

Auri Third Party Control API



Contents

Introduction	2
Overview	2
Key Information.....	2
Command Syntax	3
Command Responses	3
Errors	4
List of Supported Commands – TX2N Transmitter	6
GET Commands	6
SET Commands.....	6
List of Supported Commands – D4 and D16 Docking Stations	7
GET Commands	7
SET Commands.....	7
Testing a Connection.....	8
Checking Responses.....	8
Node.js Example	8
Q-Sys Lua Example.....	11

Introduction

This documentation describes the API for Ampetronic and Listen Technologies Auri products, including an overview of the interface and syntax, list of commands and examples of usage.

Where available for the required control system, we recommend the use of provided plugins.

The document assumes that you are familiar with using APIs and have the knowledge to integrate sending and receiving commands into your control system of choice.

Simple set and forget commands can generally be implemented very simply, parsing responses may require more experience in scripting or development for the intended platform.

Overview

Support for the API was introduced in firmware version 1.5, alongside Auri Manager version 1.4 being required to enable the API. If you are running older versions, please update before attempting to use the API.

The API is disabled by default and must be enabled via the Settings page of each device in Auri Manager. See the system manual for more details on how to configure your device.

The API uses plaintext UDP commands with no authentication to read and write data, so if enabling the API ensure that the security of the network is considered suitable to avoid misuse.

A limited subset of commands are supported, primarily focused on monitoring device status and managing broadcast security.

Each command sent will receive a response over the same UDP interface.

Key Information

You will need to know the IP address of the device before connecting, this can be found through the Devices page on Auri Manager.

It is recommended to use a fixed IP to avoid the API connection being interrupted, either by assigning a static IP address or reserving the address in the DHCP server.

You must ensure that UDP traffic over port 54666 is allowed between the Auri product and the device intended to control it.

Usage	Type	Address	Port
API	UDP (Unicast)	Device IP (DHCP, Static or Link Local)	54666

Command Syntax

Commands must be terminated by the “\r” carriage return (ASCII 13).

All commands intended to read data start with GET.

All commands intended to write data start with SET.

Commands have hierarchical identifiers depending on the property in question, for example a system command will simply be –

```
GET SYSTEM STATUS\r
```

However, a radio command must specify the radio number, eg.

```
GET RADIO 1 ENCRYPTION\r
```

And going further an input command must specify the stream and input numbers, eg.

```
GET AUDIO STREAM 1 INPUT 1 MUTE\r
```

SET commands use the same syntax, with a value specified after an “=” sign, the spacing is important and must be followed exactly to avoid errors.

For Boolean fields the values are specified as ON or OFF, eg.

```
SET RADIO 1 ENCRYPTION = ON\r
```

For string fields the values must be encapsulated in quote marks, eg.

```
SET RADIO 1 ENCRYPTION BROADCAST_NAME = "Meeting Room"\r
```

Command Responses

All commands will receive a response over the same UDP interface.

Both GET and SET commands will respond in the same way, confirming the current value of the parameter, eg.

Command	>> GET RADIO 1 ENCRYPTION PRIVACY_KEY\r
Response	>> RADIO 1 ENCRYPTION PRIVACY_KEY = "Password"\r
Command	>> SET RADIO 1 ENCRYPTION PRIVACY_KEY = "New Password"\r
Response	>> RADIO 1 ENCRYPTION PRIVACY_KEY = "New Password"\r

Errors

If the API encounters an error then it will return this in place of the usual response.

The below table shows a list of potential error responses, the description and an example. For each example the following approach is used –

> INVALID COMMAND
> ERROR RESPONSE
> CORRECT COMMAND
> NORMAL RESPONSE

Error Code	Description	Example
201	No GET or SET specified	> SYSTEM STATUS\r > ERROR 201 <SYSTEM STATUS> > GET SYSTEM STATUS\r > SYSTEM STATUS = 0
202	No module identifier specified	> GET STATUS\r > ERROR 202 <STATUS> > GET SYSTEM STATUS\r > SYSTEM STATUS = 0
203	No parameter identifier specified	> GET SYSTEM\r > ERROR 203 <SYSTEM> > GET SYSTEM STATUS\r > SYSTEM STATUS = 0
204	Parameter setting not specified	> SET SYSTEM IDENTIFY\r > ERROR 204 <SYSTEM IDENTIFY> > SET SYSTEM IDENTIFY = ON\r > SYSTEM IDENTIFY = ON
205	Unable to process message	> GET SYSTEM STATUS > ERROR 205 <SYSTEM STATUS> > GET SYSTEM STATUS\r > SYSTEM STATUS = 0
8	Unknown identifier specified	> GET RADIO 3 ENCRYPTION\r > ERROR 8 <RADIO 3 ENCRYPTION> > GET RADIO 2 ENCRYPTION\r > RADIO 2 ENCRYPTION = ON

17	Parameter setting too long	> SET RADIO 1 ENCRYPTION PRIVACY_KEY = "12345678901234567" > ERROR 17 <RADIO 1 ENCRYPTION PRIVACY_KEY> > SET RADIO 1 ENCRYPTION PRIVACY_KEY = "1234567890123456" > RADIO 1 ENCRYPTION PRIVACY_KEY = "1234567890123456"
37	Parameter setting too short	> SET RADIO 1 ENCRYPTION PRIVACY_KEY = "123" > ERROR 37 <RADIO 1 ENCRYPTION PRIVACY_KEY> > SET RADIO 1 ENCRYPTION PRIVACY_KEY = "1234" > RADIO 1 ENCRYPTION PRIVACY_KEY = "1234"

List of Supported Commands – TX2N Transmitter

The following commands are supported by the Auri TX2N and Auri TX2N-D Transmitters.

GET Commands

Command	Responses
GET SYSTEM STATUS	0 (OK) / Non-Zero (Fault)
GET RADIO [1/2] ENCRYPTION	ON/OFF
GET RADIO [1/2] ENCRYPTION BROADCAST_NAME	String
GET RADIO [1/2] ENCRYPTION PRIVACY_KEY	String
GET RADIO [1/2] TRANSMITTER OUTPUT	ON/OFF
GET AUDIO STREAM [1/2] PROGRAM INFO	String
GET AUDIO STREAM [1/2] INPUT [1/2] MUTE	ON/OFF
GET AUDIO STREAM [1/2] OUTPUT LEVEL LEFT	Integer -96 to 0
GET AUDIO STREAM [1/2] OUTPUT LEVEL RIGHT	Integer -96 to 0

SET Commands

Command	Supported Values
SET SYSTEM IDENTIFY	ON
SET RADIO [1/2] ENCRYPTION	ON/OFF
SET RADIO [1/2] ENCRYPTION BROADCAST_NAME	String, 4-32 characters, extended ASCII
SET RADIO [1/2] ENCRYPTION PRIVACY_KEY	String, blank, or 4-16 characters, extended ASCII
SET RADIO [1/2] TRANSMITTER OUTPUT	ON/OFF
SET AUDIO STREAM [1/2] PROGRAM INFO	String, 4-32 characters, extended ASCII
SET AUDIO STREAM [1/2] INPUT [1/2] MUTE	ON/OFF

List of Supported Commands – D4 and D16 Docking Stations

The following commands are supported by the Auri D4 and Auri D16 Docking Stations

GET Commands

Command	Responses
GET SYSTEM STATUS	0 (OK) / Non-Zero (Fault)
GET DOCK ENCRYPTION BROADCAST_NAME [1-32]	String
GET DOCK ENCRYPTION PRIVACY_KEY [1-32]	String

SET Commands

Command	Supported Values
SET SYSTEM IDENTIFY	ON
SET DOCK ENCRYPTION BROADCAST_NAME [1-32]	String, 4-32 characters, extended ASCII
SET DOCK ENCRYPTION PRIVACY_KEY [1-32]	String, blank, or 4-16 characters, extended ASCII

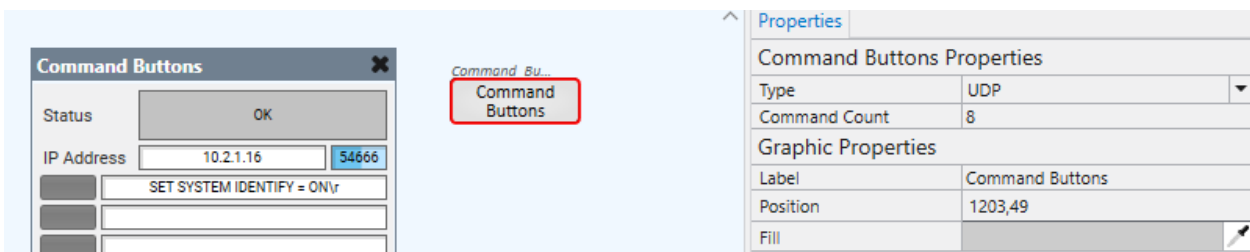
Testing a Connection

One of the simplest ways to get started is by sending the Identify command (SET SYSTEM IDENTIFY = ON) and confirming that the LED on the device starts flashing for 30 seconds.

You can test this in PowerShell using the following command, replacing 10.2.1.16 with your device IP address –

```
$cmd = "SET SYSTEM IDENTIFY = ON"; $msg = $cmd + "`r"; $u = New-Object System.Net.Sockets.UdpClient; $u.Send([Text.Encoding]::ASCII.GetBytes($msg), $msg.Length, "10.2.1.16", 54666); $u.Close()
```

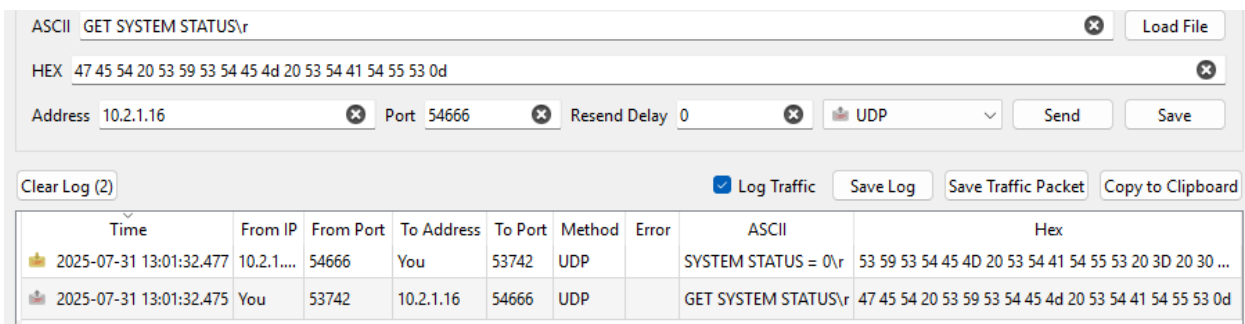
The same test can be carried out in Q-Sys using the basic Command Buttons component –



Checking Responses

In order to check a response, you will need a script or application which can maintain a UDP socket and listen for packets being sent back from the Auri device.

Packet Sender is a simple, free tool that allows sending and receiving of UDP messages, for example –



Node.js Example

An example is also provided below for a node.js implementation, which simply returns the device status to the command line.

```
const dgram = require('dgram');

// Configuration
```

```
const deviceIP = '10.2.1.16';
const devicePort = 54666;
const command = 'GET SYSTEM STATUS';
const timeout = 5000; // 5 seconds

// Create UDP client
const udpClient = dgram.createSocket('udp4');
let isComplete = false;

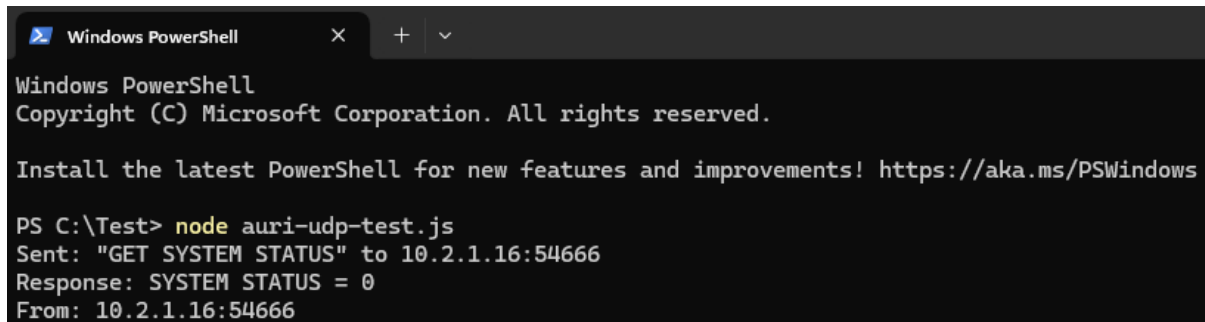
function cleanup() {
  if (!isComplete) {
    isComplete = true;
    udpClient.close();
  }
}

// Handle incoming responses
udpClient.on('message', (response, sender) => {
  console.log(`Response: ${response.toString()}`);
  console.log(`From: ${sender.address}:${sender.port}`);
  cleanup();
});

// Handle connection errors
udpClient.on('error', (error) => {
  console.error(`UDP Error: ${error.message}`);
  cleanup();
});

// Send command to device
const message = command + '\r';
udpClient.send(message, devicePort, deviceIP, (error) => {
  if (error) {
    console.error(`Send failed: ${error.message}`);
    cleanup();
    return;
  }
  console.log(`Sent: "${command}" to ${deviceIP}:${devicePort}`);
});
```

```
// Set timeout
setTimeout(() => {
  if (!isComplete) {
    console.log('No response received within timeout period');
    cleanup();
  }
}, timeout);
```



Windows PowerShell

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! <https://aka.ms/PSWindows>

PS C:\Test> node auri-udp-test.js
Sent: "GET SYSTEM STATUS" to 10.2.1.16:54666
Response: SYSTEM STATUS = 0
From: 10.2.1.16:54666

Q-Sys Lua Example

As an alternative example, the following lua script can be used in Q-Sys with the Text Controller component to execute the same function, returning the device status to the debug output.

```
local udp = UdpSocket.New()
local timeoutTimer = Timer.New()
local done = false

-- Configuration
local deviceIP = "10.2.1.16"
local devicePort = 54666
local command = "GET SYSTEM STATUS"
local timeout = 5

-- Handle incoming responses
udp.Data = function(socket, packet)
    print("Response: " .. packet.Data)
    print("From: " .. packet.Address .. ":" .. packet.Port)
    udp:Close()
    timeoutTimer:Stop()
    done = true
end

-- Set timeout
timeoutTimer.EventHandler = function()
    if not done then
        print("Timeout - no response")
        udp:Close()
    end
end

-- Send command to device
udp:Open()
udp:Send(deviceIP, devicePort, command .. "\r")
print("Sent: " .. command)
timeoutTimer:Start(timeout)
```

Page 1 Text Controller X

```
1 local udp = UdpSocket.New()
2 local timeoutTimer = Timer.New()
3 local done = false
4
5 -- Configuration
6 local deviceIP = "10.2.1.16"
7 local devicePort = 54666
8 local command = "GET SYSTEM STATUS"
9 local timeout = 5
10
11 -- UDP response handler
12 udp.Data = function(socket, packet)
13     print("Response: " .. packet.Data)
14     print("From: " .. packet.Address .. ":" .. packet.Port)
15     udp:Close()
16     timeoutTimer:Stop()
17     done = true
18 end
19
20 -- Timeout handler
21 timeoutTimer.EventHandler = function()
22     if not done then
23         print("Timeout - no response")
24         udp:Close()
25     end
26 end
27
28 -- Send command
29 udp:Open()
30 udp:Send(deviceIP, devicePort, command .. "\r")
31 print("Sent: " .. command)
32 timeoutTimer:Start(timeout)
```

<

Debug Output X [Icons] Normal v [AutoSave Icon]

2025-07-31T13:00:19.367	Starting Script
2025-07-31T13:00:19.367	Sent: GET SYSTEM STATUS
2025-07-31T13:00:19.368	Response: SYSTEM STATUS = 0
2025-07-31T13:00:19.368	From: 10.2.1.16:54666