>Эту программу нельзя удалить, так как она
используется.</h3>');
return $this->render('view', [
'model' => $model,
]);
}
$model->delete();
return $this->redirect(['index']);
}
public function actionCalendar()
{
$events = [];
$defProgram = "";
$coordinates = ObjectController::getAverageCoordinates();
if (isset($_GET["group"]))
$group = $_GET["group"];
else $group = 0;
$range = 365;
$shift = 30;
$today = time() - 3600 * 24 * $shift;
$today = strtotime(date('Y-m-d', $today));
$groupControls = GroupControl::find()
->where(['groupUuid' => $group])
```

```
->where(['between', 'date', date('Y-m-d', $today),
date('Y-m-d', $today + 86400 * ($range + $shift))])
->all();
$group = Group::find()->where(['uuid' => $group])->limit(1)->one();
if ($group && $group['deviceProgramUuid']) {
$defProgram = $group['deviceProgram']['title'];
}
$groupControlArray = [];
foreach ($groupControls as $groupControl) {
$grpCtlTimestamp = strtotime($groupControl->date);
$groupControlArray[date("Y-m-d", $grpCtlTimestamp)][$groupControl->type] =
$groupControl;
}
unset($groupControls);
for ($count = 0; $count < $range; $count++) {
// $sunrise_time = date_sunrise($today, SUNFUNCS_RET_TIMESTAMP,
$coordinates['latitude'], $coordinates['longitude']);
// $sunset_time = date_sunset($today, SUNFUNCS_RET_TIMESTAMP,
$coordinates['latitude'], $coordinates['longitude']);
$on = 0;
$off = 0;
$currentDate = date("Y-m-d", $today);
if (isset($groupControlArray[$currentDate])) {
if (isset($groupControlArray[$currentDate][1])) {
$elem = $groupControlArray[$currentDate][1];
$on = 1;
$event = new Event();
$event->id = $count * 2 + 1;
$event->title = "включение [" . $defProgram . "]";
if ($elem['deviceProgramUuid'])
$event->title = "Программа [" . $elem['deviceProgram']['title'] . "]";
$event->backgroundColor = 'green';
$event->start = $elem['date'];
$event->color = '#ffffff';
$events[] = $event;
```

```
}
}
if ($on == 0) {
$event = new Event();
$event->id = $count * 2 + 1;
$event->title = "Программа [" . $defProgram . "]";
$event->backgroundColor = 'green';
$event->start = date("Y-m-d H:i:s", $today);
$event->color = '#ffffff';
$events[] = $event;
}
$today += 24 * 3600;
}
return $this->render('calendar', [
'events' => $events,
'groupTitle' => $group['title'],
]);
}
public function actionCalendarNode($node)
{
$events = [];
if (($nodeObj = Node::find()->where(['uuid' => $node])->one()) === null) {
throw new NotFoundHttpException('The requested page does not exist.');
}
$range = 365;
$shift = 30;
$today = time() - 3600 * 24 * $shift;
$nodeControls = NodeControl::find()
->where(['nodeUuid' => $node])
->where(['between', 'date', date('Y-m-d', $today),
date('Y-m-d', $today + 86400 * ($range + $shift))])
->all();
$nodeControlArray = [];
foreach ($nodeControls as $nodeControl) {
$nodeCtlTimestamp = strtotime($nodeControl->date);
```

```
$nodeControlArray[date("Y-m-d", $nodeCtlTimestamp)][$nodeControl->type] =
$nodeControl;
}
unset($nodeControls);
for ($count = 0; $count < $range; $count++) {
$sunrise_time = date_sunrise($today, SUNFUNCS_RET_TIMESTAMP, $nodeObj->object-
>latitude, $nodeObj->object->longitude);
$sunset_time = date_sunset($today, SUNFUNCS_RET_TIMESTAMP, $nodeObj->object-
>latitude, $nodeObj->object->longitude);
$on = 0;
$off = 0;
$currentDate = date("Y-m-d", $today);
if (isset($nodeControlArray[$currentDate])) {
if (isset($nodeControlArray[$currentDate][0])) {
$elem = $nodeControlArray[$currentDate][0];
$off = 1;
$event = new Event();
$event->id = $count * 2;
$event->title = "выключение";
$event->backgroundColor = 'orange';
$event->start = $elem['date'];
$event->color = '#ffffff';
$events[] = $event;
}
if (isset($nodeControlArray[$currentDate][1])) {
$elem = $nodeControlArray[$currentDate][1];
$on = 1;
$event = new Event();
$event->id = $count * 2 + 1;
$event->title = "включение";
$event->backgroundColor = 'green';
$event->start = $elem['date'];
$event->color = '#ffffff';
$events[] = $event;
}
```

```
}
if ($off == 0) {
$event = new Event();
$event->id = $count * 2;
$event->title = "выключение";
$event->backgroundColor = 'orange';
$event->start = date("Y-m-d H:i:s", $sunrise_time);
$event->color = '#ffffff';
$events[] = $event;
}
if ($on == 0) {
$event = new Event();
$event->id = $count * 2 + 1;
$event->title = "включение";
$event->backgroundColor = 'green';
$event->start = date("Y-m-d H:i:s", $sunset_time);
$event->color = '#ffffff';
$events[] = $event;
}
//echo date("Y-m-d H:i",$event->start).PHP_EOL;
$today += 24 * 3600;
}
return $this->render('calendar-node', [
'events' => $events,
'nodeTitle' => $nodeObj->address,
]);
}
public function actionCalendarAll()
{
if (!Yii::$app->user->can(User::PERMISSION_ADMIN)) {
return $this->redirect('/site/index');
}
$events = [];
$range = 365;
$shift = 30;
```

```
$today = time() - 3600 * 24 * $shift;
$averageCoord = ObjectController::getAverageCoordinates();
for ($count = 0; $count < $range; $count++) {
$sunrise_time = date_sunrise($today, SUNFUNCS_RET_TIMESTAMP,
$averageCoord['latitude'], $averageCoord['longitude']);
$sunset_time = date_sunset($today, SUNFUNCS_RET_TIMESTAMP,
$averageCoord['latitude'], $averageCoord['longitude']);
$event = new Event();
$event->id = $count * 2;
$event->title = "выключение";
$event->backgroundColor = 'orange';
$event->start = date("Y-m-d H:i:s", $sunrise_time);
$event->color = '#ffffff';
$events[] = $event;
$event = new Event();
$event->id = $count * 2 + 1;
$event->title = "включение";
$event->backgroundColor = 'green';
$event->start = date("Y-m-d H:i:s", $sunset_time);
$event->color = '#ffffff';
$events[] = $event;
//echo date("Y-m-d H:i",$event->start).PHP_EOL;
$today += 24 * 3600;
}
return $this->render('calendar-all', [
'events' => $events,
]);
}
}
class NodeController extends Controller
{
public function behaviors()
{
return [
'access' => [
```

```
'class' => AccessControl::class,
'rules' => [
[
'allow' => true,
'roles' => ['@'],
],
],
],
'verbs' => [
'class' => VerbFilter::class,
'actions' => [
'delete' => ['POST'],
],
],
];
}
public function actionIndex()
{
if (isset($_POST['editableAttribute'])) {
if (!Yii::$app->user->can(User::PERMISSION_ADMIN)) {
return json_encode('Нет прав.');
}
$model = Node::find()
->where(['_id' => $_POST['editableKey']])
->limit(1)
->one();
if ($_POST['editableAttribute'] == 'address') {
$model['address'] = $_POST['Node'][$_POST['editableIndex']]['address'];
}
if ($_POST['editableAttribute'] == 'objectUuid') {
$model['objectUuid'] = $_POST['Node'][$_POST['editableIndex']]['objectUuid'];
}
if ($_POST['editableAttribute'] == 'nodeUuid') {
$model['nodeUuid'] = $_POST['Node'][$_POST['editableIndex']]['nodeUuid'];
}
```

```
if ($_POST['editableAttribute'] == 'software') {
$model['software'] = $_POST['Node'][$_POST['editableIndex']]['software'];
}
if ($_POST['editableAttribute'] == 'deviceStatusUuid') {
$model['deviceStatusUuid'] = $_POST['Node'][$_POST['editableIndex']]
['deviceStatusUuid'];
}
if ($_POST['editableAttribute'] == 'phone') {
$model['phone'] = $_POST['Node'][$_POST['editableIndex']]['phone'];
}
$model->save();
return json_encode('');
}
$searchModel = new NodeSearch();
$dataProvider = $searchModel->search(Yii::$app->request->queryParams);
$dataProvider->pagination->pageSize = 15;
return $this->render(
'index',
[
'searchModel' => $searchModel,
'dataProvider' => $dataProvider,
]
);
}
public function actionStatus()
{
$searchModel = new NodeSearch();
$dataProvider = $searchModel->search(Yii::$app->request->queryParams);
$dataProvider->pagination->pageSize = 50;
return $this->render(
'status',
[
'searchModel' => $searchModel,
'dataProvider' => $dataProvider,
]
```

```
);
}
public function actionView($id)
{
return $this->render(
'view',
[
'model' => $this->findModel($id),
]
);
}
public function actionCreate()
{
if (!Yii::$app->user->can(User::PERMISSION_ADMIN)) {
return $this->redirect('/site/index');
}
$model = new Node();
if ($model->load(Yii::$app->request->post())) {
// проверяем все поля, если что-то не так показываем форму с ошибками
if (!$model->validate()) {
echo json_encode($model->errors);
return $this->render('create', ['model' => $model]);
}
// сохраняем запись
if ($model->save(false)) {
return $this->redirect(['view', 'id' => $model->_id]);
}
echo json_encode($model->errors);
}
return $this->render('create', ['model' => $model]);
}
public function actionNew()
{
if (!Yii::$app->user->can(User::PERMISSION_ADMIN)) {
return $this->redirect('index');
```

```
}
$equipments = array();
/* $equipment_count = 0;
$objects = Objects::find()
->select('*')
->all();*/
return $this->render('new', ['equipments' => $equipments]);
}
public function actionDashboard($uuid, $type)
{
if (isset($_POST['on'])) {
$device = Device::find()->where(['uuid' => $_POST['device']])->limit(1)->one();
if ($device)
DeviceController::contactor($_POST['on'], $device);
}
$node = Node::find()
->where(['uuid' => $uuid])
->limit(1)
->one();
$camera = null;
$energy = null;
$coordinator = null;
$parameters = [];
if ($node) {
if (isset($_POST['reset'])) {
DeviceController::resetCoordinator($node);
}
$camera = Camera::find()->where(['nodeUuid' => $node['uuid']])->limit(1)->one();
if ($camera) {
$camera->startTranslation();
}
$parameters['control']['signal'] = $node['security'];
$energy = Device::find()
->where(['nodeUuid' => $node['uuid']])
->andWhere(['deviceTypeUuid' => DeviceType::DEVICE_ELECTRO])
```

```
->limit(1)
->one();
$coordinator = Device::find()
->where(['nodeUuid' => $node['uuid']])
->andWhere(['deviceTypeUuid' => DeviceType::DEVICE_ZB_COORDINATOR])
->limit(1)
->one();
}
if ($energy) {
$measures = (Measure::find()
->where(['type' => MeasureType::MEASURE_TYPE_CURRENT])
->orderBy('date DESC'))
->limit(100)
->all();
foreach ($measures as $measure) {
if ($measure['sensorChannel']['measureTypeUuid'] == MeasureType::VOLTAGE &&
$measure['sensorChannel']['deviceUuid'] == $energy['uuid']) {
if ($measure['parameter'] == 1)
$parameters['control']['u'] = $measure['value'];
}
if ($measure['sensorChannel']['measureTypeUuid'] == MeasureType::CURRENT &&
$measure['sensorChannel']['deviceUuid'] == $energy['uuid']) {
if ($measure['parameter'] == 1)
$parameters['control']['i'] = $measure['value'];
}
if ($measure['sensorChannel']['measureTypeUuid'] == MeasureType::POWER &&
$measure['sensorChannel']['deviceUuid'] == $energy['uuid']) {
if ($measure['parameter'] == 0)
$parameters['control']['w'] = $measure['value'];
}
}
}
if ($coordinator) {
$measure = (Measure::find()
->where(['sensor_channel.measureTypeUuid' => MeasureType::COORD_IN1])
```

```
->joinWith('sensorChannel')
->orderBy('date DESC'))
->limit(1)
->one();
if ($measure && $measure['sensorChannel'] &&
$measure['sensorChannel']['measureTypeUuid'] == MeasureType::COORD_IN1 &&
$measure['sensorChannel']['deviceUuid'] == $coordinator['uuid']) {
$parameters['control']['door'] = $measure['value'];
}
$measure = (Measure::find()
->where(['sensor_channel.measureTypeUuid' => MeasureType::COORD_IN2])
->joinWith('sensorChannel')
->orderBy('date DESC'))
->limit(1)
->one();
if ($measure && $measure['sensorChannel'] &&
$measure['sensorChannel']['measureTypeUuid'] == MeasureType::COORD_IN2 &&
$measure['sensorChannel']['deviceUuid'] == $coordinator['uuid']) {
$parameters['control']['contactor'] = $measure['value'];
}
$measure = (Measure::find()
->where(['sensor_channel.measureTypeUuid' => MeasureType::COORD_DIGI1])
->joinWith('sensorChannel')
->orderBy('date DESC'))
->limit(1)
->one();
if ($measure && $measure['sensorChannel'] &&
$measure['sensorChannel']['measureTypeUuid'] == MeasureType::COORD_DIGI1 &&
$measure['sensorChannel']['deviceUuid'] == $coordinator['uuid']) {
$parameters['control']['relay'] = $measure['value'];
}
}
$parameters['u1'] = Measure::getLastMeasureNodeByType(MeasureType::VOLTAGE,
$node['uuid'],
MeasureType::MEASURE_TYPE_CURRENT, 1);
```

```
$parameters['u2'] = Measure::getLastMeasureNodeByType(MeasureType::VOLTAGE,
$node['uuid'],
MeasureType::MEASURE_TYPE_CURRENT, 2);
$parameters['u3'] = Measure::getLastMeasureNodeByType(MeasureType::VOLTAGE,
$node['uuid'],
MeasureType::MEASURE_TYPE_CURRENT, 3);
if (!$parameters['u1']) $parameters['u1'] = '-';
else $parameters['u1'] = $parameters['u1']['value'];
if (!$parameters['u2']) $parameters['u2'] = '-';
else $parameters['u2'] = $parameters['u2']['value'];
if (!$parameters['u3']) $parameters['u3'] = '-';
else $parameters['u3'] = $parameters['u3']['value'];
$parameters['voltage'] = "<span style='color: darkgreen'>" . $parameters['u1'] . "," .
$parameters['u2'] . "," . $parameters['u3'] . "</span>";
$parameters['i1'] = Measure::getLastMeasureNodeByType(MeasureType::CURRENT,
$node['uuid'],
MeasureType::MEASURE_TYPE_CURRENT, 1);
$parameters['i2'] = Measure::getLastMeasureNodeByType(MeasureType::CURRENT,
$node['uuid'],
MeasureType::MEASURE_TYPE_CURRENT, 2);
$parameters['i3'] = Measure::getLastMeasureNodeByType(MeasureType::CURRENT,
$node['uuid'],
MeasureType::MEASURE_TYPE_CURRENT, 3);
if (!$parameters['i1']) $parameters['i1'] = '-';
else $parameters['i1'] = $parameters['i1']['value'];
if (!$parameters['i2']) $parameters['i2'] = '-';
else $parameters['i2'] = $parameters['i2']['value'];
if (!$parameters['i3']) $parameters['i3'] = '-';
else $parameters['i3'] = $parameters['i3']['value'];
$parameters['current'] = "<span style='color: darkgreen'>" . $parameters['i1'] . "," .
$parameters['i2'] . "," . $parameters['i3'] . "</span>";
$parameters['w'] = Measure::getLastMeasureNodeByType(MeasureType::POWER,
$node['uuid'],
MeasureType::MEASURE_TYPE_CURRENT, 0);
if (!$parameters['w']) $parameters['w'] = '-';
```

```
else $parameters['w'] = $parameters['w']['value'];
$parameters['power'] = "<span style='color: darkgreen'>" . $parameters['w'] . "</span>";
$w = Measure::getLastMeasureNodeByType(MeasureType::POWER, $node['uuid'],
MeasureType::MEASURE_TYPE_TOTAL_CURRENT, 0);
if (!$w) $w = '-';
else $w = $w['value'];
$parameters['total'] = "<span style='color: darkgreen'>" . $w . "</span>";
return $this->render(
'dashboard',
[
'node' => $node,
'coordinator' => $coordinator,
'camera' => $camera,
'type' => $type,
'parameters' => $parameters,
'counterDate' => date('Y-m-d'),
'counterValue' => $node->getCounterValue() . ' кВт',
]
);
}
public
function actionUpdate($id)
{
if (!Yii::$app->user->can(User::PERMISSION_ADMIN)) {
return $this->redirect('/site/index');
}
$model = $this->findModel($id);
$zbcDevice = Device::find()->where([
'nodeUuid' => $model->uuid,
'deviceTypeUuid' => DeviceType::DEVICE_ZB_COORDINATOR,
])->limit(1)->one();
$zbcMode = null;
if ($zbcDevice != null) {
$config = DeviceConfig::find()->where([
'deviceUuid' => $zbcDevice->uuid,
```

```
'parameter' => DeviceConfig::PARAM_ZB_COORDINATOR_MODE,
])->limit(1)->one();
if ($config != null) {
$zbcMode = $config->value;
} else {
$zbcMode = 0;
}
}
if ($model->load(Yii::$app->request->post())) {
if ($model->save()) {
return $this->redirect(['view', 'id' => $model->_id]);
} else {
return $this->render(
'update',
[
'model' => $model,
'zbcMode' => $zbcMode,
]
);
}
} else {
return $this->render(
'update',
[
'model' => $model,
'zbcMode' => $zbcMode,
]
);
}
}
public
function actionTree()
{
$c = 'children';
$fullTree = array();
```

```
$types = DeviceType::find()
->select('*')
->orderBy('title')
->all();
$oCnt0 = 0;
foreach ($types as $type) {
$fullTree[$oCnt0]['title'] = Html::a(
$type['title'],
['equipment-type/view', 'id' => $type['_id']]
);
$equipments = Node::find()
->select('*')
->where(['equipmentTypeUuid' => $type['uuid']])
->andWhere(['deleted' => 0])
->orderBy('serial')
->all();
$oCnt1 = 0;
foreach ($equipments as $equipment) {
$fullTree[$oCnt0][$c][$oCnt1]['title']
= Html::a(
'ул.' . $equipment['house']['street']['title'] . ', д.' . $equipment['house']['number'] . ', кв.' .
$equipment['flat']['number'],
['equipment/view', 'id' => $equipment['_id']]
);
if ($equipment['equipmentStatusUuid'] == DeviceStatus::NOT_MOUNTED) {
$class = 'critical1';
} elseif ($equipment['equipmentStatusUuid'] == DeviceStatus::NOT_WORK) {
$class = 'critical2';
} else {
$class = 'critical3';
}
$fullTree[$oCnt0][$c][$oCnt1]['status'] = '<div class="progress"><div class="'
. $class . '">' . $equipment['equipmentStatus']->title . '</div></div>';
$fullTree[$oCnt0][$c][$oCnt1]['date'] = $equipment['testDate'];
$fullTree[$oCnt0][$c][$oCnt1]['serial'] = $equipment['serial'];
```

```
$measure = Measure::find()
->select('*')
->where(['equipmentUuid' => $equipment['uuid']])
->orderBy('date DESC')
->limit(1)
->one();
if ($measure) {
$fullTree[$oCnt0][$c][$oCnt1]['measure_date'] = $measure['date'];
$fullTree[$oCnt0][$c][$oCnt1]['measure_value'] = $measure['value'];
$fullTree[$oCnt0][$c][$oCnt1]['measure_user'] = $measure['user']->name;
} else {
$fullTree[$oCnt0][$c][$oCnt1]['measure_date'] = $equipment['changedAt'];
$fullTree[$oCnt0][$c][$oCnt1]['measure_value'] = "не снимались";
$fullTree[$oCnt0][$c][$oCnt1]['measure_user'] = "-";
}
$photo = Photo::find()
->select('*')
->where(['objectUuid' => $equipment['uuid']])
->orderBy('createdAt DESC')
->limit(1)
->one();
if ($photo) {
$fullTree[$oCnt0][$c][$oCnt1]['photo_date'] = $photo['createdAt'];
$fullTree[$oCnt0][$c][$oCnt1]['photo'] = Html::a(
'<img width="100px" src="/storage/equipment/' . $photo['uuid'] . '.jpg" />',
['storage/equipment/' . $photo['uuid'] . '.jpg']
);
$fullTree[$oCnt0][$c][$oCnt1]['photo_user'] = $photo['user']->name;
} else {
$fullTree[$oCnt0][$c][$oCnt1]['photo_date'] = 'нет фото';
$fullTree[$oCnt0][$c][$oCnt1]['photo'] = '-';
$fullTree[$oCnt0][$c][$oCnt1]['photo_user'] = '-';
}
$oCnt1++;
}
```

```
$oCnt0++;
}
return $this->render(
'tree',
['equipment' => $fullTree]
);
}
public
function actionTreeMeasure()
{
ini_set('memory_limit', '-1');
$fullTree = array();
$streets = Street::find()
->select('*')
->orderBy('title')
->all();
$oCnt0 = 0;
foreach ($streets as $street) {
$house_count = 0;
$house_visited = 0;
$houses = House::find()->select('uuid,number')->where(['streetUuid' => $street['uuid']])->
orderBy('number')->all();
foreach ($houses as $house) {
$objects = Objects::find()->select('uuid,number')->where(['objectUuid' =>
$house['uuid']])->all();
foreach ($objects as $object) {
$house_count++;
$visited = 0;
$equipments = Node::find()->where(['objectUuid' => $object['uuid']])->all();
foreach ($equipments as $equipment) {
$fullTree[$oCnt0]['title']
= Html::a(
'ул.' . $equipment['house']['street']['title'] . ', д.' . $equipment['house']['number'] . ',
кв.' . $equipment['object']['number'],
['equipment/view', 'id' => $equipment['_id']]
```

```
);
$measures = Measure::find()
->select('*')
->where(['equipmentUuid' => $equipment['uuid']])
->orderBy('date DESC')
->limit(1000)
->all();
$measure_count_column = 0;
$fullTree[$oCnt0]['measure_date0'] = '';
$fullTree[$oCnt0]['measure_value0'] = '';
$fullTree[$oCnt0]['measure_date1'] = '';
$fullTree[$oCnt0]['measure_value1'] = '';
$fullTree[$oCnt0]['measure_date2'] = '';
$fullTree[$oCnt0]['measure_value2'] = '';
$fullTree[$oCnt0]['measure_date3'] = '';
$fullTree[$oCnt0]['measure_value3'] = '';
$fullTree[$oCnt0]['measure_user'] = '';
$measure_first = 0;
$measure_last = 0;
$measure_date_first = 0;
$measure_date_last = 0;
foreach ($measures as $measure) {
$fullTree[$oCnt0]['measure_date' . $measure_count_column] = $measure['date'];
$fullTree[$oCnt0]['measure_value' . $measure_count_column] =
$measure['value'];
$fullTree[$oCnt0]['measure_user'] = $measure['user']->name;
if ($measure_count_column == 0) {
$measure_first = $measure['value'];
$measure_date_first = $measure['date'];
} else {
$measure_last = $measure['value'];
$measure_date_last = $measure['date'];
}
$measure_count_column++;
if ($measure_count_column > 3) break;
```

```
}
$datetime1 = date_create($measure_date_first);
$datetime2 = date_create($measure_date_last);
if ($datetime2 && $datetime1) {
$diff = $datetime2->diff($datetime1);
$interval = $diff->format("%h") + ($diff->days * 24);
$value = number_format($measure_last - $measure_first, 2);
} else {
$interval = 0;
$value = 0;
}
$fullTree[$oCnt0]['interval'] = $interval;
$fullTree[$oCnt0]['value'] = $value;
if ($interval > 0)
$fullTree[$oCnt0]['relative'] = number_format($value / $interval, 2);
$message = Message::find()
->select('*')
->orderBy('date DESC')
->where(['flatUuid' => $equipment['flat']['uuid']])
->limit(1)
->one();
if ($message != null) {
$fullTree[$oCnt0]['message'] =
mb_convert_encoding(substr($message['message'], 0, 150), 'UTF-8', 'UTF-8');
if ($visited == 0)
$visited = 1;
$house_visited++;
}
$oCnt0++;
}
}
}
}
return $this->render(
'tree-measure',
```

```
['equipment' => $fullTree]
);
}
public
function actionDelete($id)
{
if (!Yii::$app->user->can(User::PERMISSION_ADMIN)) {
return $this->redirect('/site/index');
}
$node = Node::find()->where(['_id' => $id])->one();
if ($node) {
$node['deleted'] = true;
$node->save();
}
return $this->redirect(['index']);
}
protected
function findModel($id)
{
if (($model = Node::findOne($id)) !== null) {
return $model;
} else {
throw new NotFoundHttpException('The requested page does not exist.');
}
}
public
function actionTrends($uuid)
{
$node = Node::find()
->where(['uuid' => $uuid])
->one();
$sensorChannelPowerUuid = 0;
$sensorChannelVoltageUuid = 0;
$sensorChannelCurrentUuid = 0;
$sensorChannelFrequencyUuid = 0;
```

```
$deviceElectro = Device::find()
->where(['nodeUuid' => $node['uuid']])
->andWhere(['deleted' => 0])
->andWhere(['deviceTypeUuid' => DeviceType::DEVICE_ELECTRO])
->one();
if ($deviceElectro) {
$sensorChannel1 = SensorChannel::find()->where(['deviceUuid' => $deviceElectro['uuid']])
->andWhere(['measureTypeUuid' => MeasureType::POWER])->one();
if ($sensorChannel1)
$sensorChannelPowerUuid = $sensorChannel1['uuid'];
$sensorChannel2 = SensorChannel::find()->where(['deviceUuid' => $deviceElectro['uuid']])
->andWhere(['measureTypeUuid' => MeasureType::VOLTAGE])->one();
if ($sensorChannel2)
$sensorChannelVoltageUuid = $sensorChannel2['uuid'];
$sensorChannel3 = SensorChannel::find()->where(['deviceUuid' => $deviceElectro['uuid']])
->andWhere(['measureTypeUuid' => MeasureType::CURRENT])->one();
if ($sensorChannel3)
$sensorChannelCurrentUuid = $sensorChannel3['uuid'];
$sensorChannel4 = SensorChannel::find()->where(['deviceUuid' => $deviceElectro['uuid']])
->andWhere(['measureTypeUuid' => MeasureType::FREQUENCY])->one();
if ($sensorChannel4)
$sensorChannelFrequencyUuid = $sensorChannel4['uuid'];
}
return $this->render(
'trends',
[
'sensorChannelPowerUuid' => $sensorChannelPowerUuid,
'sensorChannelVoltageUuid' => $sensorChannelVoltageUuid,
'sensorChannelCurrentUuid' => $sensorChannelCurrentUuid,
'sensorChannelFrequencyUuid' => $sensorChannelFrequencyUuid
]
);
}
public
function actionRegister($uuid)
```

```
{
$deviceRegisters = DeviceRegister::find()
->where(['deviceUuid' => (Node::find()->where(['uuid' => $uuid])->one())]);
$provider = new ActiveDataProvider(
[
'query' => $deviceRegisters,
'sort' => false,
]
);
return $this->render(
'register',
[
'provider' => $provider
]
);
}
public function actionSetManualMode($id)
{
if (!Yii::$app->user->can(User::PERMISSION_ADMIN)) {
return $this->redirect('/site/index');
}
$model = $this->findModel($id);
$zbcDevice = Device::find()->where([
'nodeUuid' => $model->uuid,
'deviceTypeUuid' => DeviceType::DEVICE_ZB_COORDINATOR,
])->limit(1)->one();
if (Yii::$app->request->isPost && $zbcDevice != null) {
$mode = Yii::$app->request->getBodyParam('zbcmode');
$config = DeviceConfig::find()->where([
'deviceUuid' => $zbcDevice->uuid,
'parameter' => DeviceConfig::PARAM_ZB_COORDINATOR_MODE,
])->limit(1)->one();
if ($config == null) {
$config = new DeviceConfig();
$config->uuid = MainFunctions::GUID();
```

```
$config->oid = User::getOid(Yii::$app->user->identity);
$config->deviceUuid = $zbcDevice->uuid;
$config->parameter = DeviceConfig::PARAM_ZB_COORDINATOR_MODE;
}
$config->value = $mode == 1 ? 1 : 0;
$config->save();
}
return $this->redirect(['view', 'id' => $model->_id]);
}
public function actionCounterValue()
{
$request = Yii::$app->request;
$nodeUuid = $request->getQueryParam('n', null);
$date = $request->getQueryParam('d', date('Y-m-d'));
$node = Node::find()->where(['uuid' => $nodeUuid])->limit(1)->one();
if ($node == null) {
return json_encode($this->render('widget-counter-value', [
'counterValue' => '-',
'counterDate' => $date,
'node' => $node,
]));
}
$counterValue = $node->getCounterValue($date);
return json_encode($this->render('widget-counter-value', [
'counterValue' => $counterValue . ' кВт',
'counterDate' => $date,
'node' => $node,
]));
}
}
```