Moduł GATE HTTP

> **Uwaga!**
> Opisana funkcjonalność oraz integracja jest dostępna dla **GRENTON GATE HTTP, DIN, Eth (INT-211-E-01)** posiadający **firmware 1.4.2-2346 lub wyższy!**

1. Informacje ogólne

Moduł GATE HTTP to urządzenie umożliwiające systemową integrację z zewnętrznymi serwisami posługującymi się protokołem HTTP, a także szeroką grupą urządzeń i systemów zewnętrznych/trzecich firm - np. urządzeń AV z interfejsami HTTP.

> **Uwaga!**
> Dla tworzonych obiektów wirtualnych nie ma ograniczenia względem ilości obiektów - ograniczeniem jest pamięć urządzenia, na którą wpływa m.in. poziom rozbudowania logiki na module.

2. Konfiguracja modułu

> **Uwaga!**
> Przed rozpoczęciem jakiejkolwiek pracy z modułem GATE HTTP wymagana jest aktualizacja bazy interfejsów!

Ustawianie czasu za pomocą serwera NTP

Moduł GATE HTTP umożliwia ustawianie czasu za pomocą serwera NTP wraz z uwzględnieniem strefy czasowej a także zmianą czasu (zimowy/letni). Czas pobierany jest automatycznie z serwera NTP (*pool.ntp.org*).

Do konfiguracji służą trzy cechy:

- `UseNTP` – określa czy GATE używa NTP,
- `NTPTimeout` - czas oczekiwania na odpowiedź z serwera NTP,
- `TimeZone` – ustawianie strefy czasowej GATE – dostępne są 22 strefy.

> **Uwaga!**
> Pobieranie czasu z serwera NTP wymaga, aby GATE znajdowało się w sieci, która posiada połączenie z internetem.

> **Uwaga!**
> W momencie ustawienia cechy `UseCloud` = `true`, cecha `UseNTP` jest automatycznie ustawiana na wartość `true`.

2.1. Obiekty wirtualne

2.1.1. HTTPRequest

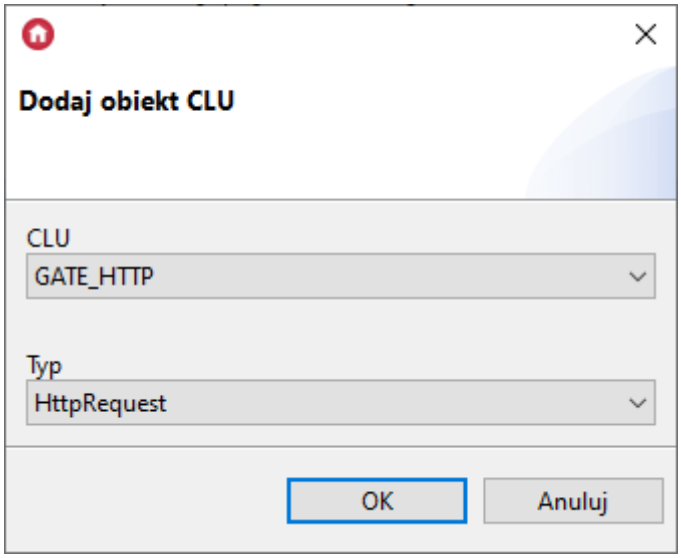
Dla HttpRequest przykładowo wykorzystywany jest serwis pogodowy http://api.openweathermap.org

Według przykładu na stronie openweathermap.org, zapytanie API wygląda następująco:

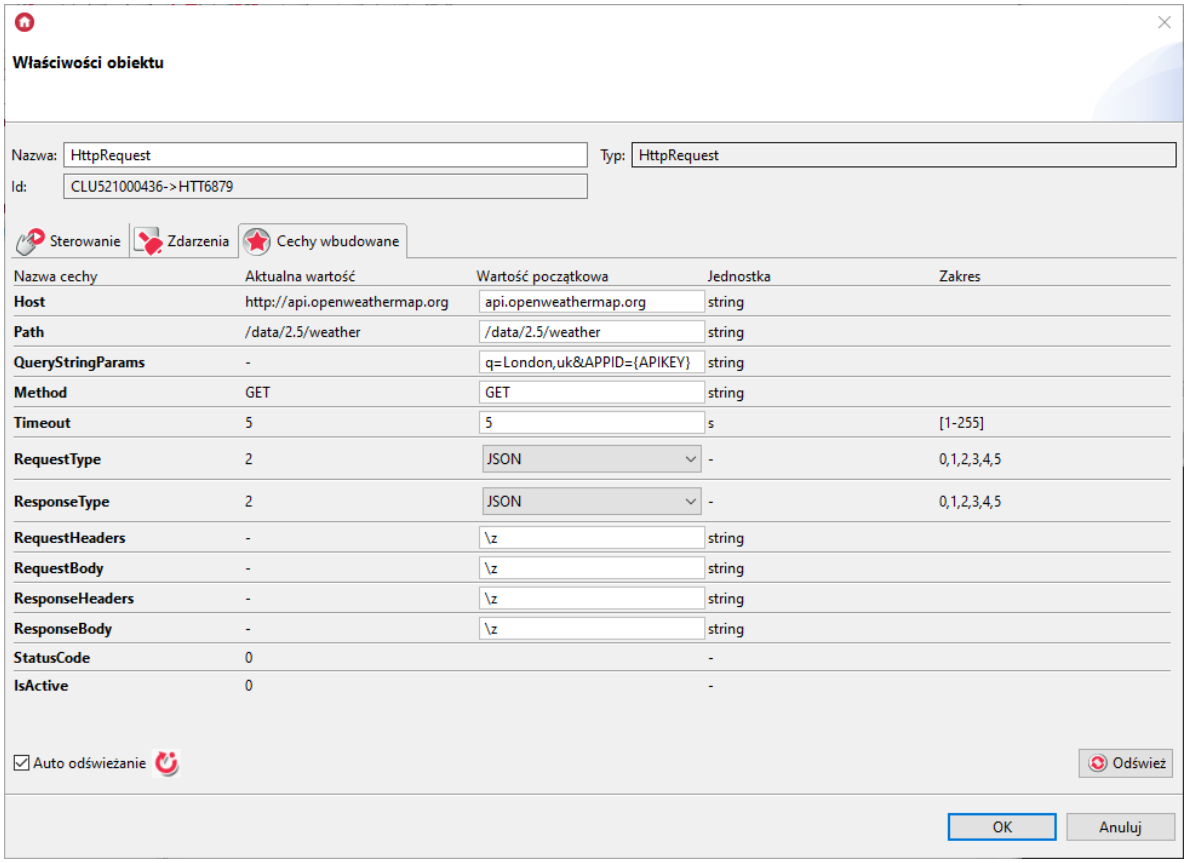
API call: http://api.openweathermap.org/data/2.5/weather?q=London&APPID={APIKEY}

HttpRequest - służy do wysyłania zapytań HTTP (typu GET, POST) do określonego hosta. Obsługiwane są standardowe Typy zawartości (content-type) m.in. JSON, XML.

Aby zastosować moduł Gate do odbierania zapytań, należy utworzyć obiekt wirtualny HttpRequest:



- W obiekcie HttpRequest należy ustawić następujące parametry:



- **Host:** api.openweathermap.org
- **Path:** /data/2.5/weather
- **QueryStringParams:** q=London&APPID={APIKEY}
- **Method:** GET
- **RequestType:** JSON
- **ResponseType:** JSON

Uwaga!
Obiekt Gate Http umożliwia obsługę połączeń szyfrowanych TLS. Jeżeli wymagane jest takie połączenie, należy na początku wartości w polu Host podać 'https://' . Jeżeli wartość nie zostanie podana, zostanie wykorzystane standardowe połączenie http.

Uwaga!
Gate Http nie obsługuje wszystkich połączeń szyfrowanych TLS, dlatego zalecamy przetestowanie połączenia z danym hostem.

Uwaga!
Podczas połączenia https czas nawiązania połączenia oraz otrzymania odpowiedzi od hosta jest dłuższy niż w przypadku połączenia http, dlatego należy zwiększyć wartość dla parametru Timeout.

Uwaga!
Cechy opisane jako nieustawialne są cechami zawierającymi odpowiedzi. Wartości początkowe tych cech należy pozostawić niezmienione. Wszelkie operacje na tych zmiennych należy wykonywać na skryptach (oraz zmiennych lokalnych).

Po wysłaniu konfiguracji i wywołaniu Metody SendRequest, StatusCode przyjmuje wartość 200 (OK).

Otrzymana odpowiedź na zapytanie jest przetrzymywana w cesze ResponseBody. Dla ustawionego ResponseType JSON, odpowiedź jest parsowana z json do tabeli. Wartość cechy jest niewidoczna z poziomu OM. Wartości odpowiedzi należy wyciągnąć z odpowiedzi z poziomu skryptu.

2.1.2. Pobieranie określonych wartości z otrzymanej odpowiedzi (XML,JSON)

Uwaga!
Uzyskaną odpowiedź ResponseBody należy przypisać do zmiennej lokalnej (w skrypcie).

Przykładowo:

local resp = GATE->httpr_openweather_json->ResponseBody

Następnie w skryptach należy wykonywać operację na zmiennej resp!

Uwaga!
Skrypty odczytujące zawartość przetrzymywaną w cesze ResponseBody muszą być wykonane na module GATE HTTP.

Otrzymane odpowiedzi w zależności od ich typu (ResponseType) są odpowiednio parsowane do postaci tabeli.

Przykładowe odczyty wartości są zapisywane do zmiennych lokalnych (wewnątrz skryptu).

Aby była możliwość wykorzystania zmiennej poza skryptem (np. do wyświetlania w aplikacji), należy ją przypisać do zmiennych globalnych (cech użytkownika).

Poniżej przykładowe odpowiedzi w formacie XML oraz JSON oraz sposób odczytania danej wartości (w przedstawionych przykładach wykorzystano odpowiedzi z serwisu pogodowego openweathermap.org)

A. JSON:

Przykładowa odpowiedź (openweathermap.org):

```
resp = [[
{"coord":
{"lon":145.77,"lat":-16.92},
"weather":[{"id":803,"main":"Clouds","description":"broken
clouds","icon":"04n"}],
"base":"cmc stations",
"main":
{"temp":293.25,"pressure":1019,"humidity":83,"temp_min":289.82,"temp_max":295.3
7},
"wind":{"speed":5.1,"deg":150},
"clouds":{"all":75},
"rain":{"3h":3},
"dt":1435658272,
"sys":
{"type":1,"id":8166,"message":0.0166,"country":"AU","sunrise":1435610796,"sunse
t":1435650870},
"id":2172797,
"name":"Cairns",
"cod":200}
]]
```

Jak odczytać:

- Wartość parametru **lon**

```
{"coord":
{"lon":145.77,"lat":-16.92},
"weather":[{"id":803,"main":"Clouds","description":"broken
clouds","icon":"04n"}],
"base":"cmc stations",
"main":
{"temp":293.25,"pressure":1019,"humidity":83,"temp_min":289.82,"temp_max":295.3
7},
```

W skrypcie:

```
    local lon = resp.coord.lon
```

Po wywołaniu skryptu do zmiennej lokalnej (zmienna skryptu) zostanie przypisana wartość *145.77* .

- Wartość parametru **description**

```
{"coord":
{"lon":145.77,"lat":-16.92},
"weather":[{"id":803,"main":"Clouds","description":"broken
clouds","icon":"04n"}],
"base":"cmc stations",
"main":
{"temp":293.25,"pressure":1019,"humidity":83,"temp_min":289.82,"temp_max":295.3
7},
```

W skrypcie:

```
local description = resp.weather[1].description
```

Po wywołaniu skryptu do zmiennej lokalnej (zmienna skryptu) zostanie przypisana wartość *„broken clouds”.*

B. XML:

Przykładowa odpowiedź (openweathermap):

```
resp= [[
<current>
   <city id="2643741" name="City of London">
    <coord lon="-0.09" lat="51.51">
    <country>GB</country>
    <sun rise="2015-06-30T03:46:57" set="2015-06-30T20:21:12">
   </city>
   <temperature value="72.34" min="66.2" max="79.88" unit="fahrenheit"/>
   <humidity value="43" unit="%">
   <pressure value="1020" unit="hPa">
   <wind>
    <speed value="7.78" name="Moderate breeze">
    <direction value="140" code="SE" name="SouthEast">
   </wind>
   <clouds value="0" name="clear sky">
   <visibility value="10000">
   <precipitation mode="no">
   <weather number="800" value="Sky is Clear" icon="01d">
   <lastupdate value="2015-06-30T08:36:14">
</current>
]]
```

Jak odczytać:

- Wartość atrybutu id w tagu **city**

```
<current>
    <city id="2643741" name="City of London">
    <coord lon="-0.09" lat="51.51">
    <country>GB</country>
<sun rise="2015-06-30T03:46:57" set="2015-06-30T20:21:12">
</city>
```

W skrypcie:

```
local  city_id = resp[1].id
```

Po wywołaniu skryptu do zmiennej lokalnej (zmienna skryptu) zostanie przypisana wartość *2643741*.

- Wartość znajdująca się pomiędzy tagiem **country**

```
 <current>
    <city id="2643741" name="City of London">
     <coord lon="-0.09" lat="51.51">
     <country>GB</country>
     <sun rise="2015-06-30T03:46:57" set="2015-06-30T20:21:12">
   </city>
```

W skrypcie:

```
local  country = resp[1][2][1]
```

Po wywołaniu skryptu do zmiennej lokalnej (zmienna skryptu) zostanie przypisana wartość *„GB”*.

- Nazwa tagu **country**

```
<current>
 <city id="2643741" name="City of London">
    <coord lon="-0.09" lat="51.51">
    <country>GB</country>
    <sun rise="2015-06-30T03:46:57" set="2015-06-30T20:21:12">
   </city>
```

W skrypcie:

```
local nameTag = resp[1][2].xmlTag
```

Po wywołaniu skryptu do zmiennej lokalnej (zmienna skryptu) zostanie przypisana wartość „country”.

2.1.3. Przygotowanie nagłówków zapytania (RequestHeaders) / odczyt nagłówków odpowiedzi (ResponseHeaders)

W celu ustawienia nagłówków zapytania należy utworzyć skrypt i określić zmienną lokalną skryptu.

Przykłady ustawienia:

- tablica par nazwa / wartość, pod kolejnymi indeksami zawarte są kolejne pary składające się z nazwy nagłówka i wartości

```
local header = {
    { name = 'Month', value = 'April' },
    { name = 'Year', value = '2023' },
    { name = 'Name', value = 'Home' },
}

GATE->HttpRequest->RequestHeaders = header
```

- ciąg znaków (kolejne nagłówki oddzielone są przy pomocy ‘\r\n’)

```
local header =
    'Month: April\r\n' ..
    'Year: 2023\r\n' ..
    'Name: Home\r\n'

GATE->HttpRequest->RequestHeaders = header
```

Odpowiedzi `ResponseHeaders` zwracane są w formie tablicy lua. Przykładowy skrypt służący odczytu `ResponeHeaders`

```
local header = GATE->HttpRequest->ResponeHeaders

for i,v in ipairs(header) do
    print(v.name .. ': ' .. v.value)
end
```

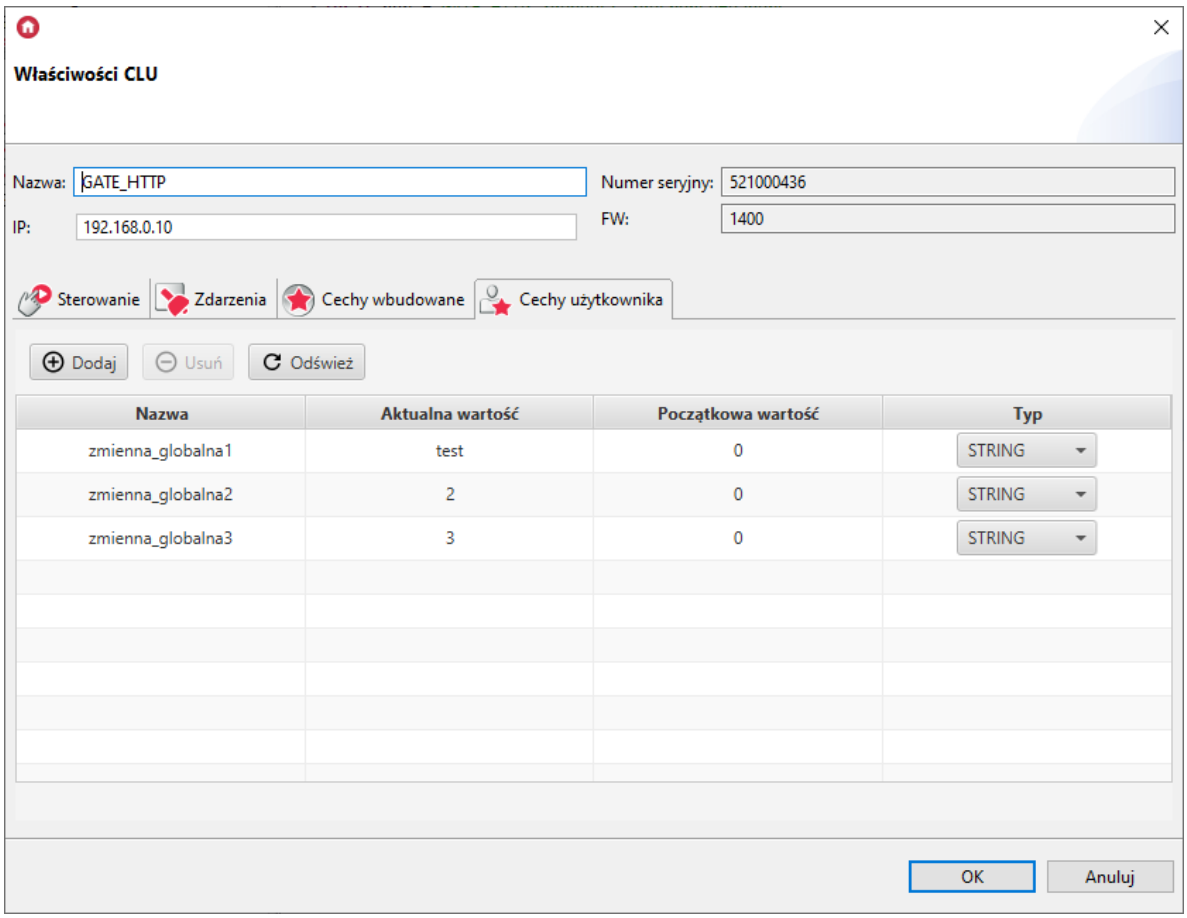
Wartości przechowywane są w zmiennych lokalnych skryptu. Aby użyć wartości w systemie należy wyciągnąć wartości do zmiennych globalnych. Przykładowo:

- tworzymy trzy zmienne globalne: `zmienna_globalna1`, `zmienna_globalna2`, `zmienna_globalna3`
- tworzymy skrypt, gdzie `test1`, `test2`, `test3` są to nazwy nagłówków otrzymanych w odpowiedzi

```
local header = GATE_HTTP->Request->ResponseHeaders

for i,v in ipairs(header) do
    if v.name == 'test1' then
        zmienna_globalna1 = v.value
    end
    if v.name == 'test2' then
        zmienna_globalna2 = v.value
    end
    if v.name == 'test3' then
        zmienna_globalna3 = v.value
    end
end
```

Po wywołaniu skryptu zwracane są wartości poszczególnych nagłówków:



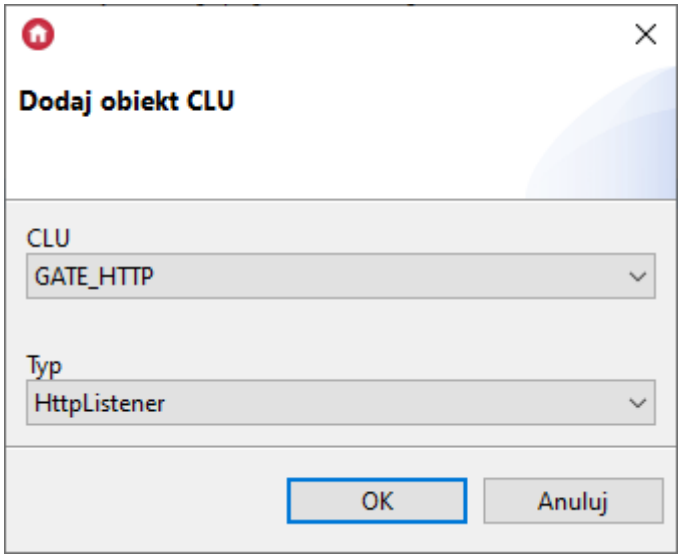
2.2.1. HttpListener

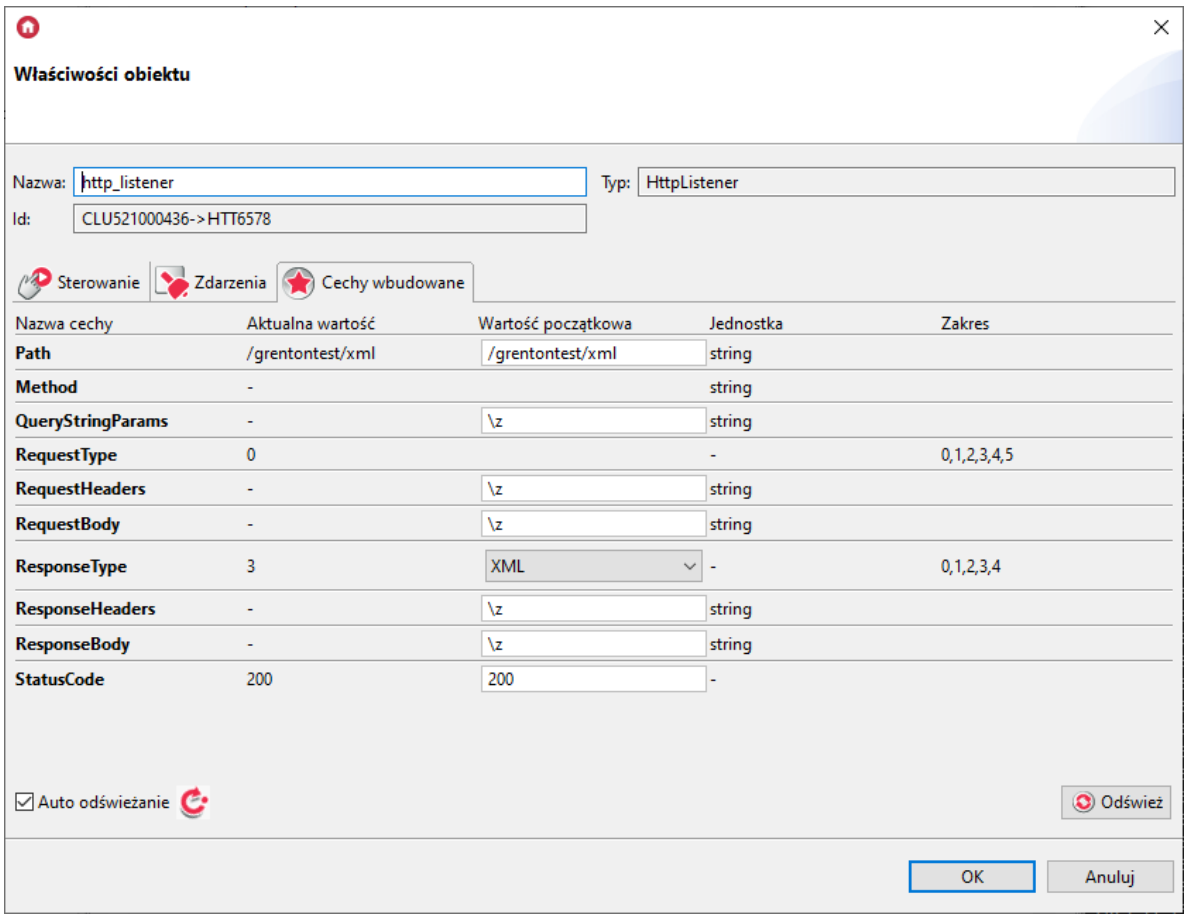
Obiekt HttpListener służy do otrzymywania zapytań HTTP (typu GET, POST). Wysyłana odpowiedź zwrotna może być serializowana do jednego ze standardowych typów m.in. JSON, XML. W obiekcie HttpListener ważne jest, aby na każdy przychodzący Request odesłać odpowiedź (Response).

W przypadku nasłuchiwania na zapytanie Request do modułu Gate - przykładowo (korzystając np. z przeglądarki internetowej):

GET 192.168.4.12/grentontest/xml

Należy utworzyć obiekt wirtualny HttpListener





W obiekcie HttpListener należy ustawić następujące parametry:

- **Path:** /grentontest/xml
- **ResponseType:** XML
- **StatusCode:** 200

> **Uwaga!**
> Cechy opisane jako nieustawialne są cechami zawierającymi odpowiedzi. Wartości początkowe tych cech należy pozostawić niezmienione. Wszelkie operacje na tych zmiennych należy wykonywać na skryptach (oraz zmiennych lokalnych)

Do zdarzenia OnRequest należy utworzyć skrypt, który będzie tworzył poprawną odpowiedź i wysyłał ją zwrotnie.

2.2.2. Przygotowanie odpowiedzi wysyłanej do serwera

Odpowiedź jest tworzona w zmiennej lokalne resp.

Po przygotowaniu odpowiedzi należy ją ustawić dla cechy ResponseBody(resp), a następnie wysłać za pomocą metody SendResponse()

A. XML:

Aby w odpowiedzi wysłać wartość danej cechy:

```
local resp ="<clu><temperature>" ..CLUZ->x103478262_ONEW_SENSOR1->Value.."
</temperature></clu>"
GATE_2->Listener_XML->SetResponseBody(resp)
GATE_2->Listener_XML->SendResponse()
```

Przesłana odpowiedź wygląda następująco:

```
<clu>
  <temperature>22.5</temperature>
</clu>
```

B.JSON:

```
local resp = {
Temp = CLUZ->x103478262_ONEW_SENSOR1->Value
}
GATE_2->Listener_JSON->SetResponseBody(resp)
GATE_2-> Listener_JSON->SendResponse()
```

Przesłana odpowiedź wygląda następująco:

```
{"Temp":22.6}
```

2.2.3. Odczyt wartości kluczy z parametru querystringparams

Zgodnie z opisem cechy QueryStringParams jej wartość nie jest ustawialna, można odczytać ją w skrypcie. Jeżeli w zapytaniu zostaje wysłane querystring z kluczami (keys), to z poziomu skryptu można odczytać daną wartość - jest zapisana w postaci tabeli.

Poszczególne wartości kluczy można uzyskać na zasadzie:

```
value1 = qs.klucz1
```

Dla otrzymanego zapytania:

192.168.1.12/grentontest/query?light1=on&light2=off&light3=on

Należy utworzyć skrypt:

```
local qs = HTTP_L->grentontest_query_listener->QueryStringParams

local test0 = qs.light1
local test1 = qs.light2
local test2 = qs.light3

HTTP_L->grentontest_query_listener->SetResponseBody()
HTTP_L->grentontest_query_listener->SendResponse()
```

Wszystkie wartości kluczy zostaną zapisane w zmiennych lokalnych (test0, test1, test2).

2.2.4. Przygotowanie nagłówków odpowiedzi (ResponseHeaders) / odczyt nagłówków zapytania (RequestHeaders)

W celu ustawienia nagłówków odpowiedzi `ResponeHeaders` należy utworzyć skrypt i określić zmienną lokalną skryptu.

Przykłady ustawienia:

- tablica par nazwa / wartość, pod kolejnymi indeksami zawarte są kolejne pary składające się z nazwy nagłówka i wartości

```
local header = {
    { name = 'Month', value = 'April' },
    { name = 'Year', value = '2023' },
    { name = 'Name', value = 'Home' },
}

GATE->HttpListener->ResponseHeaders = header
```

- ciąg znaków (kolejne nagłówki oddzielone są przy pomocy '\r\n')

```
local header =
    'Month: April\r\n' ..
    'Year: 2023\r\n' ..
    'Name: Home\r\n'

GATE->HttpListener->ResponseHeaders = header
```

`RequestHeaders` zwracane są w formie tablicy lua. Przykładowy skrypt służący odczytu `RequestHeaders`:

```
local header = GATE->HttpRequest->ResponeHeaders

for i,v in ipairs(header) do
    print(v.name .. ': ' .. v.value)
end
```

Wartości przechowywane są w zmiennych lokalnych skryptu. Aby użyć wartości w systemie należy wyciągnąć wartości do zmiennych globalnych. Przykład wykonania opisany został w punkcie 2.1.3.

2.3.1. Timer

Timery są wirtualnymi obiektami tworzonymi w ramach danego modułu GATE. Timery mogą być wykorzystywane wszędzie tam, gdzie potrzebne jest wywołanie metody po określonym czasie lub też jej cykliczne wywoływanie.

> **Uwaga!**
> Zalecane jest wykorzystywanie obiektu Timer w przypadku cyklicznego wysyłania zapytań za pomocą obiektu HttpRequest.

Timer może pracować w dwóch trybach:

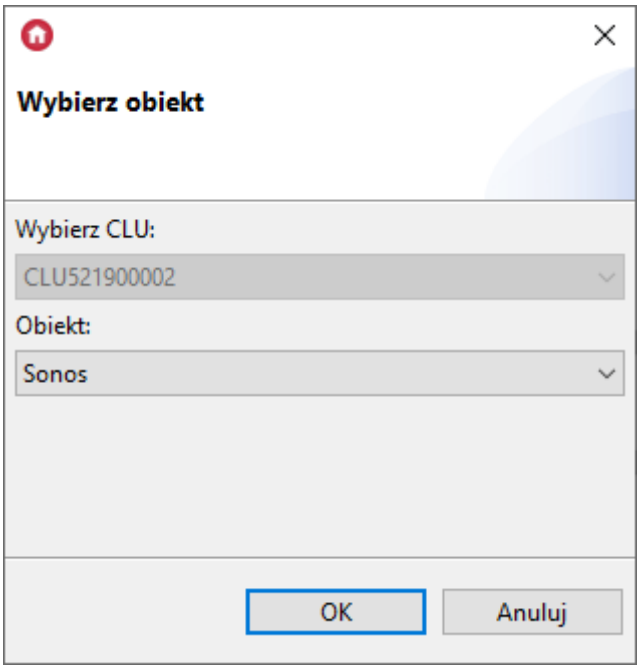
- `Countdown`
  Po wystartowaniu, odlicza ustalony czas. Po zakończeniu odliczania uruchamiana jest metoda powiązana ze zdarzeniem `OnTimer`, a timer zatrzymuje się i nie odlicza, aż do następnego uruchomienia metodą `Start`.
- `Interval`
  Timer cykliczny - po starcie zaczyna odliczać ustawiony czas. Po jego upłynięciu timer wywołuje metodę powiązaną ze zdarzeniem `OnTimer`, a sam timer ponownie zaczyna odliczać zadany czas. Sytuacja powtarza się, aż do momentu zatrzymania metodą `Stop`.

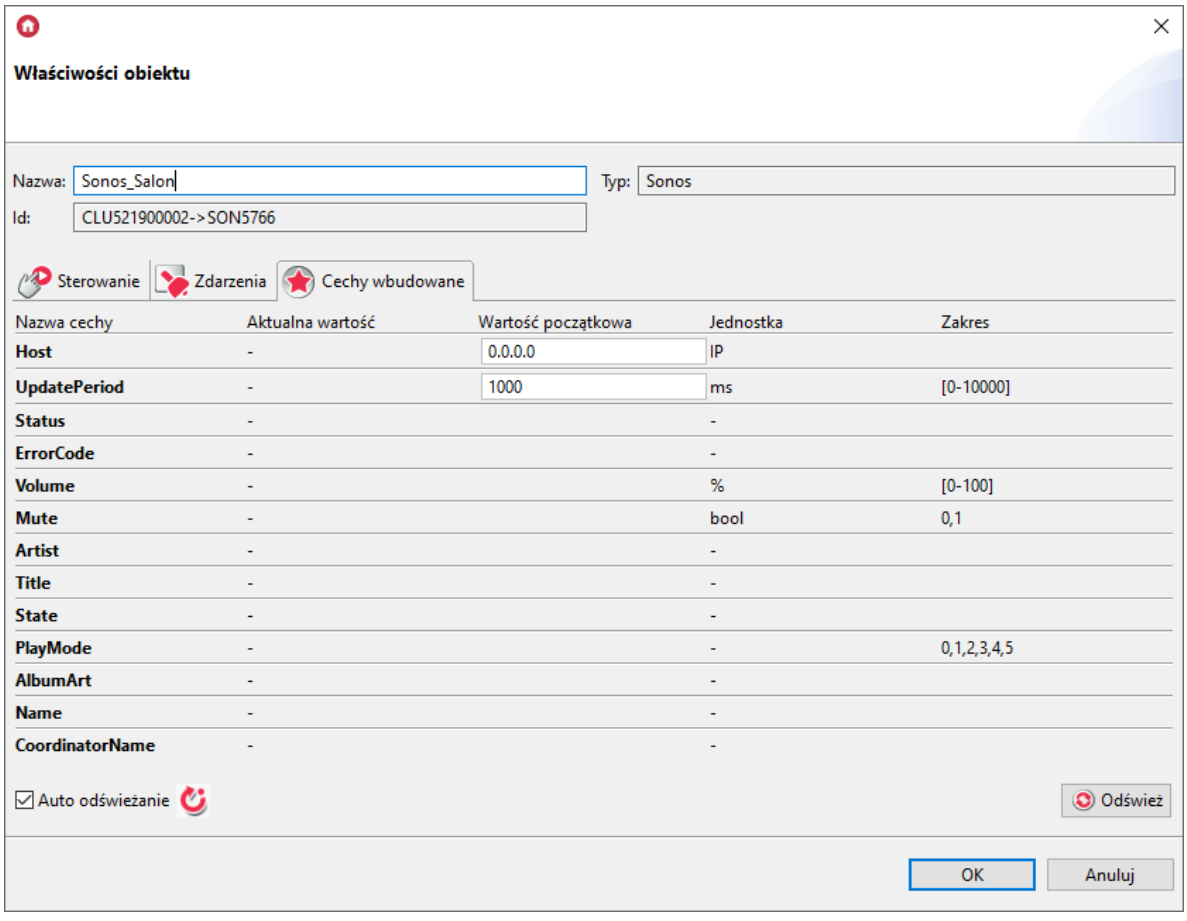
2.4.1. Sonos

Obiekt wirtualny Sonos służy do integracji głośników marki Sonos z systemem Grenton za pomocą modułu GATE.

Przed przystąpieniem do integracji głośnika z systemem należy skonfigurować głośnik w sieci lokalnej za pomocą dedykowanej aplikacji producenta i odczytać jego przydzielony adres IP.

W celu połączenia głośnika z systemem należy utworzyć nowy obiekt wirtualny Sonos:

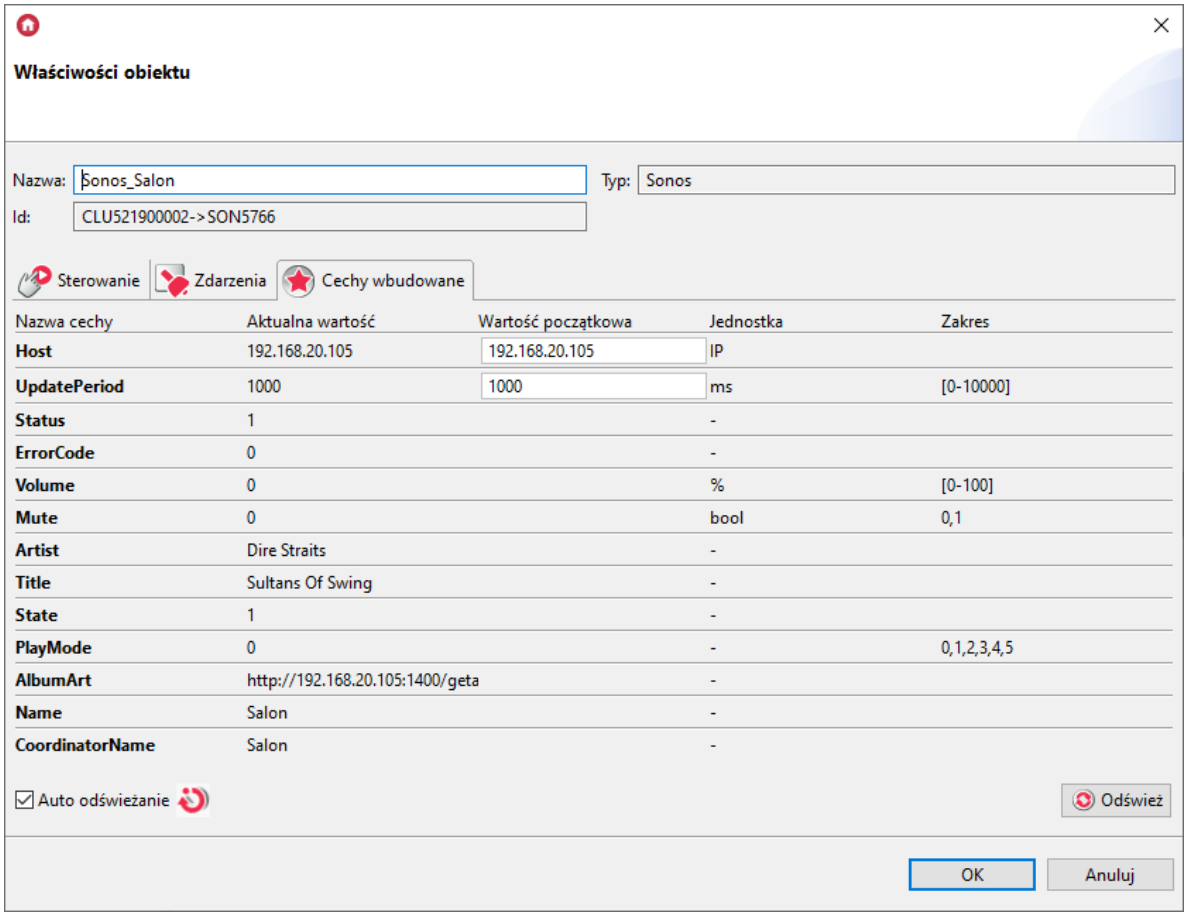




W cechach wbudowanych obiektu należy ustawić następujące parametry:

- **Host:** np. 192.168.20.105 (adres IP głośnika)
- **UpdatePeriod:** 1000

Po wysłaniu konfiguracji cecha wbudowana `Status` powinna przyjąć wartość `1`. Świadczy to o poprawnym połączeniu głośnika z systemem.



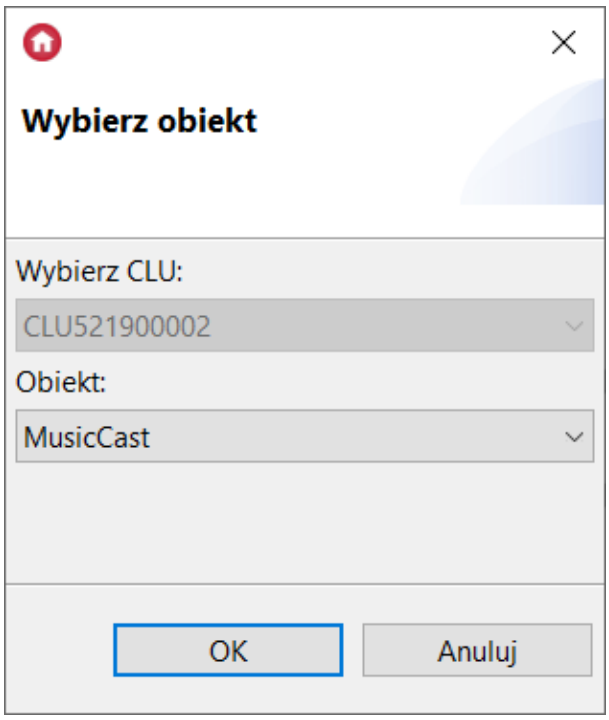
Uwaga!
W przypadku korzystania z większej liczby obiektów mogą występować problemy wynikające z ograniczonej przepustowości sieci i/lub urządzeń. W takiej sytuacji, wraz z dalszym wzrostem liczby utworzonych obiektów, zalecane jest zwiększanie wartości cechy UpdatePeriod.

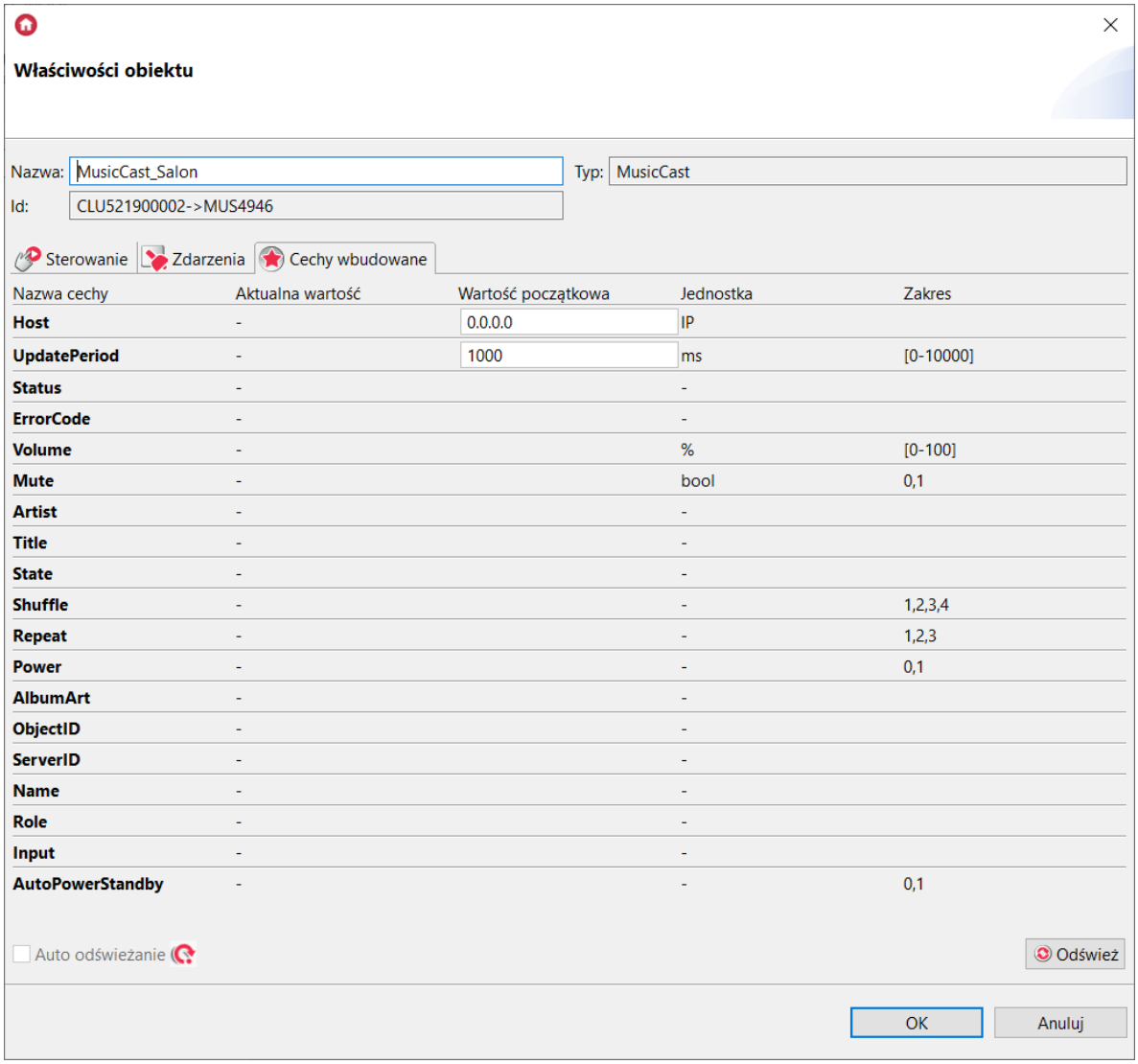
2.5.1. MusicCast

Obiekt wirtualny MusicCast służy do integracji głośników marki Yamaha z systemem Grenton za pomocą modułu GATE.

Przed przystąpieniem do integracji głośnika z systemem należy skonfigurować głośnik w sieci lokalnej za pomocą dedykowanej aplikacji producenta i odczytać jego przydzielony adres IP.

W celu połączenia głośnika z systemem należy utworzyć nowy obiekt wirtualny MusicCast:



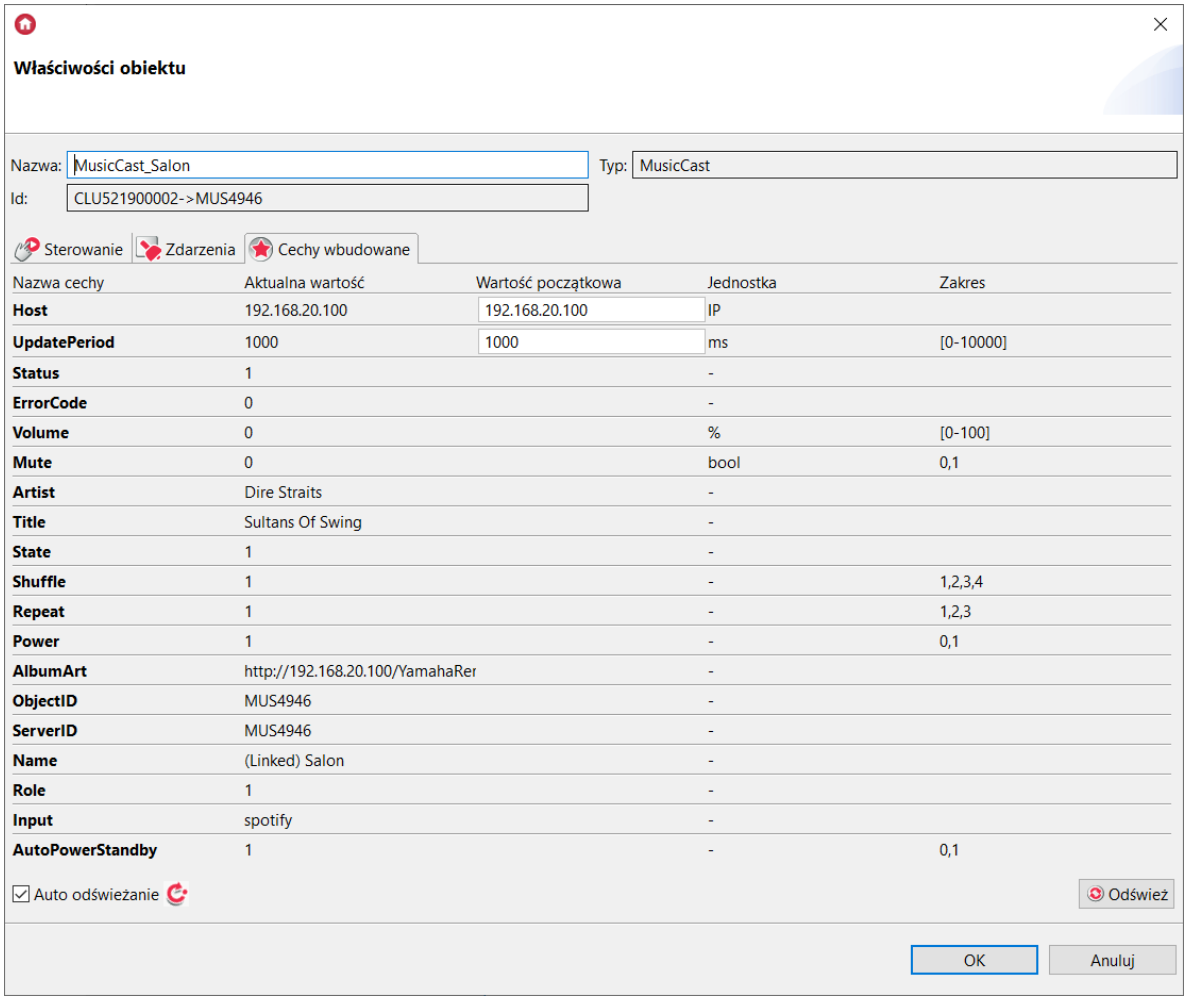


Nazwa cechy	Aktualna wartość	Wartość początkowa	Jednostka	Zakres
Host	-	0.0.0.0	IP	
UpdatePeriod	-	1000	ms	[0-10000]
Status	-		-	
ErrorCode	-		-	
Volume	-		%	[0-100]
Mute	-		bool	0,1
Artist	-		-	
Title	-		-	
State	-		-	
Shuffle	-		-	1,2,3,4
Repeat	-		-	1,2,3
Power	-		-	0,1
AlbumArt	-		-	
ObjectID	-		-	
ServerID	-		-	
Name	-		-	
Role	-		-	
Input	-		-	
AutoPowerStandby	-		-	0,1

W cechach wbudowanych obiektu należy ustawić następujące parametry:

- **Host:** np. 192.168.20.100 (adres IP głośnika)
- **UpdatePeriod:** 1000

Po wysłaniu konfiguracji cecha wbudowana `Status` powinna przyjąć wartość `1`. Świadczy to o poprawnym połączeniu głośnika z systemem.

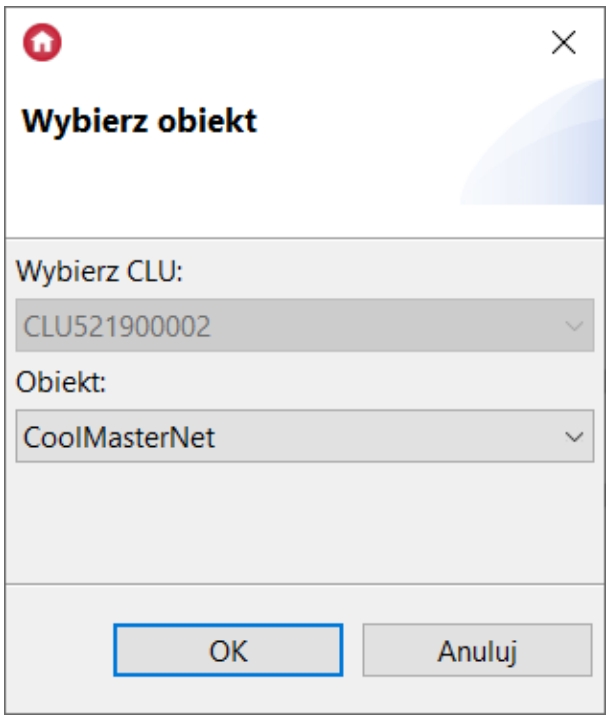


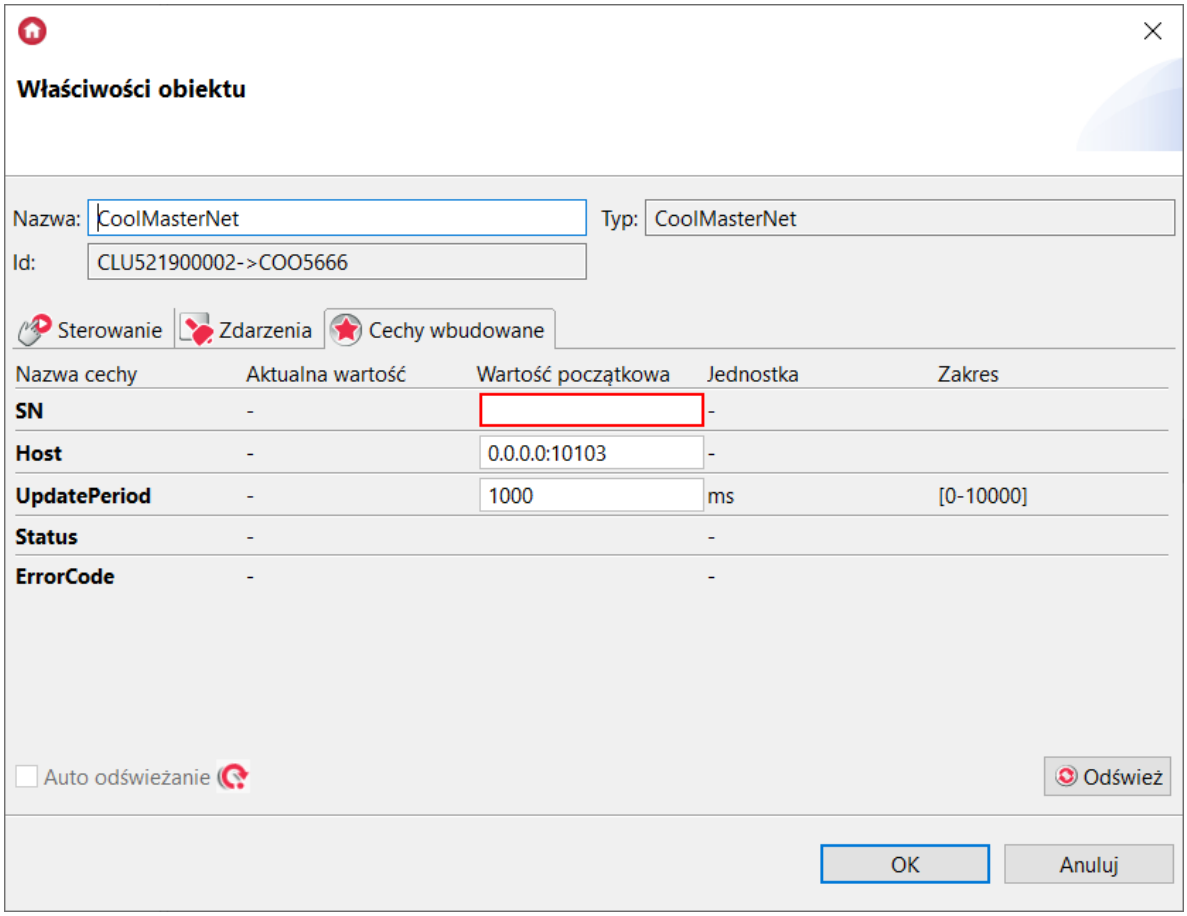
> **Uwaga!**
> W przypadku korzystania z większej liczby obiektów mogą występować problemy wynikające z ograniczonej przepustowości sieci i/lub urządzeń. W takiej sytuacji, wraz z dalszym wzrostem liczby utworzonych obiektów, zalecane jest zwiększanie wartości cechy UpdatePeriod.

2.6.1. CoolMasterNet

Obiekt wirtualny CoolMasterNet służy do integracji systemu Grenton z jednostkami CoolAutomation (CoolMasterNet, CoolLinkHub) w celu sterowania klimatyzatorem lub grupą klimatyzatorów.

W celu połączenia jednostki z systemem należy utworzyć nowy obiekt wirtualny CoolMasterNet:

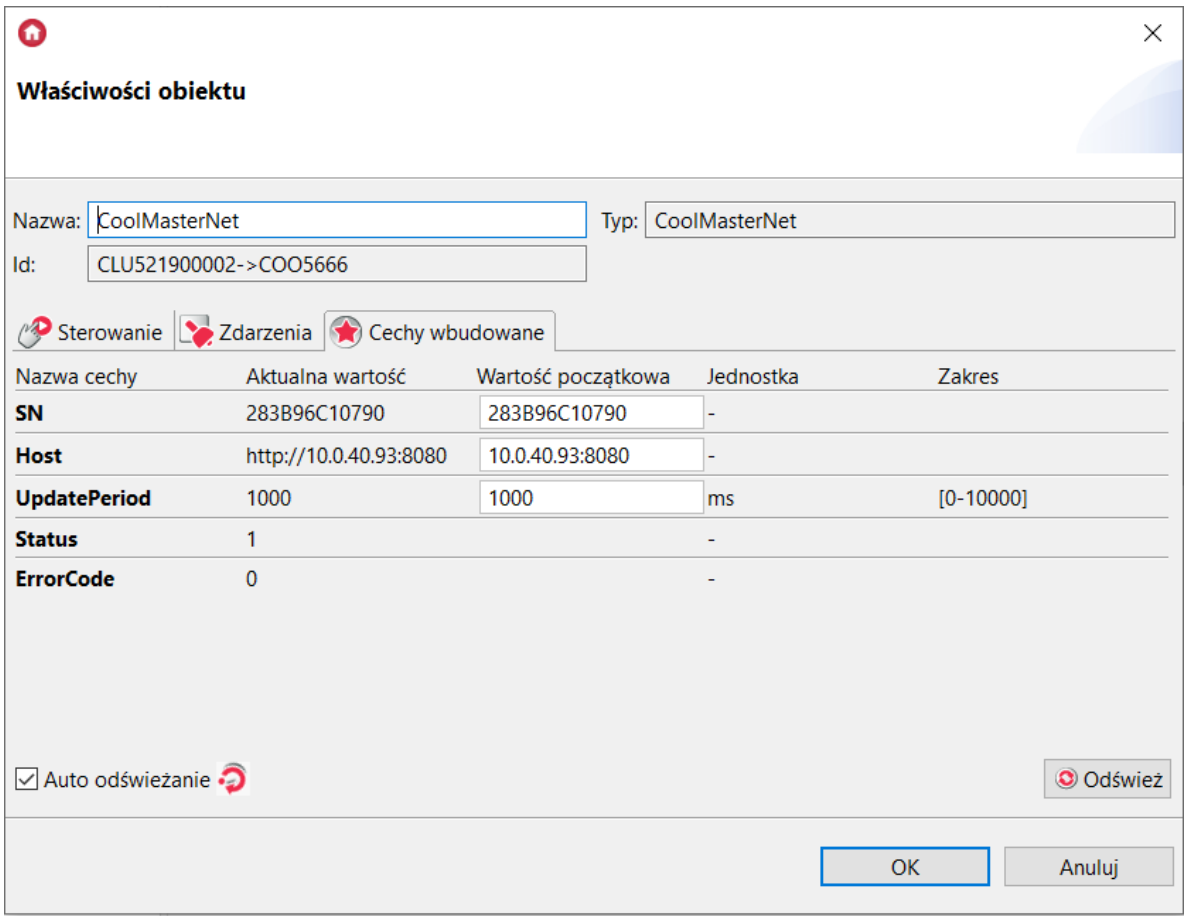




W cechach wbudowanych obiektu należy ustawić następujące parametry:

- **SN:** np. - 283B96C10790 (numer seryjny jednostki)
- **Host:** np. - 192.168.0.213:10103 (adres IP jednostki)

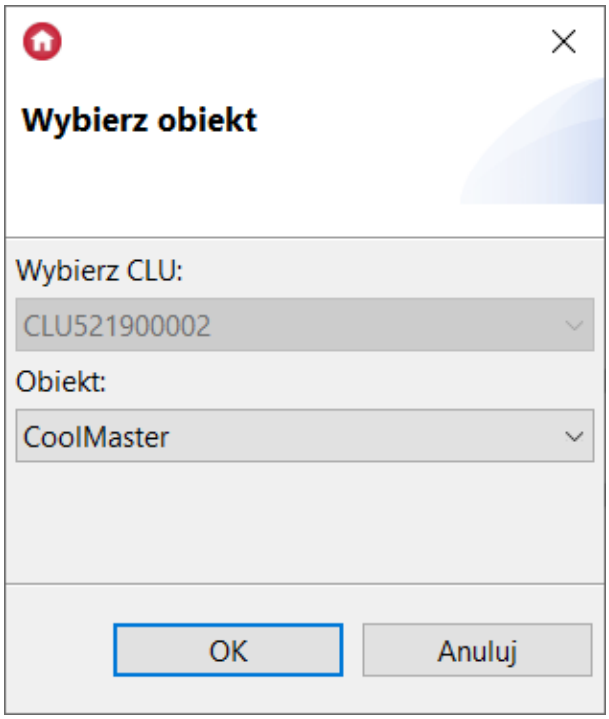
Po wysłaniu konfiguracji cecha `Status` powinna przyjąć wartość `1`. Świadczy to o poprawnym połączeniu jednostki z systemem.

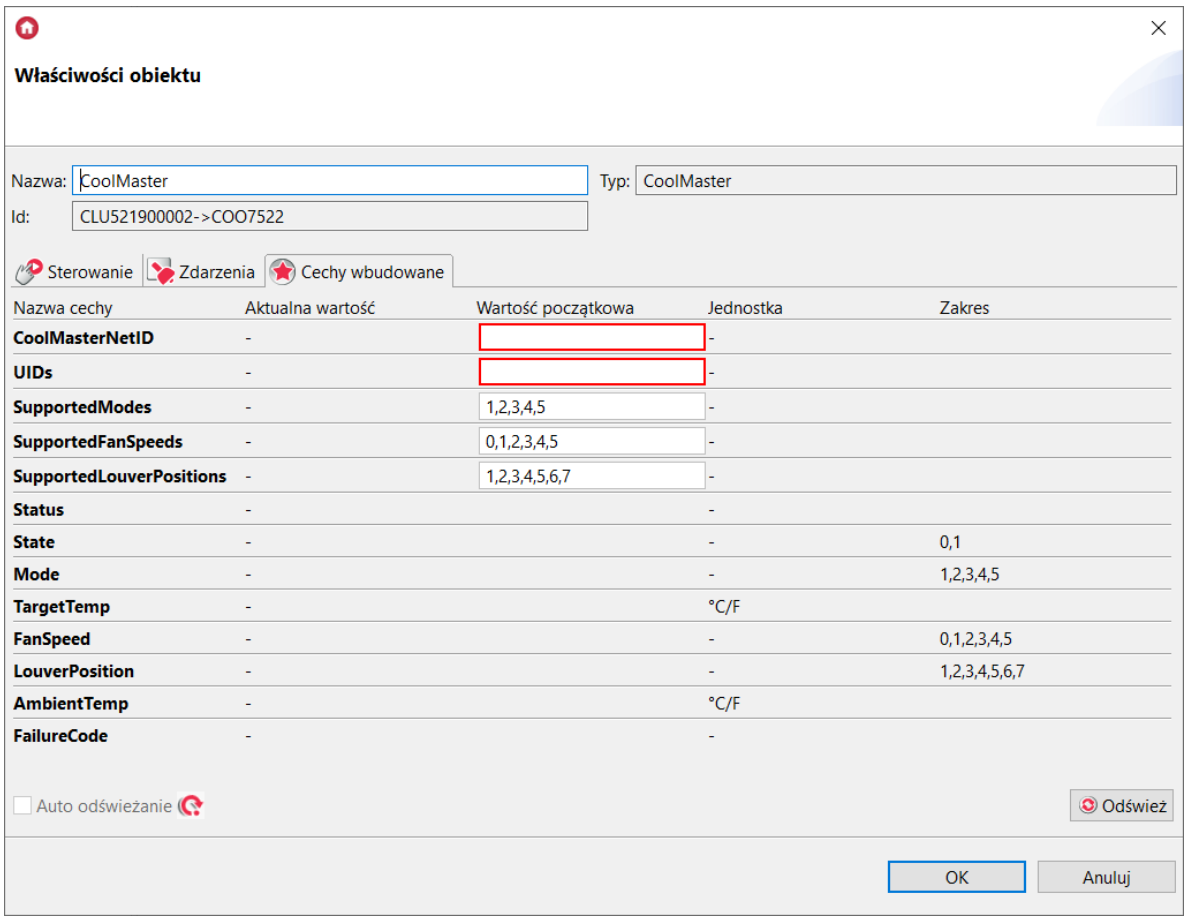


2.7.1. CoolMaster

Obiekt wirtualny CoolMaster służy do sterowania klimatyzatorem lub grupą klimatyzatorów. Aby prawidłowo korzystać z obiektu należy najpierw skonfigurować obiekt wirtualny CoolMasterNet (patrz pkt 2.6.1.).

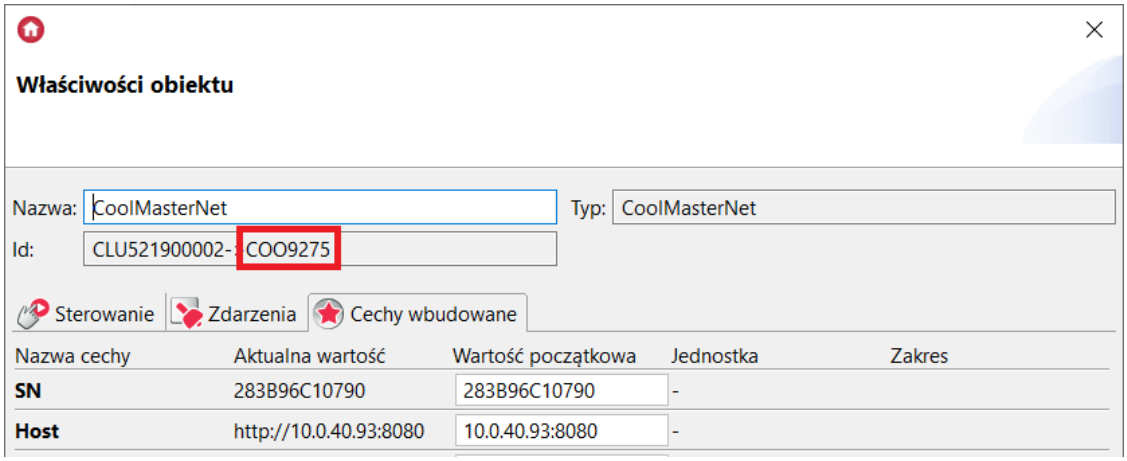
Następnie utworzyć nowy obiekt wirtualny CoolMaster:



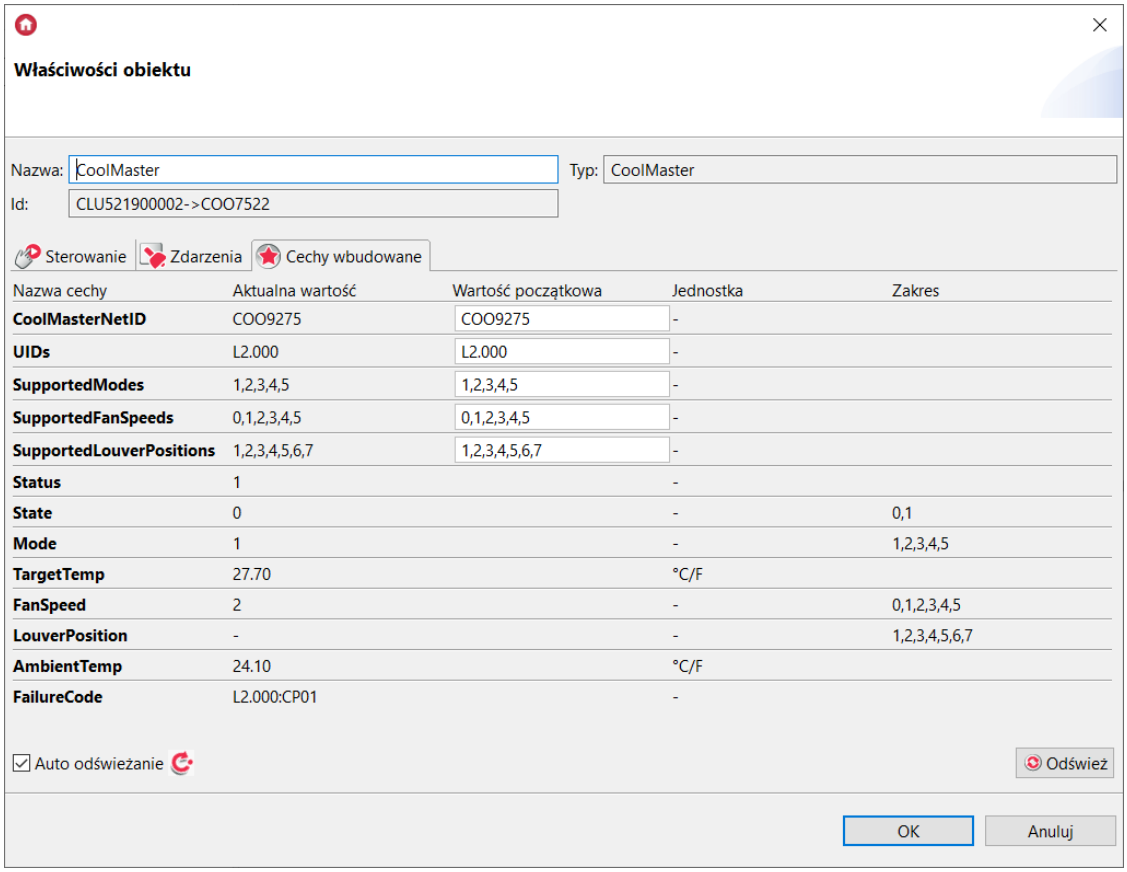


W cechach wbudowanych obiektu należy ustawić:

- **CoolMasterNetID:** np. COO9275 (ID wcześniej utworzonego obiektu CoolMasterNet):



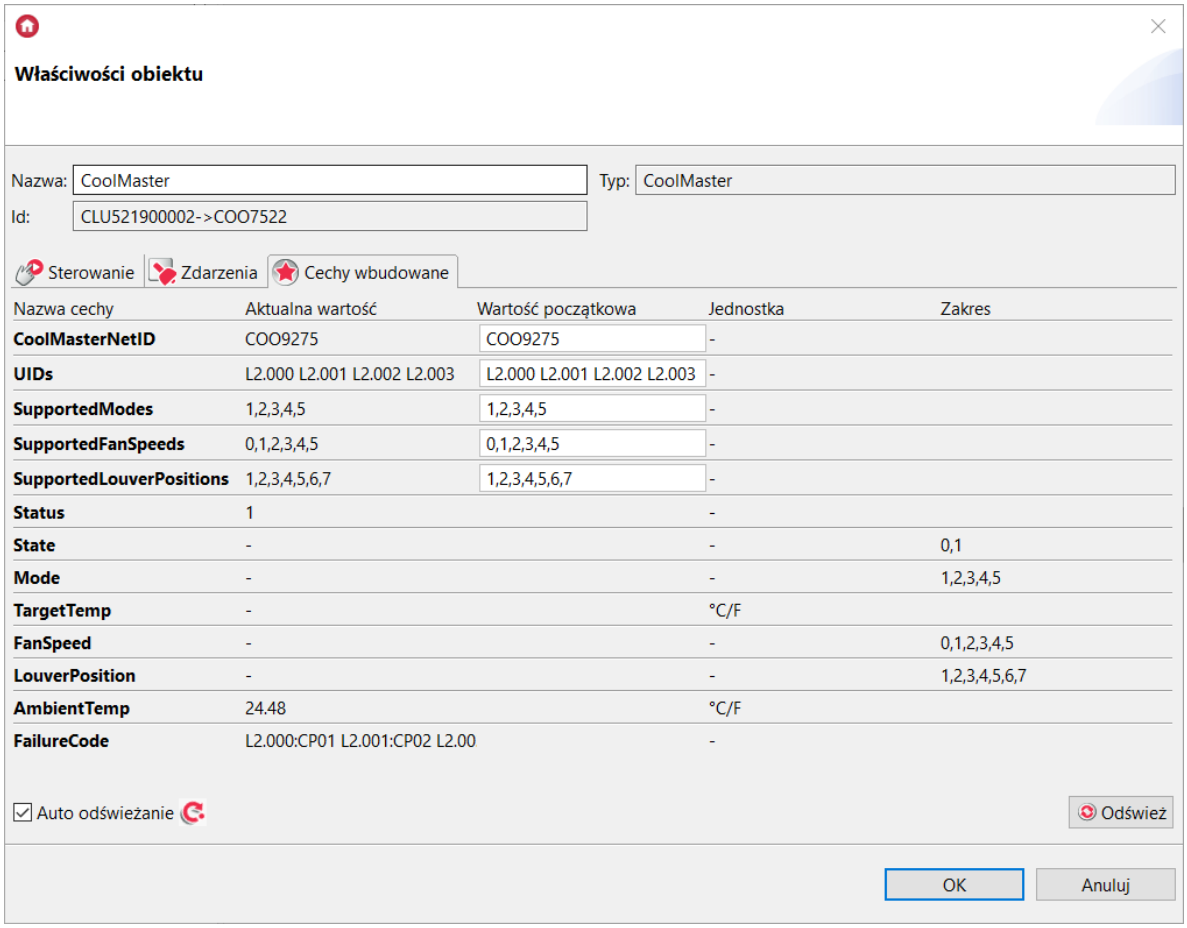
- **UIDs:** np. L2.000 (identyfikator klimatyzatora)



Po wysłaniu konfiguracji cecha `Status` powinna przyjąć wartość `1`, a wartości pozostałych cech powinny zostać poprawnie wczytane.

Sterowanie większą ilością klimatyzatorów za pomocą jednego obiektu CoolMaster

Jeden obiekt CoolMaster może współpracować z wieloma klimatyzatorami podłączonymi do tej samej jednostki. W tym celu w obiekcie CoolMaster w cesze `UIDs`, po spacji należy wpisać kolejne identyfikatory klimatyzatorów:



Uwaga!
Wartość `-` dla cech `State`, `Mode`, `TargetTemp`, `FanSpeed`, `LouverPosition` oznacza, że wartości danej cechy są różne dla co najmniej jednego klimatyzatora z grupy - stan desynchronizacji.

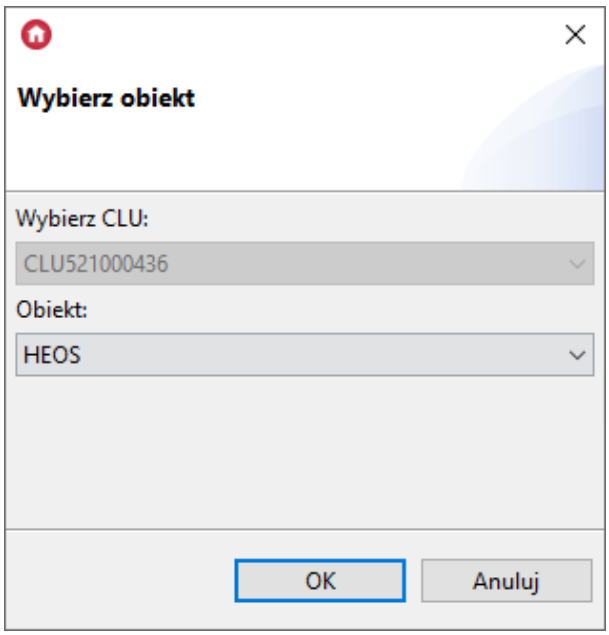
Uwaga!
Cechy `State`, `Mode`, `TargetTemp`, `FanSpeed`, `LouverPosition`, `AmbientTemp`, `FailureCode` przed pierwszym nawiązaniem połączenia z jednostką lub w momencie stanu desynchronizacji nie posiadają wartości, w Object Manager wyświetlane jako `-`. W celu uniknięcia błędów w skryptach przed porównywaniem takiej cechy należy sprawdzić, czy posiada ona wartość:
if(cecha ~= nil)

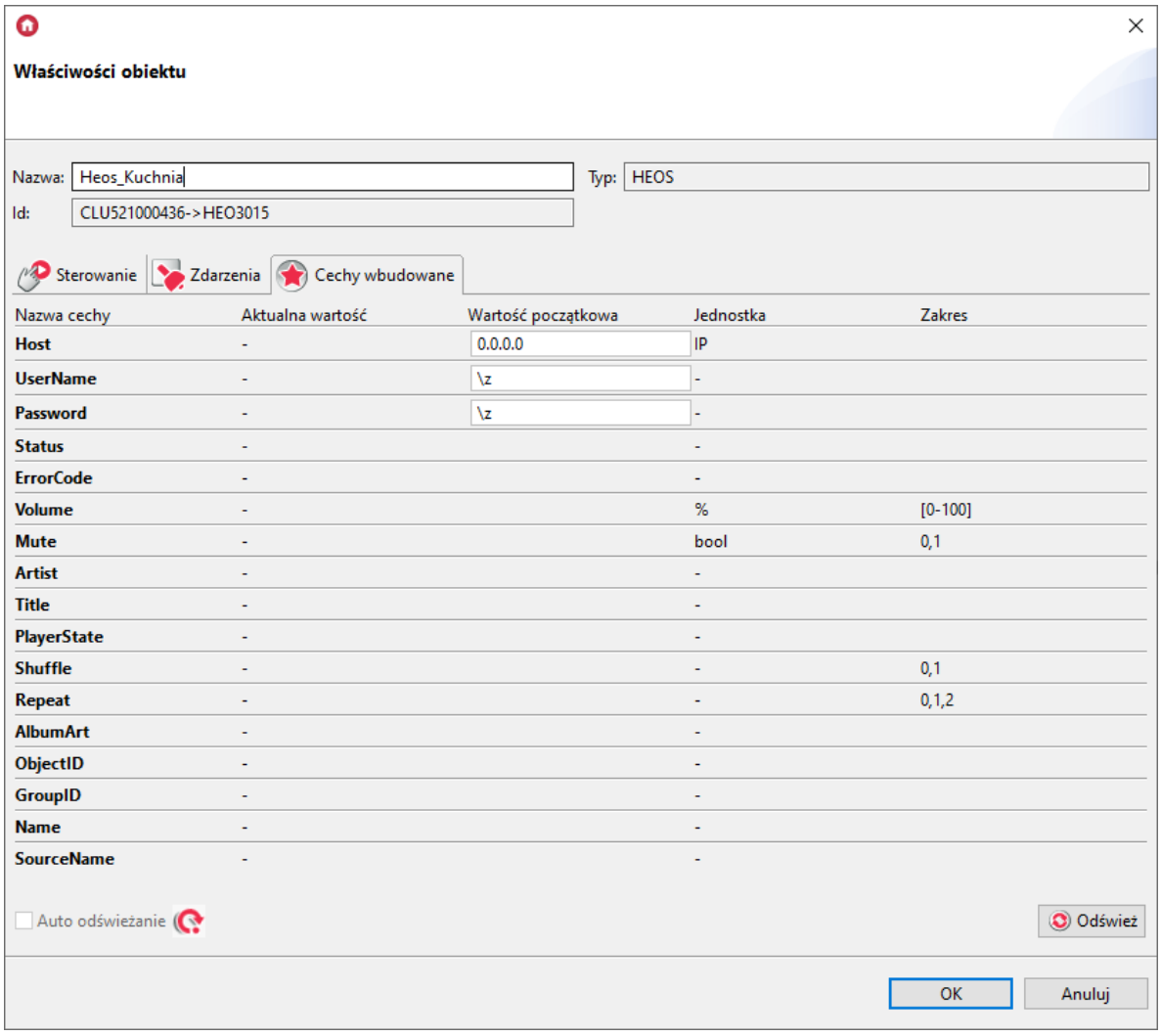
2.8.1. HEOS

Obiekt wirtualny HEOS służy do integracji głośników marki Denon oraz amplitunerów Denon/Marantz AVR posiadających wbudowaną obsługę HEOS z systemem Grenton za pomocą modułu GATE.

Przed przystąpieniem do integracji głośnika z systemem należy skonfigurować głośnik/amplituner w sieci lokalnej za pomocą dedykowanej aplikacji producenta i odczytać jego przydzielony adres IP.

W celu połączenia głośnika z systemem należy utworzyć nowy obiekt wirtualny HEOS:





W cechach wbudowanych obiektu należy ustawić następujące parametry:

- **Host:** np. 192.168.0.4 (adres IP głośnika/amplitunera)

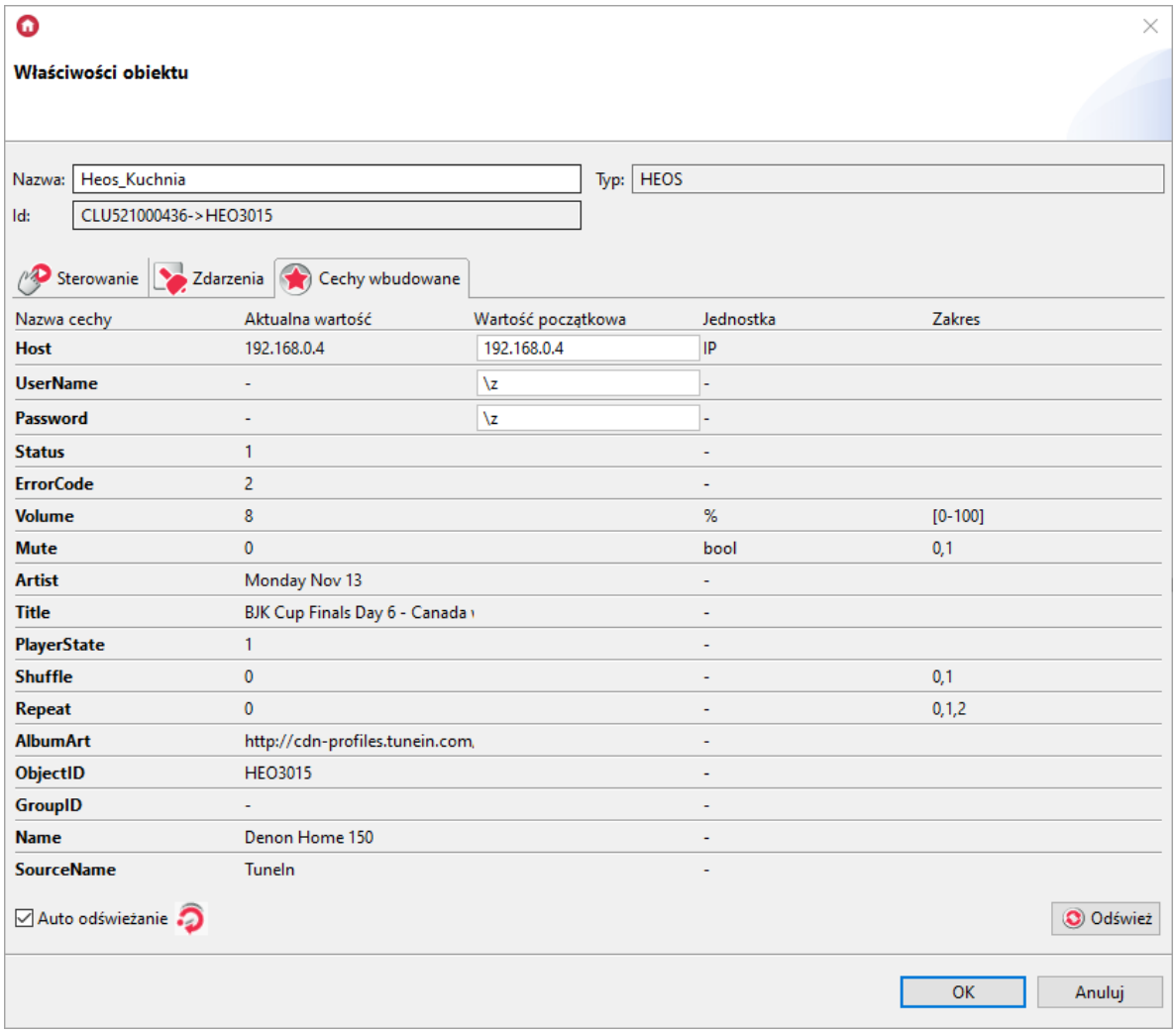
Opcjonalnie można ustawić

- **UserName**: nazwa użytkownika konta Heos
- **Password**: hasło użytkownika konta Heos

> **Uwaga!**

> W celu poprawnego działania metody `PlayPresetStation` ustawienie `UserName` oraz `Password` jest wymagane.

Po wysłaniu konfiguracji cecha wbudowana `Status` powinna przyjąć wartość `1`. Świadczy to o poprawnym połączeniu głośnika z systemem.

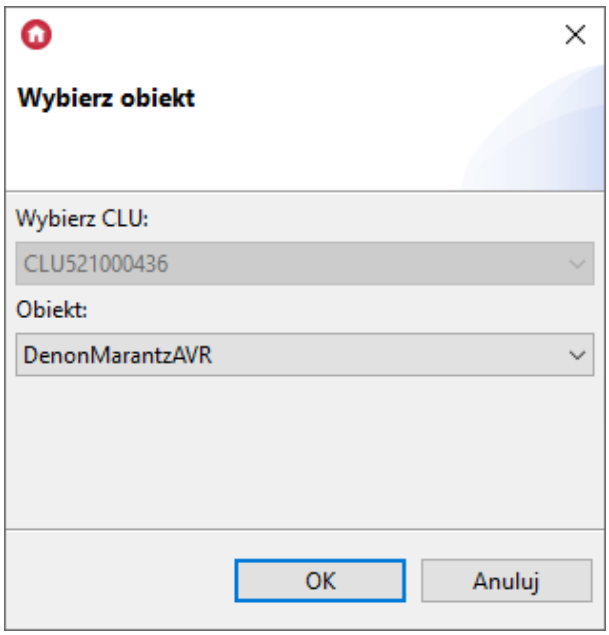


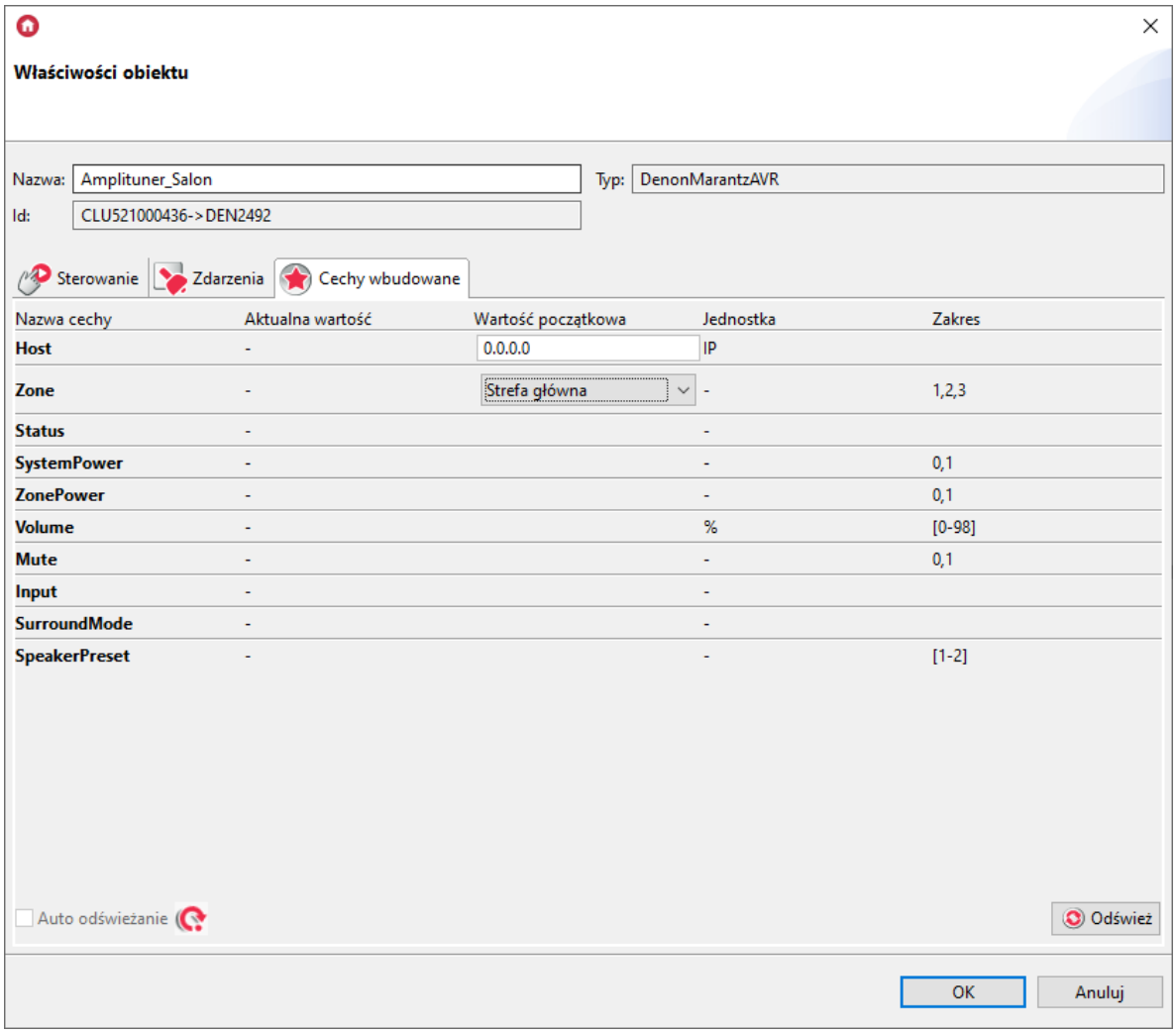
2.9.1. DenonMarantzAVR

Obiekt wirtualny DenonMarantzAVR służy do integracji urządzeń z rodziny Denon oraz Marantz AVR z systemem Grenton za pomocą modułu GATE.

Przed przystąpieniem do integracji urządzenia należy skonfigurować urządzenie w sieci lokalnej za pomocą dedykowanej aplikacji producenta i odczytać jego przydzielony adres IP.

W celu połączenia urządzenia z systemem należy utworzyć nowy obiekt wirtualny DenonMarantzAVR:

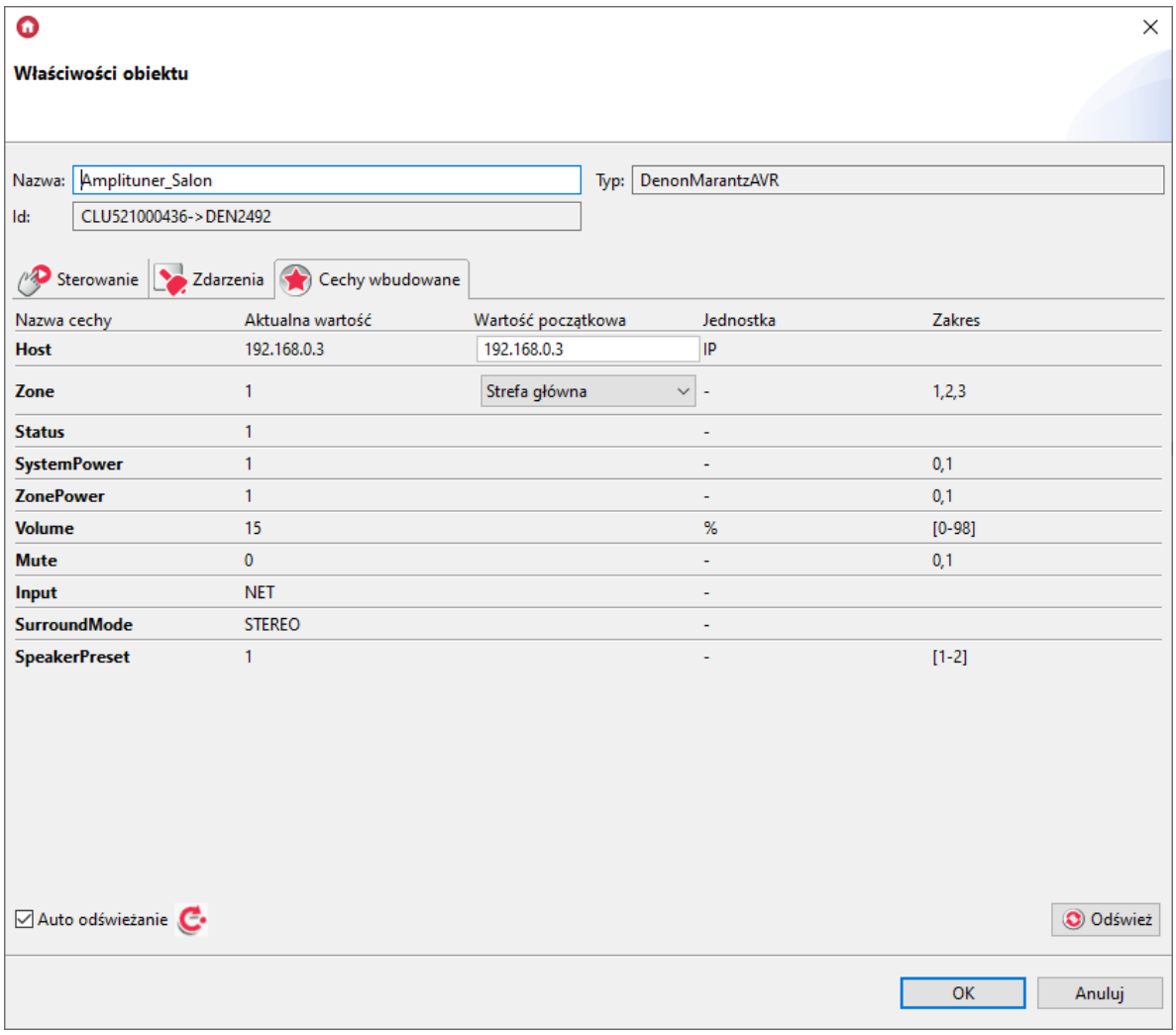




W cechach wbudowanych obiektu należy ustawić następujące parametry:

- **Host:** np. 192.168.0.3 (adres IP urządzenia)
- **Zone:** np. Strefa główna

Po wysłaniu konfiguracji cecha wbudowana `Status` powinna przyjąć wartość `1`. Świadczy to o poprawnym połączeniu urządzenia z systemem.



Uwaga!

W przypadku urządzeń Denon/Maranatz, które nie posiadają podziału na strefy należy ustawić cechę `Zone` = Strefa główna.

Uwaga!

Metoda `SetSystemPower` w przypadku amplitunera posiadającego kilka stref wyłącza i włącza wszystkie dostępne strefy na raz. W celu włączenia/wyłączenia pojedynczej strefy należy użyć metody `SetZonePower`.

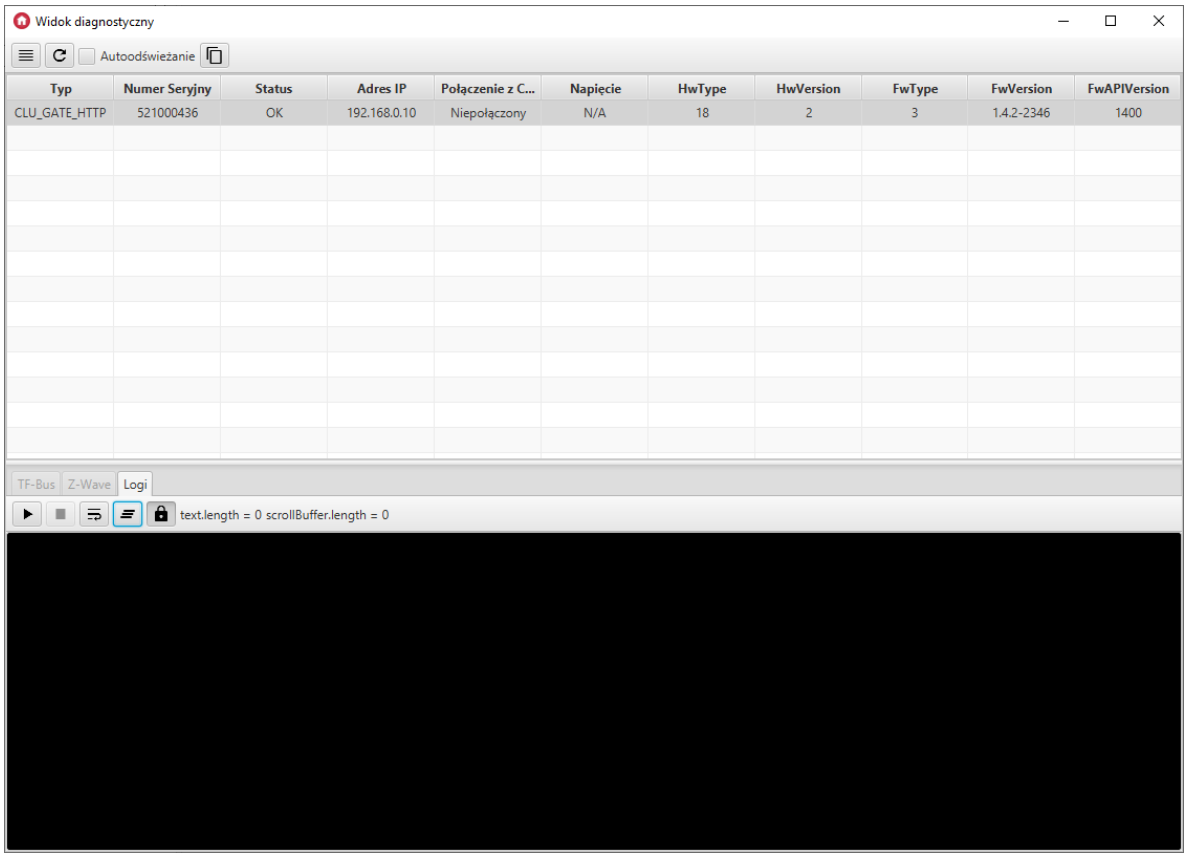
Uwaga!

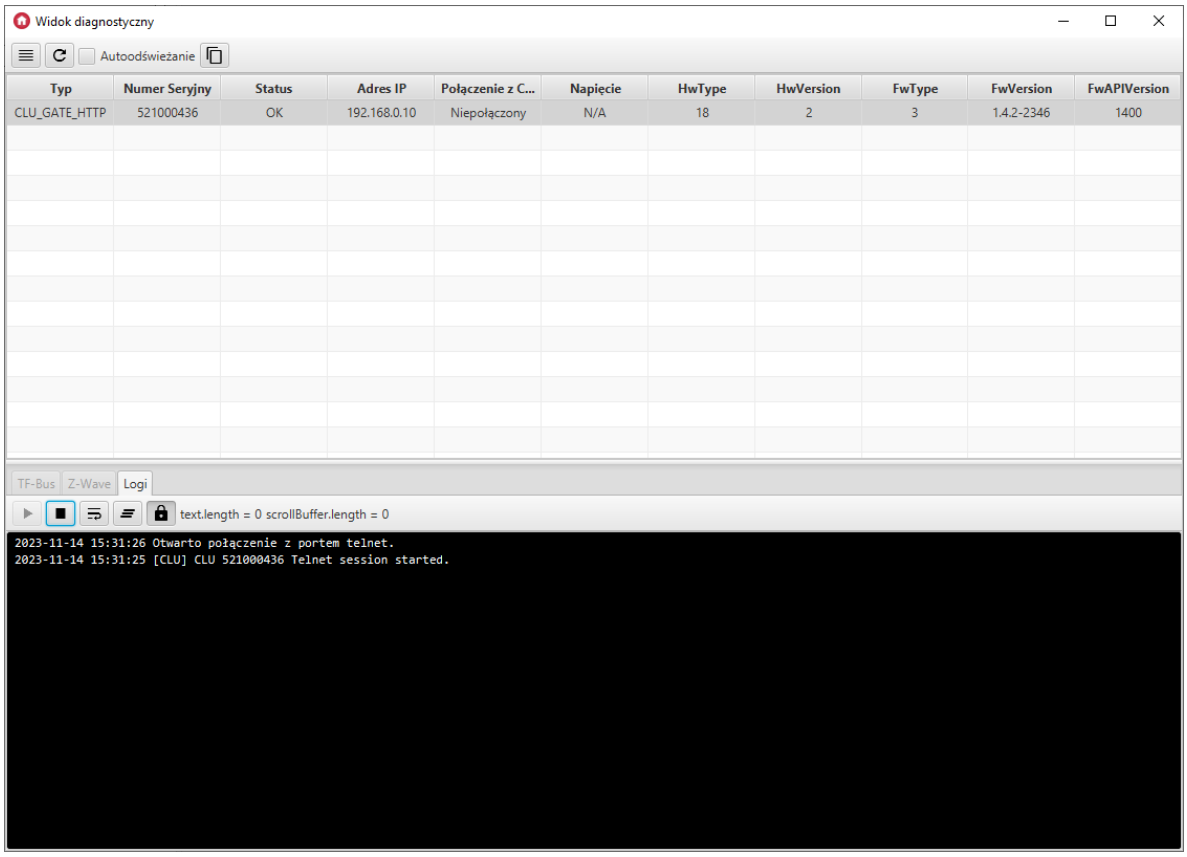
W zależności od typu urządzenia i jego dostępnych funkcji dostępne są wybrane metody. Przed użyciem metody należy zweryfikować wsparcie dla jej funkcji w instrukcji konkretnego urządzenia.

3. Możliwość połączenia z Gate za pomocą TELNET

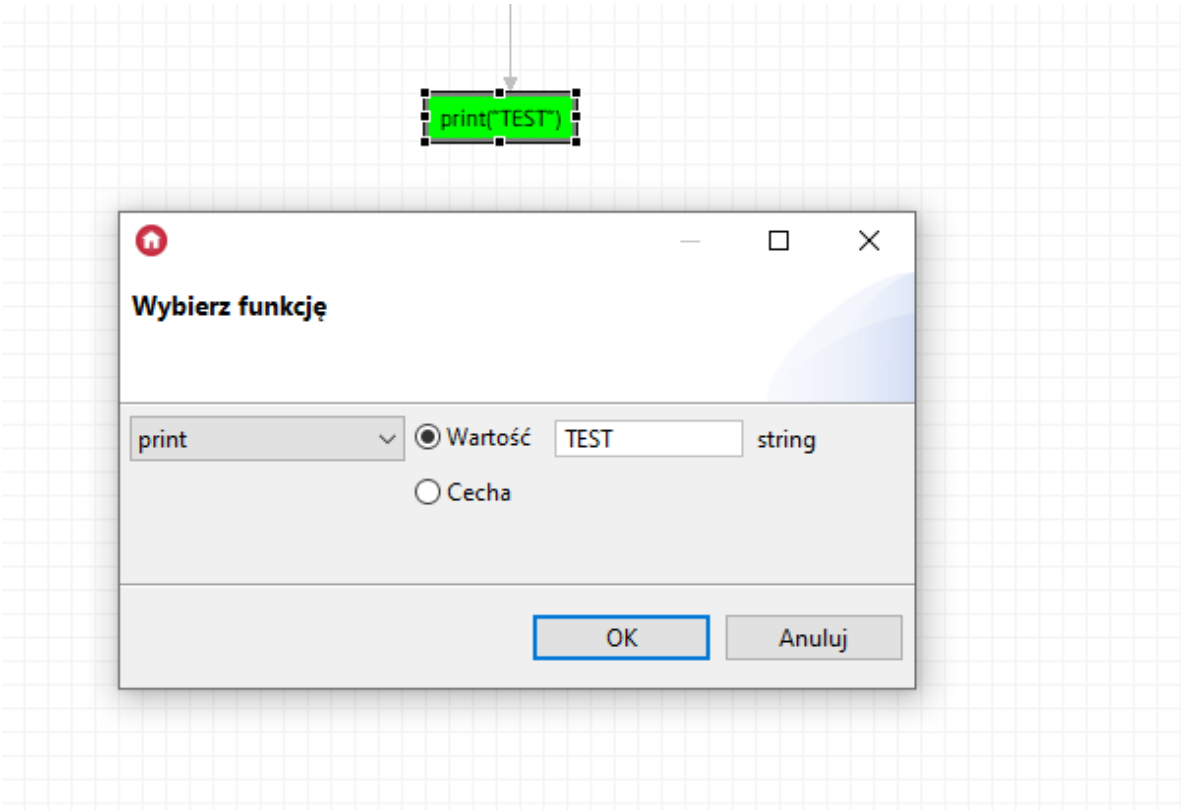
Dla modułu Gate Http możliwy jest podgląd skryptów Lua. W przypadku błędu konfiguracji (tryb emergency) możliwy jest podgląd miejsca błędu w utworzonej konfiguracji LUA. W tym celu należy skorzystać z funkcjonalności logowania. Aby to zrobić należy w obiekcie GATE ustawić poziom logowania za pomocą cechy `TelnetLogLevel` = DEBUG i wysłać konfigurację do modułu.

Następnie należy otworzyć widok diagnostyczny i uruchomić logowanie jak na rysunku poniżej. Więcej informacji odnośnie logowania znajduje się w rozdziale 11. Widok diagnostyczny..





Aby na konsoli wyświetlić np. wartość danej cechy, należy wykorzystać komponent *Blok funkcyjny* i wybrać metodę *Print*, a następnie wybrać żądaną cechę.



4. Kompleksowa integracja z systemami zewnętrznymi przy użyciu urządzenia GATE HTTP

Opis konfiguracji krok po kroku na przykładzie wyjścia przekaźnikowego.

4.1. System

Powiedzmy, że mamy prosty system złożony z następujących elementów:

- CLU Z-Wave - nazwany (Name) "CluZ"
- Moduł przekaźnikowy - na potrzeby przykładu użyjemy jednego wyjścia o nazwie "Relay"
- Gate Http - nazwa "GateHttp"

4.2. Sterowanie wyjściem

W celu umożliwienia sterowania wyjściem przekaźnikowym z zewnętrznego systemu tworzymy nowy obiekt typu HttpListener na GateHttp i konfigurujemy jak poniżej:

- **Name:** RelayControlListener
- **Path:** /relaycontrol

Pozostałe parametry pozostawiamy na razie bez zmian.

Skrypt

Aby obiekt `RelayControlListener` zadziałał należy utworzyć dla skrypt, który będzie obsługiwał przychodzące zapytania Http.

Tutaj warto zauważyć, że z tego skryptu mamy dostęp do całego systemu i wszystkich jego funkcjonalności. To otwiera praktycznie nieograniczone możliwości, ale też rodzi pewne ryzyka, zwłaszcza jeśli funkcjonalność Gate'a nie jest dobrze przemyślana. Dlatego zwracamy szczególną uwagę, że implementując funkcjonalność Gate'a należy dobrze przemyśleć sposób działania jaki chcemy osiągnąć oraz to jak działanie Gate'a może zależeć albo wpływać na inne elementy systemu. Przykłady takiego podejścia będą też omawiane dalej.

Wracając do skryptu kontrolującego Relay. Chcemy mieć możliwość załączenia lub rozłączenia Relay'a wysyłając do niego oczekiwany stan (On/Off) albo wykonania metody `Switch`. Takie podejście do implementacji umożliwia podłączenie do niego zarówno kontroli typu przełącznika bistabilnego jak i monostabilnego.

Przechodząc do działania, tworzymy na GateHttp skrypt o nazwie `RelayControlOnRequest`, i w trybie edycji kodu wrzucamy to co poniżej:

```
-- RelayControlOnRequest()
local data = GateHttp->RelayControlListener->QueryStringParams
if data == nil then
    CluZ->Relay->Switch(0)
else
    if data.cmd == "setValue" then
        local val = tonumber(data.val)
        if(val == 1) then
            CluZ->Relay->SwitchOn(0)
        elseif(val == 0) then
            CluZ->Relay->SwitchOff(0)
        end
    end
end

GateHttp->RelayControlListener->StatusCode = 200
GateHttp->RelayControlListener->ResponseBody = "OK"
GateHttp->RelayControlListener->SendResponse()
```

Następnie przypisujemy skrypt do zdarzenia `OnRequest` obiektu `RelayControlListener` i wysyłamy konfigurację do systemu.

Powyższy skrypt pobiera z obiektu `RelayControlListener` wartości parametrów zapytania i zależnie od tego co w nich się znajduje wykonuje odpowiednie akcje. Po czym odsyła do klienta status operacji - w tym przypadku `200, OK`.

Działanie można łatwo przetestować za pomocą zwykłej przeglądarki internetowej wpisując poniższe URL'e (oczywiście adres IP należy zamienić na rzeczywisty adres Waszego Gate'a Http):

`http://192.168.88.4/relaycontrol?cmd=setValue&val=1` - Włącza Relay

`http://192.168.88.4/relaycontrol?cmd=setValue&val=0` - Wyłącza Relay

`http://192.168.88.4/relaycontrol` - Przełącza stan Relay'a

Jak widać na przykładach możemy użyć Listenera na dwa sposoby. Jeśli parametr `cmd` (komenda do wykonania) i `val` (wartość do ustawienia) są odpowiednio zdefiniowane to ustawiają konkretny stan Relay'a. Jeśli pominiemy w URL'u te parametry to obiekt działa jak przełącznik (Switch).

Powyższy przykład może zostać dalej rozbudowany o kolejne komendy, jeśli jest potrzeba wykonania innych komend. Można też dodać kolejne parametry identyfikujące obiekt, na którym te komendy należy wykonać.

4.3. Pobieranie stanu

W poprzednim kroku umożliwiliśmy sterowania obiektem w systemie z zewnątrz. Bardzo często w kolejnym kroku pojawia się potrzeba udostępnienia także możliwości pobrania aktualnego stanu obiektu.

Jedną z szybszych i najbardziej intuicyjnych metod (niekoniecznie najlepszą) jest zdefiniowane kolejnego Listenera który pobierze wartość `Value` z obiektu Relay i odeśle do klienta. Najprostszy skrypt realizujący taką funkcjonalność wyglądać może jak poniżej.

```
-- RelayStateOnRequest()
GateHttp->RelayState->StatusCode = 200
GateHttp->RelayState->ResponseBody = "Relay State: "..CluZ->Relay->Value
GateHttp->RelayState->SendResponse()
```

Wpisując poniższy URL do przeglądarki widać, że dostajemy odpowiedź ze stanem obiektu Relay (w prostej postaci tekstowej, ale to nie format przysłania danych jest tematem tego przykładu).

`http://192.168.88.4/relaystate` - zwraca `Relay State: 0` lub `Relay State: 1` zależnie od stanu obiektu.

Powyższy przykład na pierwszy rzut oka działa dobrze, ale spróbujmy się przyglądnąć bliżej.

4.4. Kolejność zdarzeń

Skonstruowaliśmy właśnie interfejs (API) Http posiadający dwie metody:

- **/relaycontrol** - umożliwia kontrolowanie obiektem Relay
- **/relaystate** - zwraca aktualny stan (wartość) obiektu Relay

Po szybkim przetestowaniu wszystko działa dobrze, ale jak pisaliśmy wyżej, należy jeszcze zastanowić się jak takie metody zostaną użyte. Mianowicie, łatwo sobie wyobrazić, że w zewnętrznym systemie te dwie metody zostaną użyte tuż po sobie: wywołanie akcji przełączenia oraz po otrzymaniu odpowiedzi, odczytanie stanu w celu potwierdzenia, że akcja nastąpiła i synchronizacji statusu.

I tutaj może nastąpić nieoczekiwane działanie sytemu - Relay się załącza, ale zwracany stan jest 0, czyli nieprawidłowy. Przyczyną takiego stanu rzeczy jest fakt, że operacje te są wykonywane asynchronicznie na dwóch różnych urządzeniach. Nie ma gwarancji, że operacja zmiany stanu Relay'a zdąży się wykonać zanim zapytamy o jego stan. Wywołanie akcji zmiany stanu w skrypcie `RelayControlOnRequest()` wywoływane jest asynchronicznie, co oznacza, że skrypt nie czeka aż CluZ wykona zadanie.

Rozważany przypadek jest bardzo prosty i praktycznie zawsze zadziała, ale w przypadku bardziej skomplikowanych operacji (gdy zaangażowane są różne obiekty celu, do wykonania operacji konieczna jest jeszcze wymiana danych, przesłanie cech, itd.) ryzyko, że pobranie statusu nastąpi przed realną zmianą stanu obiektu(ów) jest realny i w złożonych systemach często obserwujemy takie efekty.

4.5. Synchronizacja zdarzeń

Powyższy problem można rozwiązać zmuszając skrypt `RelayControlOnRequest()` aby zaczekał, aż CluZ realnie wywoła na docelowym urządzeniu akcję zmiany stanu. Można to w prosty sposób osiągnąć za pomocą funkcji `clu.await()`. Np. wywołanie:

`CluZ->Relay->Switch(0)`

zamieniamy na:

`clu.await(CluZ->Relay->Switch(0))`

(Pozostałe wywołania akcji na CluZ zamieniamy w analogiczny sposób).

Od teraz nasz Listener nie wyśle potwierdzenia `200, OK` zanim akcja na CluZ nie zostanie realnie wykonana więc klient korzystający z tego interfejsu nie zostanie wprowadzony w błąd poprzez zbyt szybkie potwierdzenie wykonania zadania.

Funkcja `clu.await()` ma jednak pewne ograniczenie. Limit czasowy na wykonanie wywołania jest 800ms i jeśli nie uda się zrealizować zadania w takim czasie, skrypt zakończy się timeout'em i klient Http dostanie w odpowiedzi błąd Http: `500 Internal Server Error`.

W większości przypadków takie timeout nie jest problemem i system będzie działał poprawnie, ale w przypadku złożonych operacji i/lub gdy CluZ będzie obciążone innymi zadaniami może się to wydarzyć. Sposób na rozwiązanie problemu w takim przypadku został opisany w kolejnej sekcji.

4.6. Potwierdzenie zwrotne

W złożonych systemach oraz tam, gdzie zależy nam na dużej niezawodności i stabilności działania integracji należy opóźnić odpowiedź Listenera Http do czasu otrzymania wprost potwierdzenia z CluZ, że zadanie zrealizowano.

W tym celu rozbijamy działanie na dwa etapy. Zamiast jednego skryptu realizującego całość zadania definiujemy dwa: pierwszy realizuje zadanie, drugi wysyła odpowiedź Http po potwierdzeniu przez CluZ, że zakończono wykonywanie zadania.

Dla przejrzystości definiujemy nowy skrypt i przypisujemy go do Listenera:

- Event `OnRequest`: GateHttp->SplitSyncOnRequest()

Skrypt `SplitSyncOnRequest()` wygląda następująco:

```
-- SplitSyncOnRequest()
local data = GateHttp->RelayControlListener->QueryStringParams
if data == nil then
    CluZ->SplitSyncCluzTask("Switch")
    return
else
    if data.cmd == "setValue" then
        local val = tonumber(data.val)
        if(val == 1) then
            CluZ->SplitSyncCluzTask("On")
            return
        elseif(val == 0) then
            CluZ->SplitSyncCluzTask("Off")
            return
        end
    end
end

GateHttp->RelayControlListener->StatusCode = 400
GateHttp->RelayControlListener->ResponseBody = "Bad request"
GateHttp->RelayControlListener->SendResponse()
```

W każdym miejscu skryptu, gdy delegujemy zadnie do CluZ (tym razem poprzez dodatkowy skrypt o czym za chwilę), kończymy działanie naszego skryptu bez wysyłania odpowiedzi Http do klienta. Jeśli wykonanie skryptu dojdzie do końcowych linii będzie to oznaczać, że nie udało się prawidłowo zinterpretować zapytania co oznacza, że jest ono nieprawidłowo i odsyłamy błąd `400, Bad request`. Przy okazji dodaliśmy kolejny poziom zabezpieczenia przed nieprawidłowymi parametrami wywołania.

Zadanie nie jest teraz jak poprzednio wykonywane bezpośrednio na docelowym obiekcie Relay, ale delegowane do skryptu na CluZ o nazwie `SplitSyncCluzTask(action: string)`. Zastosowana notacja oznacza, że skrypt jest wywoływany z parametrem o nazwie `action`, który jest typu `string` - nie mylić z notacją LUA gdzie nie definiujemy typu parametru wywołania funkcji. Parametr `action` definiuje konkretną akcję do wywołania na obiekcie Relay. Działanie jest identyczne jak w poprzednim przypadku.

```
-- SplitSyncCluzTask(action: string)
if(action == "On") then
    CluZ->Relay->SwitchOn(0)
elseif (action == "Off") then
    CluZ->Relay->SwitchOff(0)
elseif (action == "Switch") then
    CluZ->Relay->Switch(0)
else
    -- Unknown action
    GateHttp->SplitSyncRequestCompleted(false)
  -- Return to avoid double completion
  return
end
GateHttp->SplitSyncRequestCompleted(true)
```

Zależnie od zdefiniowanej akcji wykonywana jest odpowiednia metoda na obiekcie Relay. Na końcu informujemy GateHttp, że zakończyliśmy zadanie i należy odesłać do klienta odpowiedź. Została do tego celu stworzona metoda na GateHttp o nazwie `SplitSyncRequestCompleted(success: boolean)`, która jako parametr przyjmuje wartość logiczną: `true` jeśli udało się wykonać akcję, `false` w przeciwnym przypadku.

```
-- SplitSyncRequestCompleted(success: boolean)
if success then
    GateHttp->RelayControlListener->StatusCode = 200
    GateHttp->RelayControlListener->ResponseBody = "OK"
else
    GateHttp->RelayControlListener->StatusCode = 405
    GateHttp->RelayControlListener->ResponseBody = "Not allowed"
end
GateHttp->RelayControlListener->SendResponse()
```

GateHttp poprzez powyższą metodę wysyła odpowiedź do klienta informującą o sukcesie lub błędzie zależnie od otrzymanego parametru. Tym sposobem zrealizowaliśmy w pełni synchroniczną metodę Http, która nie ma ograniczenia czasowego na działanie. W bardziej zaawansowanych przypadkach można jeszcze bardziej usprawnić działanie systemu poprzez wywołanie funkcji `SplitSyncRequestCompleted(success: boolean)` w odpowiedzi na zdarzenia informujące o zmianie wartości konkretnego obiektu. Co daje pewność, że zmiana nastąpiła i jeszcze bardziej zwiększa stabilność systemu.

> **Uwaga!**
> W przypadku danych otrzymywanych z zewnętrznych systemów należy zawsze stosować metodę ograniczonego zaufania co do ich poprawności. Zalecamy nieprzekazywanie bezpośrednio wartości do metod i skryptów wewnątrz systemu a stosowanie konkretnych akcji w zależności od wartości metod jak widać w powyższych skryptach. W przypadku konieczności bezpośredniego użycia zmiennych otrzymanych z zewnątrz należy je przekazywać za pośrednictwem zmiennych użytkownika (które są adresowalne w całym systemie Grenton i można je swobodnie przesyłać

między urządzeniami CLU). Dodatkowo każdą zmienną otrzymaną z zewnątrz należy w skrypcie walidować pod kątem jest poprawności, wartości, zakresu. Brak odpowiedniej weryfikacji otrzymanych wartości może powodować nieoczekiwane działanie systemu, otworzyć dostęp do niechcianych funkcjonalności a nawet powodować błędy oraz przejście CLU w tryb Emergency.

4.7. Timeout

Stworzony Listener działa już prawie niezawodnie. Dlaczego prawie? Zastanówmy się co się stanie, jeśli CluZ z jakiegoś powodu nigdy nie wywoła metody `SplitSyncRequestCompleted(success: boolean)`. GateHttp zostaje wtedy w stanie oczekiwania na zakończenie obsługi bieżącego zapytania i przestaje reagować na kolejne zapytania.

Oczywiście w dobrze skonfigurowanym systemie nie powinno się to zdarzyć. Jednak zawsze może wystąpić niespodziewana sytuacja i dlatego każdy element systemu powinien być skonfigurowany tak aby działał możliwie niezależnie i był odporny na błędy w innych obszarach. Dlatego nasz Listener też powinien być w pełni odporny na takie sytuacje.

W tym celu na GateHttp zdefiniujemy obiekt Timer, który będzie pilnował, żeby oczekiwanie na odpowiedź CluZ nie trwała w nieskończoność. Parametry nowego obiektu:

- **Name:** SplitSyncTimeout
- **Event OnTimer:** GateHttp->SplitSyncTimeoutOnTimer()
- **Time:** 3000 - tutaj należy dobrać czas odpowiednio do konkretnej sytuacji, na potrzeby przykładu przyjmujemy 3s (3000ms)
- **Mode:** CountDown

Skrypt wykonywany po upłynięciu zadanego czasu wygląda następująco:

```
-- SplitSyncTimeoutOnTimer()
GateHttp->RelayControlListener->StatusCode = 408
GateHttp->RelayControlListener->ResponseBody = "Timeout"
GateHttp->RelayControlListener->SendResponse()
```

Działanie skryptu jest dość proste, po prostu zwraca błąd `408, Timeout`.

Żeby wszystko zadziałało należy odpowiednio zmodyfikować skrypty `SplitSyncOnRequest()` oraz `SplitSyncRequestCompleted(success: boolean)`.

```
-- SplitSyncOnRequest()
local data = GateHttp->RelayControlListener->QueryStringParams
if data == nil then
    CluZ->SplitSyncCluzTask("Switch")
    GateHttp->SplitSyncTimeout->Start()
    return
else
    if data.cmd == "setValue" then
        local val = tonumber(data.val)
        if(val == 1) then
            CluZ->SplitSyncCluzTask("On")
            GateHttp->SplitSyncTimeout->Start()
            return
        elseif(val == 0) then
            CluZ->SplitSyncCluzTask("Off")
            GateHttp->SplitSyncTimeout->Start()
            return
        end
```

```
    end
end

GateHttp->RelayControlListener->StatusCode = 400
GateHttp->RelayControlListener->ResponseBody = "Bad request"
GateHttp->RelayControlListener->SendResponse()
```

Za każdym razem, gdy delegujemy zadanie do CluZ startujemy Timer `SplitSyncTimeout`.

```
-- SplitSyncRequestCompleted(success: boolean)
if(GateHttp->SplitSyncTimeout->State == 1) then
    GateHttp->SplitSyncTimeout->Stop()
    if success then
        GateHttp->RelayControlListener->StatusCode = 200
        GateHttp->RelayControlListener->ResponseBody = "OK"
    else
        GateHttp->RelayControlListener->StatusCode = 405
        GateHttp->RelayControlListener->ResponseBody = "Not allowed"
    end
    GateHttp->RelayControlListener->SendResponse()
end
```

Natomiast w skrypcie `SplitSyncRequestCompleted(success: boolean)` sprawdzamy najpierw czy Timer jest ciągle w stanie `1` (włączonym). Zapobiega to przed niepotrzebną próbą wysyłania odpowiedzi w sytuacji, gdy timeout już wystąpił - czas na odpowiedź się skończyć i została wysłana odpowiedź informująca o wystąpieniu błędu `408, Timeout`. Jeśli Timer ciągle działa (normalna sytuacja, czas na odpowiedź się nie wyczerpał) zatrzymujemy Timer i dalej postępujemy tak jak poprzednio.

4.8. Wiele obiektów

Wróćmy jeszcze na chwilę do metody pobierającej stan Relay'a. W szczególności przyjrzyjmy się jeszcze raz następującej linii:

```
GateHttp->RelayState->ResponseBody = "Relay State: "..CluZ->Relay->Value
```

Kluczowe tutaj jest pobranie wartości cechy Value obiektu Relay:

```
CluZ->Relay->Value
```

Ta metoda działa dobrze, ale należy zdawać sobie sprawę, że wartość tej cechy pobierana jest w momencie wykonywania skryptu. W jej wyniku następuje komunikacje między GateHttp a CluZ przez sieć. Jest to wywołanie synchroniczne, czyli metoda czeka aż zostanie dostarczona odpowiedź z wartością cechy `Value` obiekty Relay. Wiemy już o pewnych ograniczeniach takiego wywołania. W tym konkretnym przypadku zagrożeń jest jeszcze więcej. Mianowicie, wartość tej cechy jest pobierana za każdym razem, gdy klient zapyta o jej wartość przez interfejs Http co generuje niepotrzebny ruch w systemie. Dodatkowo wprowadza niepotrzebne opóźnienie w systemie. Jeśli takich zapytań jest dużo to może to mieć wpływ na wydajność systemu. W niektórych zwłaszcza prostych przypadkach jest to akceptowalne i system będzie sobie z tym dobrze radził. Ale nie zawsze.

Wyobraźmy sobie, że w systemie jest wiele obiektów i potrzebujemy w odpowiedzi dostarczyć statusy wszystkich z nich (w postaci JSON lub CSV). Jeśli w takim przypadku zastosujemy identyczną metodę to skrypt realizujący takie zadanie może wyglądać mniej więcej jak poniżej:

```
GateHttp->RelayState->StatusCode = 200

local response = CluZ->Relay01->Value
response = response .. "," .. CluZ->Relay02->Value
response = response .. "," .. CluZ->Relay03->Value
response = response .. "," .. CluZ->Relay04->Value
response = response .. "," .. CluZ->Relay05->Value
response = response .. "," .. CluZ->Relay06->Value
response = response .. "," .. CluZ->Relay07->Value
response = response .. "," .. CluZ->Relay08->Value
response = response .. "," .. CluZ->Relay09->Value
response = response .. "," .. CluZ->Relay10->Value
response = response .. "," .. CluZ->Relay11->Value

GateHttp->RelayState->ResponseBody = "System State: ".. response
GateHttp->RelayState->SendResponse()
```

W rzeczywistym systemie obiektów Relay może być znacznie więcej. Każda linia powoduje odpytanie do CluZ o wartość cechy Value przez sieć. Zebranie stanu wszystkich obiektów może zabrać sporo czasu. Opóźnia to znacznie odpowiedź i na czas wykonywania operacji blokuje GateHttp.

Seria odpytań występuje każdorazowo, gdy klient zapyta o stan systemu. W większości przypadków wartość cechy między zapytaniami zmienia się tylko dla jednego obiektu, tego właśnie zmienionego. Te wszystko powoduje dużo niepotrzebnego ruchu i negatywnie wpływa na szybkość działania systemu. Z punktu widzenia użytkownika końcowego system może w takich przypadkach działać niestabilnie, mieć nieoczekiwane opóźnienia, zawieszać się na krótkie lub dłuższe chwile a nawet gubić niektóre zdarzenia.

4.9. Stan dla złożonego systemu

W celu rozwiązania powyższego problemu należy trochę inaczej podejść do zadania pobierania stanu urządzeń. W dalszej części tej sekcji, dla prostoty przykładów, wrócimy do pojedynczego obiektu Relay, ale podana metoda zadziała praktycznie dla dowolnej liczby obiektów.

Powiedzmy, że zamiast odpytywać zdalne CluZ o stan obiekty Relay za każdym razem, kiedy klient o niego zapyta moglibyśmy trzymać jego wartość lokalnie w zmiennej użytkownika GateHttp. Dzięki temu, kiedy klient zapyta bez żadnych opóźnień, zwracamy jej wartość natychmiast bez żadnych opóźnień, nazwijmy ją `RelayValueOnGateHttp`. Co więcej chcielibyśmy wyeliminować wszystkie zapytania synchronizujące jej wartość i dostawać informacje tylko wtedy, kiedy jest to potrzebne, czyli kiedy wartość cechy `CluZ->Relay->Value` się zmieni. Aby to osiągnąć do zdarzenia `OnValueChange` obiektu Relay przypisujemy następującą komendę:

`GateHttp->RelayValueOnGateHttp=CluZ->Relay->Value`

Co mniej więcej oznacza: Za każdym razem, kiedy wartość cechy Value się zmieni, przypisz do cechy użytkownika `RelayValueOnGateHttp` na GateHttp tą nową wartość. Od teraz po stronie GateHttp zawsze będziemy mieli aktualną wartość obiektu Relay. W momencie zapytania wysyłamy po prostu tą wartość do klienta. Aby to zrealizować modyfikujemy skrypt `RelayStateOnRequest()` w następujący sposób:

```
-- RelayStateOnRequest()
GateHttp->RelayState->StatusCode = 200
GateHttp->RelayState->ResponseBody = "Relay State: "..GateHttp-
>RelayValueOnGateHttp
GateHttp->RelayState->SendResponse()
```

Jak wspomniano wcześniej można zastosować dla dowolnie wielu obiektów i nie powoduje żadnego negatywnego wpływu na wydajność systemu, ponieważ komunikowane są tylko zmiany poszczególnych wartości w momencie, kiedy wystąpią.

4.10. Push Notyfikacje

Idąc krok dalej na drodze do idealnej integracji zaimplementujmy jeszcze jedno usprawnienie. Jak dotąd klient sam musiał dopytywać co chwilę czy przypadkiem coś nie zmieniło się w systemie. Jeśli system ma być responsywny to takie zapytania muszą odbywać się często. Częste zapytania generują niepotrzebny ruch i zwiększają ryzyko opóźnień, zwłaszcza w obsłudze zdarzeń bardzo wrażliwych na opóźnienia jak np.: włączanie oświetlenia, gdzie użytkownik od razu czuje, że akcja nie nastąpiła natychmiast po dotknięciu przycisku.

Dodatkowo klient nie jest notyfikowany natychmiast a zmianie w systemie, ale dopiero w momencie, kiedy sam dopyta czy aby nic się nie zmieniło.

Rozwiązaniem jest metoda Push stanu, gdzie to system sam aktywnie wysyła notyfikację a zmianie stanu urządzenia w systemie. W celu zaimplementowania takiego mechanizmu tworzymy nowy obiekt na GateHttp typu HttpRequest:

- **Name:** StatePushNotification
- **Host: IP:** Port serwera http nasłuchującego informacji o zmianach stanu
- **Path:** /statechanged
- **Method:** PUT

Pozostałe ustawienia bez zmian.

Następnie dodajemy nowy skrypt `SendStatePushNotification(newValue: number)`:

```
-- SendStatePushNotification(newValue: number)
GateHttp->StatePushNotification->SetQueryStringParams("val="..newValue)
GateHttp->StatePushNotification->SendRequest()
```

Aby poinformować klienta o nowym statusie należy wywołać skrypt podając jako parametr nową wartość cechy `Value`. Najlepiej zrobić to w zdarzeniu `OnValueChange` obiektu Relay. Ponieważ przypisaliśmy krok wcześniej wartość do zmiennej użytkownika `GateHttp->RelayValueOnGateHttp`, możemy jej użyć, aby uniknąć niepotrzebnego ponownego przesyłanie tej wartości. Zatem przypisanie wyglądać będzie następująco:

```
GateHttp->SendStatePushNotification(GateHttp->RelayValueOnGateHttp)
```

Należy pamiętać, że kopiowanie wartości do cechy `RelayValueOnGateHttp` musi nastąpić wcześniej.

Od teraz za każdym razem, gdy wartość cechy `Value` obiektu Relay się zmieni automatycznie zostanie wysłana notyfikacja z nową wartością.

Wybrana metoda przesyła nową wartość jako parametr URL, ale można oczywiście sformatować odpowiedź w dowolny sposób i przesłać w ciele wiadomości ustawiając wartość za pomocą metody `SetRequestBody(value)`.

> **Uwaga!**
> Należy pamiętać, że Gate Http otwiera nieograniczone możliwości kooperacji z systemem i można za jego pomocą wykonać dowolną operację, nawet szkodliwą. Dlatego ważne jest, aby konfiguracja Gate Http była starannie przemyślana i wykonana z najwyższą dbałością.

5. Przywracanie ustawień fabrycznych - *Hard Reset*

Uruchomienie funkcji *Hard Reset* na module GATE Http powoduje:

- Usunięcie zapisanej konfiguracji;
- Sformatowanie partycji pamięci flash;
- Usunięcie wszystkich utworzonych obiektów LUA;
- Utratę komunikacji pomiędzy OM / HM a modułem Gate.

W celu przywrócenia ustawień fabrycznych funkcją *Hard Reset* należy wykonać następujące czynności (zgodnie z podaną kolejnością):

- Odłączyć zasilanie od modułu Gate;
- Nacisnąć i przytrzymać przycisk *Reset* na module (przycisk znajduje się pod dolną zaślepką modułu);
- Podłączyć zasilanie do modułu Gate;
- Trzymać wciśnięty przycisk *Reset* przez co najmniej 10 sekund - podczas resetu dioda zielona będzie świecić światłem ciągłym. Prawidłowe wykonanie resetu zostanie potwierdzone 3-krotnym mrugnięciem diody zielonej.
- Po upływie 10 sekund zwolnić przycisk *Reset*
- Odczekać około 60 sekund aż do momentu, gdy na module diody - zielona oraz czerwona - będą mrugać naprzemiennie (tryb *Emergency*)

Po wykonaniu procedury na module zostanie wyczyszczona konfiguracja, natomiast sam moduł przestanie być widoczny (brak odpowiedzi na *Keep-Alive*) w projekcie z poziomu Object Managera. Aby ponownie przywrócić moduł, należy wykonać CLU Discovery a następnie wysłać konfigurację.

6. Parametry konfiguracyjne

> **Uwaga!**
> Opisana funkcjonalność oraz integracja jest dostępna dla **GRENTON GATE HTTP, DIN, Eth (INT-211-E-01)** posiadający **firmware 1.4.2-2346 lub wyższy!**

A. Obiekt GATE

CECHY

Nazwa	Opis
`Uptime`	Czas pracy urządzenia od ostatniego resetu (w sekundach)
`ClientReportInterval`	Okres raportowania o zmianach cech
`Date`	Aktualna data
`Time`	Aktualny czas (hh:mm:ss)
`LocalTime`	Aktualny lokalny znacznik czasu
`TimeZone`	Strefa czasowa
`UnixTime`	Aktualny uniksowy znacznik czasu
`FirmwareVersion`	Wersja oprogramowania Gate
`UseCloud`	Określa czy GATE łączy się do chmury
`CloudConnection`	Określa status połączenia GATE z chmurą
`NTPTimeout`	Czas oczekiwania na odpowiedź z serwera NTP
`UseNTP`	Określa czy GATE używa NTP
`PrimaryDNS`	Preferowany serwer DNS
`SecondaryDNS`	Alternatywny serwer DNS
`TelnetLogLevel`	Określa poziom logowania
`OverloadDetection`	Określa, czy gate powinien zgłaszać przeciążenie procesora używając czerwonej diody
`ResetReason`	Określa przyczynę restartu urządzenia: `0` - włączenie zasilania `2` - przeładowanie konfiguracji `3` - wyjątek systemowy

METODY

Nazwa	Opis
`SetDateTime`	Ustawia datę i czas
`StartConsole`	Uruchamia konsolę Lua
`StartConsoleOnReboot`	Uruchamia konsolę Lua przy kolejnym uruchomieniu modułu
`SetClientReportInterval`	Ustawia okres raportowania o zmianach cech
`SetPrimaryDNS`	Ustawia cechę PrimaryDNS
`SetSecondaryDNS`	Ustawia cechę SecondaryDNS
`SetTelnetLogLevel`	Określa poziom logowania

ZDARZENIA

Nazwa	Opis
`OnInit`	Zdarzenie wywoływane jednorazowo w momencie inicjalizacji urządzenia

B. Obiekt HttpRequest

> **Uwaga!**
> Cechy opisane jako nieustawialne są cechami zawierającymi odpowiedzi. Wartości początkowe tych cech należy pozostawić niezmienione. Wszelkie operacje na tych zmiennych należy wykonywać na skryptach (oraz zmiennych lokalnych).

CECHY

Nazwa	Opis
`Host`	Adres hosta
`Path`	Ścieżka zapytania
`QueryStringParams`	Parametry zapytania. \z oznacza brak
`Method`	Typ metody wysyłanej w zapytaniu np. GET, POST
`Timeout`	Dopuszczalny czas odpowiedzi
`RequestType`	Typ zawartości wysyłanego zapytania. Definiuje parametr *content-type* w nagłówku zapytania. W zależności od wybranego typu zawartość cechy `RequestBody` jest odpowiednio serializowana: `0 - None` - niezdefiniowany. W nagłówku nie jest wysyłane content-type. Zawartość cechy `RequestBody` nie jest serializowana. `1 - Text` - *content-type: text/plain*. Zawartość cechy `RequestBody` nie jest serializowana. `2 - JSON` - *content-type: application/json*. Zawartość cechy `RequestBody` jest serializowana do formatu JSON. `3- XML` - *content-type: text/xml*. Zawartość cechy `RequestBody` jest serializowana do formatu XML. `4 - FormData` - *content-type: application/x-www-form-urlencoded*. Zawartość cechy `RequestBody` jest serializowana do tabeli. `5 - Other` - typ zawartości (*content-type*) jest inny niż wbudowany. Typ można zdefiniować umieszczając go w nagłówku (cecha `RequestHeaders`). Zawartość nie jest serializowana.
`ResponseType`	Typ oczekiwanej odpowiedzi. Definiuje parametr *Accept* w nagłówku zapytania. W zależności od wybranego typu zawartość otrzymanej odpowiedzi (cechy `ResponseBody`) jest odpowiednio parsowana do tabeli: `0 - None` - parametr *Accept* nie jest wysyłany w nagłówku wysyłanego zapytania. Odpowiedź (cecha `ResponseBody`) nie jest parsowana. `1 - Text` - *Accept: text/plain*. Odpowiedź (cecha `ResponseBody`) nie jest parsowana. `2 - JSON` - *Accept: application/json*. Odpowiedź (cecha `ResponseBody`) jest parsowana z JSON. `3 - XML` - *Accept: text/xml*. Odpowiedź (cecha `ResponseBody`) jest parsowana z XML. `4 - FormData` - *Accept: application/x-www-form-urlencoded*. Odpowiedź (cecha ResponseBody) jest parsowana. `5 - Other` - parametr *Accept* nagłówka jest inny niż wbudowany. Parametr można zdefiniować umieszczając go w nagłówku (cecha `RequestHeaders`).
`RequestHeaders`	Dodatkowe nagłówki zapytania HTTP. `\z` oznacza brak zawartości.
`RequestBody`	Zawartość wiadomości wysyłanej w zapytaniu. `\z` oznacza brak zawartości

Nazwa	Opis
`ResponseHeaders`	Nagłówki odpowiedzi HTTP
`ResponseBody`	Zawartość wiadomości otrzymanej po wysłaniu zapytania. (cecha wykorzystywana do odczytu w skryptach - nieustawialna)
`StatusCode`	Status odpowiedzi HTTP
`IsActive`	Stan aktywności zapytania HTTP

METODY

Nazwa	Opis
`SendRequest`	Wysyła zapytanie
`AbortRequest`	Przerywa obsługę zapytania
`Clear`	Usuwa treść zapytania
`SetHost`	Ustawia adres hosta
`SetPath`	Ustawia ścieżkę zapytania
`SetQueryStringParams`	Ustawia parametry zapytania
`SetMethod`	Ustawia metodę zapytania
`SetTimeout`	Ustawia dopuszczalny czas odpowiedzi
`SetRequestType`	Ustawia typ zawartości wysyłanego zapytania (content-type)
`SetResponseType`	Ustawia typ oczekiwanej odpowiedzi na zapytanie
`SetRequestHeaders`	Ustawia dodatkowe nagłówki HTTP w zapytaniu
`SetRequestBody`	Ustawia zawartość wiadomości w zapytaniu

ZDARZENIA

Nazwa	Opis
`OnRequestSent`	Zdarzenie wywoływane w momencie wysłania zapytania
`OnResponse`	Zdarzenie wywoływane w momencie otrzymania odpowiedzi

C. Obiekt HttpListener

> **Uwaga!**
> Cechy opisane jako nieustawialne są cechami zawierającymi odpowiedzi. Wartości początkowe tych cech należy pozostawić niezmienione. Wszelkie operacje na tych zmiennych należy wykonywać na skryptach (oraz zmiennych lokalnych)

CECHY

Nazwa	Opis
`Path`	Ścieżka zapytania
`Method`	Typ metody otrzymanej w zapytaniu np. `GET`, `POST`
`QueryStringParams`	Zwraca parametry zapytania HTTP (cecha wykorzystywana do odczytu w skryptach - nieustawialna)
`RequestType`	Typ otrzymanego zapytania. W zależności od wybranego typu, zawartość otrzymanego zapytania (cechy `RequestBody`) jest odpowiednio parsowana do tabeli: `0 - None` - Odpowiedź nie jest parsowana. `1 - Text` - Odpowiedź nie jest parsowana. `2 - JSON` - Odpowiedź jest parsowana z JSON. `3 - XML` - Odpowiedź jest parsowana z XML. `4 - FormData` - Odpowiedź jest parsowana. `5 - Other` - Odpowiedź nie jest parsowana. Cecha `RequestBody` zwraca treść zapytania HTTP (cecha wykorzystywana do odczytu w skryptach - nieustawialna).
`RequestHeaders`	Zwraca nagłówki zapytania HTTP (cecha wykorzystywana do odczytu w skryptach - nieustawialna)
`RequestBody`	Zwraca treść zapytania HTTP (cecha wykorzystywana do odczytu w skryptach - nieustawialna)
`ResponseType`	Typ zawartości wysłanej odpowiedzi na zapytanie. Definiuje parametr *content-type* w nagłówku odpowiedzi. W zależności od wybranego typu, zawartość cechy `ResponseBody` jest odpowiednio serializowana: `0 - None` - niezdefiniowany. W nagłówku nie jest wysyłane *content-type*. Zawartość nie jest serializowana. `1 - Text` - *content-type: text/plain*. Zawartość nie jest serializowana. `2 - JSON` - *content-type: application/json*. Zawartość `RequestBody` jest serializowana do formatu JSON. `3 - XML` - *content-type: text/xml*. Zawartość `RequestBody` jest serializowana do formatu XML. `4 - FormData` - *content-type: application/x-www-form-urlencoded*. Zawartość `RequestBody` jest serializowana. `5 - Other` - parametr *Accept* nagłówka jest inny niż wbudowany. Parametr można zdefiniować umieszczając go w nagłówku (cecha `RequestHeaders`).
`ResponseHeaders`	Dodatkowe nagłówki odpowiedzi HTTP
`ResponseBody`	Zwraca treść odpowiedzi HTTP (cecha wykorzystywana do odczytu w skryptach).

Nazwa	Opis
`StatusCode`	Status wysyłanej odpowiedzi HTTP. Obsługiwane statusy: `200` - OK `201` - Utworzono `202` - Przyjęto `204` - Brak zawartości `205` - Przywróć zawartość `400` - Nieprawidłowe zapytanie `403` - Zabroniony `404` - Nie znaleziono `405` - Niedozwolona metoda `406` - Niedozwolone `408` - Koniec czasu oczekiwania na żądanie `409` - Konflikt `410` - Zniknął (usunięto)

METODY

Nazwa	Opis
`SendResponse`	Wysyła odpowiedź na zapytanie
`Clear`	Usuwa treść odpowiedzi
`SetPath`	Ustawia ścieżkę zapytania
`SetResponseType`	Ustawia typ oczekiwanej odpowiedzi na zapytanie
`SetResponseHeaders`	Ustawia dodatkowe nagłówki odpowiedzi HTTP
`SetResponseBody`	Ustawia treść odpowiedzi
`SetStatusCode`	Ustawia status odpowiedzi

ZDARZENIA

Nazwa	Opis
`OnRequest`	Zdarzenie wywoływane w momencie otrzymania zapytania

D. Obiekt Timer

CECHY

Nazwa	Opis
`Time`	Zliczany czas (w ms)
`Mode`	Tryb pracy timera: `0` – zliczenie w dół (countdown), `1` – cykliczny (interval)
`State`	Aktualny stan pracy timera: `0` – zatrzymany (stopped), `1` – liczy (counting)

METODY

Nazwa	Opis
`SetTime`	Ustawia czas timera (w ms)
`SetMode`	Ustawia tryb pracy: `0` – zliczenie w dół (countdown), `1` – cykliczny (interval)
`Start`	Uruchamia timer
`Stop`	Zatrzymuje timer

ZDARZENIA

Nazwa	Opis
`OnTimer`	Zdarzenie wywoływane przy zliczeniu timera
`OnStart`	Zdarzenie wywoływane przy uruchomieniu timera
`OnStop`	Zdarzenie wywoływane przy zatrzymaniu timera

E. Obiekt Sonos

CECHY

Nazwa	Opis
`Host`	Adres IP głośnika
`UpdatePeriod`	Okres aktualizacji stanu
`Status`	Stan komunikacji z głośnikiem: `0` - brak połączenia, `1` - połączono
`ErrorCode`	Ostatni kod błędu: `0` - brak błędu, wartości ujemne - ujemny kod odpowiedzi HTTP, wartości dodatnie - kod błędu UPnP
`Volume`	Głośność w zakresie 0 - 100%
`Mute`	Stan wyciszenia: `0` - Wyłączone, `1` - Włączone
`Artist`	Nazwa autora
`Title`	Tytuł utworu
`State`	Stan odtwarzania: `0` - zatrzymane, `1` - odtwarzanie, `2` - pauza, `3` - stan przejściowy
`PlayMode`	Tryb odtwarzania: `0` - normalny, `1` - powtarzaj wszystkie, `2` - powtarzaj jeden, `3` - losowy, bez powtarzania, `4` - losowy, powtarzaj wszystkie, `5` - losowy, powtarzaj jeden
`AlbumArt`	Adres okładki albumu
`Name`	Nazwa głośnika
`CooridnatorName`	Nazwa koordynatora grupy

METODY

Nazwa	Opis
`SetUpdatePeriod`	Ustawia okres aktualizacji stanu
`SetVolume`	Ustawia głośność w zakresie od 0% do 100%
`SetMute`	Ustawia stan wyciszenia
`SetPlayMode`	Ustawia tryb odtwarzania
`Play`	Rozpoczyna odtwarzanie
`Pause`	Wstrzymuje odtwarzanie (pauza)
`Stop`	Zatrzymuje odtwarzanie
`Next`	Przełącza na następną ścieżkę
`Prev`	Przełącza na poprzednią ścieżkę
`VolumeUp`	Zwiększa głośność o wartość określoną w procentach
`VolumeDown`	Zmniejsza głośność o wartość określoną w procentach
`SwitchMute`	Przełącza stan wyciszenia
`SwitchPlay`	Przełącza stan odtwarzania pomiędzy pauzą, a odtwarzaniem
`LeaveGroup`	Usuwa głośnik z grupy, jeżeli w jakiejś się znajduje
`JoinGroup`	Dodaje głośnik do grupy określonej przez nazwę koordynatora

ZDARZENIA

Nazwa	Opis
`OnConnected`	Zdarzenie wywoływane po nawiązaniu połączenia z głośnikiem
`OnDisconnected`	Zdarzenie wywoływane po zerwaniu połączenia z głośnikiem
`OnError`	Zdarzenie wywoływane po wystąpieniu błędu
`OnChange`	Zdarzenie wywoływane po zmianie wartości Mute, Volume, Title, Artist, State, PlayMode, AlbumArt, CoordinatorName
`OnMuteChange`	Zdarzenie wywoływane po zmianie wartości Mute
`OnVolumeChange`	Zdarzenie wywoływane po zmianie wartości Volume
`OnTitleChange`	Zdarzenie wywoływane po zmianie wartości Title
`OnArtistChange`	Zdarzenie wywoływane po zmianie wartości Artist
`OnStateChange`	Zdarzenie wywoływane po zmianie wartości State
`OnPlayModeChange`	Zdarzenie wywoływane po zmianie wartości PlayMode
`OnAlbumArtChange`	Zdarzenie wywoływane po zmianie wartości AlbumArt
`OnCoordinatorNameChange`	Zdarzenie wywoływane po zmianie wartości CoordinatorName

F. Obiekt MusicCast

CECHY

Nazwa	Opis
`Host`	Adres IP głośnika
`UpdatePeriod`	Okres aktualizacji stanu
`Status`	Stan komunikacji z głośnikiem: `0` - brak połączenia, `1` - połączono
`ErrorCode`	Ostatni kod błędu: `0` - brak błędu, wartości ujemne - ujemny kod odpowiedzi HTTP, wartości dodatnie - kod błędu Yamaha Extended Control
`Volume`	Głośność w zakresie 0 - 100%
`Mute`	Stan wyciszenia: `0` - wyłączone, `1` - włączone
`Artist`	Nazwa autora
`Title`	Tytuł utworu
`State`	Stan odtwarzania: `1` - odtwarzanie, `2` - zatrzymane, `3` - pauza
`Shuffle`	Tryb odtwarzania losowego: `1` - wyłączone, `2` - włączone, `3` - utwory, `4` - albumy
`Repeat`	Tryb powtarzania: `1` - wyłączone, `2` - jeden utwór, `3` - wszystkie utwory
`Power`	Stan zasilania: `0` - uśpienie, `1` - włączone
`AlbumArt`	Adres okładki albumu
`ObjectID`	ID obiektu
`ServerID`	ID obiektu serwera grupy
`Name`	Nazwa głośnika

Nazwa	Opis
`Role`	Rola głośnika w grupie: `1` - nie jest częścią grupy, `2` - klient, `3` - serwer
`Input`	Źródło odtwarzania
`AutoPowerStandby`	Stan automatycznego uśpienia: `0` - wyłączone, `1` - włączone

METODY

Nazwa	Opis
`SetUpdatePeriod`	Ustawia okres aktualizacji stanu
`SetVolume`	Ustawia głośność w zakresie od 0% do 100%
`SetMute`	Ustawia stan wyciszenia
`SetShuffle`	Ustawia tryb losowego odtwarzania
`SetRepeat`	Ustawia tryb powtarzania
`SetPower`	Ustawia stan zasilania
`SetAutoPowerStandby`	Ustawia stan AutoPowerStandby
`Play`	Rozpoczyna odtwarzanie
`Pause`	Wstrzymuje odtwarzanie (pauza)
`Stop`	Zatrzymuje odtwarzanie
`Next`	Przełącza na następną ścieżkę
`Prev`	Przełącza na poprzednią ścieżkę
`VolumeUp`	Zwiększa głośność o wartość określoną w procentach
`VolumeDown`	Zmniejsza głośność o wartość określoną w procentach
`SwitchMute`	Przełącza stan wyciszenia
`SwitchPlay`	Przełącza stan odtwarzania pomiędzy pauzą, a odtwarzaniem
`DestroyGroup`	Rozbija bieżącą grupę głośników
`JoinGroup`	Dodaje głośnik do grupy określonej przez ServerID
`LeaveGroup`	Usuwa głośnik z bieżącej grupy
`SetInput`	Ustawia źródło odtwarzania

ZDARZENIA

Nazwa	Opis
`OnConnected`	Zdarzenie wywoływane po nawiązaniu połączenia z głośnikiem
`OnDisconnected`	Zdarzenie wywoływane po zerwaniu połączenia z głośnikiem
`OnError`	Zdarzenie wywoływane po wystąpieniu błędu
`OnChange`	Zdarzenie wywoływane po zmianie wartości Volume, Mute, Artist, Title, State, Shuffle, Repeat, Power, AlbumArt, Input, AutoPowerStandby, ServerID, Role
`OnMuteChange`	Zdarzenie wywoływane po zmianie wartości Mute
`OnVolumeChange`	Zdarzenie wywoływane po zmianie wartości Volume
`OnTitleChange`	Zdarzenie wywoływane po zmianie wartości Title
`OnArtistChange`	Zdarzenie wywoływane po zmianie wartości Artist
`OnStateChange`	Zdarzenie wywoływane po zmianie wartości State
`OnShuffleChange`	Zdarzenie wywoływane po zmianie wartości Shuffle
`OnRepeatChange`	Zdarzenie wywoływane po zmianie wartości Repeat
`OnPowerChange`	Zdarzenie wywoływane po zmianie wartości Power
`OnAlbumArtChange`	Zdarzenie wywoływane po zmianie wartości AlbumArt
`OnInputChange`	Zdarzenie wywoływane po zmianie wartości Input
`OnAutoPowerStandbyChange`	Zdarzenie wywoływane po zmianie wartości AutoPowerStandby
`OnGroupChange`	Zdarzenie wywoływane po wystąpieniu zmiany w obrębie grupy (ServerID, Role)

G. Obiekt CoolMasterNet

CECHY

Nazwa	Opis
`SN`	Numer seryjny jednostki CoolMasterNet
`Host`	Adres jednostki CoolMasterNet w formie http://host:port
`UpdatePeriod`	Okres aktualizacji stanu
`Status`	Stan połączenia: `0` - brak połączenia, `1` - połączono
`ErrorCode`	Ostatni kod błędu CoolMasterNet: `0` - brak błędu, `1` - błąd nawiązywania połączenia TCP, lub kod błędu HTTP

METODY

Nazwa	Opis
`SetUpdatePeriod`	Ustawia okres aktualizacji stanu
`TurnAllOn`	Włącza wszystkie klimatyzatory
`TurnAllOff`	Wyłącza wszystkie klimatyzatory

ZDARZENIA

Nazwa	Opis
`OnConnected`	Zdarzenie wywoływane po nawiązaniu połączenia z jednostką
`OnDisconnected`	Zdarzenie wywoływane po zerwaniu połączenia z jednostką
`OnError`	Zdarzenie wywoływane po wystąpieniu błędu

H. Obiekt CoolMaster

CECHY

Nazwa	Opis
`CoolMasterNetID`	ID obiektu CoolMasterNet
`UIDs`	Jeden lub więcej identyfikatorów klimatyzatorów oddzielonych spacją
`SupportedModes`	Lista wspieranych trybów pracy oddzielonych przecinkiem
`SupportedFanSpeeds`	Lista wspieranych prędkości wentylatora oddzielonych przecinkiem, wpisanie "-" oznacza brak wsparcia
`SupportedLouverPositions`	Lista wspieranych pozycji żaluzji regulującej przepływ powietrza oddzielonych przecinkiem, wpisanie "-" oznacza brak wsparcia
`Status`	Stan połączenia: `0` - brak połączenia, `1` - połączono
`State`	Stan pracy: `1` - aktywny, `0` - zatrzymany, `-` - brak synchronizacji
`Mode`	Tryb pracy: `1` - chłodzenie, `2` - ogrzewanie, `3` - wentylator, `4` - suszenie, `5` - automatyczny, `-` - brak synchronizacji stanu
`TargetTemp`	Zadana temperatura `-` - brak synchronizacji stanu
`FanSpeed`	Prędkość wentylatora: 0-5, `5` - auto, `-` - brak synchronizacji stanu
`LouverPosition`	Pozycja żaluzji regulującej przepływ powietrza: `0` - brak wsparcia, `1` - automatyczna, `2` - horyzontalna, `3` - 30°, `4` - 45°, `5` - 60°, `6` - wertykalna, `7` - zatrzymana, `-` - brak synchronizacji stanu
`AmbientTemp`	Temperatura otoczenia lub wartość średnia temperatury w przypadku grupy urządzeń

Nazwa	Opis
`FailureCode`	Kod błędu

METODY

Nazwa	Opis
`SetSupportedModes`	Ustawia listę wspieranych trybów pracy
`SetSupportedFanSpeeds`	Ustawia listę wspieranych prędkości wentylatora
`SetSupportedLouverPositions`	Ustawia listę wspieranych pozycji szczeliny wentylacyjnej
`SetState`	Ustawia stan pracy
`SetMode`	Ustawia tryb pracy
`SetTargetTemp`	Ustawia wartość zadanej temperatury
`SetFanSpeed`	Ustawia zadaną prędkość wentylatora
`SetLouverPosition`	Ustawia pozycję żaluzji regulującej przepływ powietrza
`TurnOn`	Włącza klimatyzator lub grupę klimatyzatorów
`TurnOff`	Wyłącza klimatyzator lub grupę klimatyzatorów
`SwitchMode`	Przełącza tryb pracy na kolejny

ZDARZENIA

Nazwa	Opis
`OnConnected`	Zdarzenie wywoływane po nawiązaniu połączenia z jednostką
`OnDisconnected`	Zdarzenie wywoływane po zerwaniu połączenia z jednostką
`OnChange`	Zdarzenie wywoływane po zmianie wartości State, Mode, TargetTemp, FanSpeed, LouverPosititon
`OnModeChange`	Zdarzenie wywoływane po zmianie wartości Mode
`OnTargetTempChange`	Zdarzenie wywoływane po zmianie wartości TargetTemp
`OnFanSpeedChange`	Zdarzenie wywoływane po zmianie wartości FanSpeed
`OnLouverPositionChange`	Zdarzenie wywoływane po zmianie wartości LouverPosition
`OnTurnOn`	Zdarzenie wywoływane po włączeniu klimatyzatora lub grupy klimatyzatorów
`OnTurnOff`	Zdarzenie wywoływane po wyłączeniu klimatyzatora lub grupy klimatyzatorów
`OnFailure`	Zdarzenie wywoływane po wystąpieniu błędu
`OnDesynchronization`	Zdarzenie wywoływane po desynchronizacji cech klimatyzatorów należących do grupy

G. Obiekt HEOS

CECHY

Nazwa	Opis
`Host`	Adres IP HEOS
`UserName`	Nazwa użytkownika
`Password`	Hasło
`Status`	Stan komunikacji z głośnikiem: `0` - brak połączenia, `1` - połączono
`ErrorCode`	Ostatni kod błędu HEOS CLI
`Volume`	Głośność w zakresie 0 - 100%
`Mute`	Stan wyciszenia: `0` - wyłączone, `1` - włączone
`Artist`	Nazwa autora
`Title`	Tytuł utworu
`PlayerState`	Stan odtwarzania: `0` - zatrzymane `1` - pauza, `2` - odtwarzanie
`Shuffle`	Tryb odtwarzania losowego: `0` - wyłączone, `1` - włączone
`Repeat`	Tryb powtarzania: `0` - wyłączone, `1` - jeden utwór, `2` - wszystkie utwory
`AlbumArt`	Adres okładki albumu
`ObjectID`	ID obiektu
`GroupID`	ID obiektu lidera grupy
`Name`	Nazwa głośnika
`SourceName`	Źródło odtwarzania

METODY

Nazwa	Opis
`SetVolume`	Ustawia głośność w zakresie od 0% do 100%
`SetMute`	Ustawia stan wyciszenia
`SetShuffle`	Ustawia tryb losowego odtwarzania
`SetRepeat`	Ustawia tryb powtarzania
`Play`	Rozpoczyna odtwarzanie
`Pause`	Wstrzymuje odtwarzanie (pauza)
`Stop`	Zatrzymuje odtwarzanie
`Next`	Przełącza na następną ścieżkę
`Prev`	Przełącza na poprzednią ścieżkę
`VolumeUp`	Zwiększa głośność o wartość określoną w procentach
`VolumeDown`	Zmniejsza głośność o wartość określoną w procentach
`SwitchMute`	Przełącza stan wyciszenia
`SwitchPlay`	Przełącza stan odtwarzania pomiędzy pauzą, a odtwarzaniem
`AddToGroup`	Dodaje głośnik określony przez ObjectID do grupy bieżącego głośnika
`DestroyGroup`	Rozbija bieżącą grupę głośników
`PlayPresetStation`	Odtwarza stację/utwór określony na liście ulubionych w aplikacji HEOS
`PlayInputSource`	Ustawia fizyczne źródło odtwarzania o podanej nazwie, zgodnie z dokumentacją HEOS, np. `inputs/aux1`
`PlayUrl`	Odtwarza strumień wskazany przy pomocy adresu url
`PlayUSB`	Odtwarza plik audio z nośnika USB za pomocą pełnej ścieżki pliku wraz z rozszerzeniem np. `komunikaty/wyciek.mp3`
`PlayUSBClip`	Odtwarza plik audio z nośnika USB za pomocą pełnej ścieżki pliku wraz z rozszerzeniem np. `komunikaty/wyciek.mp3`, w przypadku wywołania pliku audio podczas odtwarzania kolejki/stacji opcjonalne przywraca poprzednie odtwarzanie

ZDARZENIA

Nazwa	Opis
`OnConnected`	Zdarzenie wywoływane po nawiązaniu połączenia z głośnikiem
`OnDisconnected`	Zdarzenie wywoływane po zerwaniu połączenia z głośnikiem
`OnError`	Zdarzenie wywoływane po wystąpieniu błędu
`OnChange`	Zdarzenie wywoływane po zmianie wartości Mute, Volume, Title, Artist, PlayerState, Shuffle, Repeat, AlbumArt, SourceName lub GroupID
`OnMuteChange`	Zdarzenie wywoływane po zmianie wartości Mute
`OnVolumeChange`	Zdarzenie wywoływane po zmianie wartości Volume
`OnTitleChange`	Zdarzenie wywoływane po zmianie wartości Title
`OnArtistChange`	Zdarzenie wywoływane po zmianie wartości Artist
`OnPlayerStateChange`	Zdarzenie wywoływane po zmianie wartości PlayerState
`OnShuffleChange`	Zdarzenie wywoływane po zmianie wartości Shuffle
`OnRepeatChange`	Zdarzenie wywoływane po zmianie wartości Repeat
`OnAlbumArtChange`	Zdarzenie wywoływane po zmianie wartości AlbumArt
`OnSourceChange`	Zdarzenie wywoływane po zmianie wartości SourceName
`OnGroupChange`	Zdarzenie wywoływane po zmianie wartości GroupID
`OnPlaybackError`	Zdarzenie wywoływane po wystąpieniu błędu odtwarzania
`OnClipEnd`	Zdarzenie wywoływane po zakończeniu odtwarzania rozpoczętego przy pomocy PlayUSBClip

H. Obiekt DenonMarantzAVR

CECHY

Nazwa	Opis
`Host`	Adres IP amplitunera AV
`Zone`	Strefa amplitunera AV
`Status`	Stan komunikacji z głośnikiem: `0` - brak połączenia, `1` - połączono
`SystemPower`	Stan zasilania systemu: `0` - uśpienie `1` - włączone
`ZonePower`	Stan zasilania strefy: `0` - wyłączone `1` - włączone
`Volume`	Głośność w zakresie od 0% do 98%
`Mute`	Stan wyciszenia: `0` - wyłączone, `1` - włączone
`Input`	Źródło sygnału
`SurroundMode`	Tryb dźwięku przestrzennego
`SpeakerPreset`	Preset głośników

METODY

Nazwa	Opis
`SetSystemPower`	Ustawia stan zasilania systemu
`SetZonePower`	Ustawia stan zasilenia strefy
`SetVolume`	Ustawia głośność w zakresie od 0% do 98%
`SetMute`	Ustawia stan wyciszenia
`SetInput`	Ustawia źródło sygnału
`SetSourroundMode`	Ustawia tryb dźwięku przestrzennego
`SetSpeakerPreset`	Ustawia wybrany preset głośników
`VolumeUp`	Zwiększa głośność o wartość wyrażoną w procentach
`VolumeDown`	Zmniejsza głośność o wartość wyrażoną w procentach
`SwitchMute`	Ustawia stan wyciszenia
`QuickSelecet`	Wybiera ustawienia Quick Select

ZDARZENIA

Nazwa	Opis
`OnConnected`	Zdarzenie wywoływane po nawiązaniu połączenia z głośnikiem
`OnDisconnected`	Zdarzenie wywoływane po zerwaniu połączenia z głośnikiem
`OnChange`	Zdarzenie wywoływane po zmianie wartości Volume, Mute, Artist, Title, State, Shuffle, Repeat, Power, AlbumArt, Input, AutoPowerStandby, ServerID, Role
`OnMuteChange`	Zdarzenie wywoływane po zmianie wartości Mute
`OnVolumeChange`	Zdarzenie wywoływane po zmianie wartości Volume
`OnSystemPowerChange`	Zdarzenie wywoływane po zmianie wartości SystemPower
`OnZonePowerChange`	Zdarzenie wywoływane po zmianie wartości ZonePower
`OnInputChnage`	Zdarzenie wywoływane po zmianie wartości Input
`OnSurroundModeChange`	Zdarzenie wywoływane po zmianie wartości SurroundMode
`OnSpeakerPresetChange`	Zdarzenie wywoływane po zmianie wartości SpeakerPreset