NPxx

NP10, NP5, NP2

CONTROL SPECIFICATIONS

Communication Protocol

TCP/IP and RS485

Document Revision 2.0 (22nd Feb 2021)

COMMUNICATION SYSTEMS

Revision History

Revision	Date	Software Version	Description of Change
2.0	22nd Feb 2021	4.0.0	Improved "Play audio track" and "Play playlist". Added "SUSPENSION", "DATETIME", "INFOPLAY", "Virtual keypad" And "Delete File"
1.1	10th Dec 2017	3.1.1	User account , TCP-IP communication port, Request-Response protocol, RS485
1.0	09th Nov 2017	3.0.2	Initial release

Introduction

This document describes how to use the communication protocol to control the NPxx audio device remotely.
The protocol is compliant with JSON notation, and any command sent to the NPxx have a response from it.
To know the JSON notation see this website: www.json.org.

Protocol Specifications

Request:
{"req": { <CmdType> :{ <Cmd> }}}

Response:
{"res": { <CmdType> :{ <CmdResponse> }}}

The **packet** is made up to 4 parts

request:

1	2	3	4
Static Header **{"req":**	Command type **{ <CmdType> :**	JSON object **{ <Cmd> }**	Termination brackets **}}**

response:

1	2	3	4
Static Header **{"res":**	Command type **{ <CmdType> :**	JSON object **{ <CmdResponse> }**	Termination brackets **}}**

An object is an unordered set of name/value pairs. An object begins with { (left brace) and ends with} (right brace). Each name is followed by :(colon) and the name/value pairs are separated by, (comma).

{ parameter-name : parameter-value } or
{ parameter-name1 : parameter-value1, parameter-name2 : parameter-value2, … }

Parameter-name is a string in double quotes Parameter-value can be a string in double quotes, or a number.

A string is a sequence of zero or more Unicode characters, wrapped in double quotes, without using backslash escapes.

Examples:
“myname”, “#£??” or “234abc”

A number is represented as an integer.
Examples:
1, -25 or 2345

Example:
Set the volume at value -30dB. The command to send is {"req":{"set":{"AUVOL":-30}}}

Example:
Set the volume at value -30dB and Line In Mix to OFF.
The command to send is {"req":{"set":{"AUVOL":-30, "AUADMIXON": 0}}}

In the response packet the members include the response of any name/value pair, and could be present an error indication to specify a problem occurs performing the operation requested.
When the parameter-name was changed or the parameter-name was execute with the indicated value successfully, the response will be the same content as the original command sent or the follow: “parameter-name”:”done”

Otherwise, if an error happen when the parameter- name was changed or the parameter-name was execute with the indicated value, the response will have the “Error” label, in the field packet where the error occurs.

Example
Set the volume at value -30dB.
The command to send is {"req":{"set":{"AUVOL":-30}}}
The response from the NPxx is {"res":{"set":{"AUVOL":-30}}}

Example:
Set the volume at value -30dB and Line In Mix to OFF.
The command to send is {"req":{"set":{"AUVOL":-30, "AUADMIXON": 0}}}
The response from the NPxx is {"res":{"set":{"AUVOL":-30, "AUADMIXON": 0}}}

Example - wrong command type
Write “fet” in place of “set”
The command to send is {"req":{"fet":{"AUVOL":-30}}}
The response from the NPxx is {"res":{"Error":"No operation in req"}}

Example - wrong command name
Write “AUDIOVOL” in place of “AUVOL”
The command to send is {"req":{"set":{"AUDIOVOL":-30}}}
The response from the NPxx is {"res":{"set":{"AUDIOVOL":"Error - invalid parameter"}}}

Request-response packet when you want to know a parameter-names value.
We see haw to set a value for a specific parameter-name. In general, If we want to get from the NPxx the value of a specific parameter, the name/value pair is parameter-name : “”

Example
Get the volume value.
The command to send is {"req":{"get":{"AUVOL":""}}}
The response from the NPxx is {"res":{"get":{"AUVOL":-30}}}

Example
Get the volume and the Line In Mix values.
The command to send is {"req":{"get":{"AUVOL":"", "AUADMIXON": ""}}}
The response from the NPxx is {"res":{"get":{"AUVOL":-30, "AUADMIXON": 0}}}

User account

The server inside the NPxx device, manage two different accounts: guests and admin.
For both is it possible to customize its login password.
Their default passwords are
guest password = “guest”
admin password = “admin”

The server process one packet at a time, so if the web page is doing dialogue with the device firmware, in the same time it excludes the possible communication from a TCP-IP or RS485 client. These three possible communications are exclusive.

From the RS485 communication, it is not necessary to execute a login account.

From the TCP-IP and the web page communications, it is necessary to execute an account login.

If the account is “guest” and its password is “guest” (the default password), in the moment a client communicates with the NPxx it will open a new account section, without any login phase.
If the guest password is different from its default or the user want to access as an admin, it is necessary to execute an account login.

TCP-IP communication port

The TCP-IP port number, dedicated to communicating with a client, as request-response protocol, is 11000.
The port number 80 is dedicated to the web pages.

Request-Response protocol

After sent a packet to the NPxx device, before to send a new one, it is necessary to wait for the relative response.

RS485

The RS485 protocol is the same as the TCP-IP except for the termination character and the device ID field.
This bus must have the carriage return character at the end of the packet to send.

request:

1	2	2	3	4	5
Static Header **{“req”:**	Device ID **{ “IDDEV”:n,**	Command type **<CmdType> :**	JSON object **{ <Cmd> }**	Termination brackets **}}**	carriage return 0x20 or '\r'

response:

1	2	2	3	4	5
Static Header **{“res”:**	Device ID **{ “IDDEV”:n,**	Command type **<CmdType> :**	JSON object **{ <CmdResponse> }**	Termination brackets **}}**	carriage return 0x20 or '\r'

Example (Device ID = 3)
Set the volume at value -30dB. The command to send is {"req":{"IDDEV":3, "set":{"AUVOL":-30}}}'\r'

The Device ID is a number in the range between 0 and 252, where the value 0 is used for broadcast command type.

Example – Set value in any NPxx device connected to the RS485 bus.
Set the volume at value -30dB. The command to send is {"req":{"IDDEV":0, "set":{"AUVOL":-30}}}'\r'

When a broadcast command is received it will be processed, but no response it will be sent to the client.

The maximum buffer size for a generic packet, as it a request or a response, is 512 bytes.
After sent a packet to the NPxx device, before to send a new one, it is necessary to wait the relative response.
Pay attention to the “Response delay” parameter, because the NPxx send the response after the time indicated in this parameter.

From the RS485 communication, it is not necessary to execute a login account.

Parameters list

Parameter name:	**BRANDID**	
Operation Type	Values	Details and examples
get	string	Return the product name Request packet {"req":{"get":{"BRANDID":""}}} Response {"res":{"get":{"BRANDID":"NP10"}}}

Parameter name:	**SWVER**	
Operation Type	Values	Details and examples
get	String : 3 digits	Return the software version installed in the device Request packet {"req":{"get":{"SWVER":""}}} Response {"res":{"get":{"SWVER":"3.0.1"}}}

Parameter name:	**HWVER**	
Operation Type	Values	Details and examples
get	String : 2 digits	Return the device hardware version Request packet {"req":{"get":{"HWVER":""}}} Response {"res":{"get":{"HWVER":"1.0"}}}

Parameter name:	**BLVER**	
Operation Type	Values	Details and examples
get	String: 1 letter and 1 digit	Return the Boot Loader software version Request packet {"req":{"get":{"BLVER":""}}} Response {"res":{"get":{"BLVER":"A.3"}}}

Parameter name:	**SERIALNUM**	
Operation Type	Values	Details and examples
get	String: 8 digits	Return the device serial number Request packet {"req":{"get":{"SERIALNUM":""}}} Response {"res":{"get":{"SERIALNUM":"73700010"}}}

Parameter name:	**MACADDR**	
Operation Type	Values	Details and examples
get	String	Return the device MAC Address Request packet {"req":{"get":{"MACADDR":""}}} Response {"res":{"get":{"MACADDR":"70:B3:D5:FF:90:00"}}}

Parameter name:	**ETHDHCP**	
Operation Type	Values	Details and examples
get, set	Number	Read or Write the DHCP propriety in the ethernet setup 0 = DHCP OFF; 1 = DHCP ON Get Request packet {"req":{"get":{"ETHDHCP":""}}} Response {"res":{"get":{"ETHDHCP":1}}} Set Request packet {"req":{"set":{"ETHDHCP": 1}}} Response {"res":{"set":{"ETHDHCP": 1}}}

Parameter name:	**ETHIPAD**	
Operation Type	Values	Details and examples
get, set	String	Read or Write the TCP/IP (IPv4) device address Get Request packet {"req":{"get":{"ETHIPAD":""}}} Response {"res":{"get":{"ETHIPAD":"192.168.0.5"}}} Set Request packet {"req":{"set":{"ETHIPAD": “192.168.0.5""}}} Response {"res":{"set":{"ETHIPAD": “192.168.0.5""}}}

Parameter name:	**ETHNM**	
Operation Type	Values	Details and examples
get, set	String	Read or Write the Net Mask (IPv4) device address Get Request packet {"req":{"get":{"ETHNM":""}}} Response {"res":{"get":{"ETHNM":"255.255.255.0"}}} Set Request packet {"req":{"set":{"ETHNM":"255.255.255.0"}}} Response {"res":{"set":{"ETHNM":"255.255.255.0"}}}

Parameter name:	**ETHGAD**	
Operation Type	Values	Details and examples
get, set	String	Read or Write the Gateway (IPv4) device address Get Request packet {"req":{"get":{"ETHGAD":""}}} Response {"res":{"get":{"ETHGAD":"192.168.0.1"}}} Set Request packet {"req":{"set":{"ETHGAD":"192.168.0.1"}}} Response {"res":{"set":{"ETHGAD":"192.168.0.1"}}}

Parameter name:	**ETHDNS1**	
Operation Type	Values	Details and examples
get, set	String	Read or Write the Primary (IPv4) DNS device address Get Request packet {"req":{"get":{"ETHDNS1":""}}} Response {"res":{"get":{"ETHDNS1":"192.168.0.150"}}} Set Request packet {"req":{"set":{"ETHDNS1":"192.168.0.150"}}} Response {"res":{"set":{"ETHDNS1":"192.168.0.150"}}}

Parameter name:	**ETHDNS2**	
Operation Type	Values	Details and examples
get, set	String	Read or Write the Secondary (IPv4) DNS device address Get Request packet {"req":{"get":{"ETHDNS2":""}}} Response {"res":{"get":{"ETHDNS2":"192.168.0.151"}}} Set Request packet {"req":{"set":{"ETHDNS2":"192.168.0.151"}}} Response {"res":{"set":{"ETHDNS2":"192.168.0.151"}}}

Parameter name:	**Date**	
Operation Type	Values	Details and examples
get, set	Numbers RTCCY = Year [20xx] RTCCM = Month [1,..,12] 1=January,..,12=December RTCCD = Day [1,..,31] RTCCWD = Week Day [1,..7] 1=Monday,..,7=Sunday	Read or Write the RTC clock date Get Request packet {"req":{"get":{"RTCCY":"","RTCCM":"","RTCCD":"","RTCCWD":""}}} Response {"res":{"get":{"RTCCY":2017,"RTCCM":11,"RTCCD":2,"RTCCWD":4}}} Thursday, 2nd November 2017 Set Request packet {"req":{"set":{"RTCCY":2017,"RTCCM":11,"RTCCD":2,"RTCCWD":4}}} Response {"res":{"set":{"RTCCY":2017,"RTCCM":11,"RTCCD":2,"RTCCWD":4}}}

Parameter name:	**Time**	
Operation Type	Values	Details and examples
get, set	Numbers RTCCHH = Hours [0,..,23] RTCCMM = Minutes[0,..,59] RTCCSS = Seconds [0,..,59]	Read or Write the RTC clock time Get Request packet {"req":{"get":{"RTCCHH":"","RTCCMM":"","RTCCSS":""}}} Response {"res":{"get":{"RTCCHH":11,"RTCCMM":32,"RTCCSS":20}}} 11:32:20 Set Request packet {"req":{"set":{"RTCCHH":11,"RTCCMM":32,"RTCCSS":20}}} Response {"res":{"set":{"RTCCHH":11,"RTCCMM":32,"RTCCSS":20}}}

Parameter name:	**Device Configuration**	
Operation Type	Values	Details and examples
get, set	Number DEVFN: device function 0 = Standerd Player 1 = Playlist Sequence 2 = Advanced Player 3 = Advanced Playlist 4 = Scheduler	Device function, set the main functionality of NPxx. When changed, is suggested to read the status and the others parameters managed in that moment. This command must be send alone, because the NPxx device change a lot of things inside itself and require 2 seconds to done it. Get Request packet {"req":{"get":{"DEVFN":""}}} Response {"res":{"get":{"DEVFN":2}}} Advanced Player Set Request packet {"req":{"set":{"DEVFN": 4}}} Scheduler Response {"res":{"set":{"DEVFN": 4}}}

Parameter name:	**Login**	
Operation Type	Values	Details and examples
get, set	String LOGINTYPE: account type [guest, admin] LOGINPWD Password string for guest or admin LOGOUT The login status has a timeout of 10 minutes, without any operation	Read the Login status or make the account access Default passwords for the two possible accounts are: “guest” for the “guest” account; “admin” for the “admin” accounts If the account “guest” has its default password, the login become immediately, without any password request. Get Request packet {"req":{"get":{"LOGINTYPE":""}}} Response {"res":{"get":{"LOGINTYPE":"admin"}}} or {"res":{"get":{"LOGINTYPE":"guest"}}} if user is logged or {"res":{"get":{"LOGINTYPE":"none"}}} if the user is not logged and the “guest” account has a password Set – login as “guest”, with password = “sS25k#Hc57” Request packet {"req":{"set":{"LOGINTYPE":"guest", "LOGINPWD":"sS25k#Hc57"}}} Response Login successfully {"res":{"set":{"LOGINTYPE":"guest"}}} or Wrong Password {"res":{"set":{"LOGINTYPE":"none"}}} The same example for the “admin” login type: {"req":{"set":{"LOGINTYPE":"admin", "LOGINPWD":"Avk34c67"}}} Login successfully {"res":{"set":{"LOGINTYPE":"admin"}}} or Wrong Password {"res":{"set":{"LOGINTYPE":"none"}}} Set – logout Request packet {"req":{"set":{"LOGOUT":""}}} Response {"res":{"set":{"LOGOUT":""}}}

Parameter name:	**DIOC**	
Operation Type	Values	Details and examples
get	Number Digital Inputs/Outputs configuration [0,..,255]	Read the Input/Output configuration. There are 8 configurable I/O, each one can be defined as input or output. A byte, defined from 8 bits, explain the 8 I/O configurations: 0 = output 1 = input In the following order little-endian: I/O8 – I/O7 - I/O6 - I/O5 - I/O4 - I/O3 - I/O2 – I/O1 Example: I/O7, I/O3 and I/O2 defined as outputs and the others as inputs, the byte configuration will be 10111001b = B9H = 185 Get Request packet {"req":{"get":{"DIOC":""}}} Response {"res":{"get":{"DIOC":185}}}

Parameter name:	**DIBC**	
Operation Type	Values	Details and examples
get, set	Number Number of Inputs/Outputs enabled [1,..,8]	Read or Write the number of I/O enabled. Only the I/O enabled can be set, if they are in outputs, or read the input level, if they are defined as inputs (read also the DIOC parameter). At minimum it is possible to set only one I/O enabled. The number of I/O enabled start from the I/O1 and proceed in great order. Example: 3 I/O enabled I/O1, I/O2 and I/O3 are enabled, the others I/Os are disabled DIBC parameter is available only when NPxx is configured as “Advanced Player” or “Advanded Playlist” Get Request packet {"req":{"get":{"DIBC":""}}} Response {"res":{"get":{"DIBC":4}}} I/O enabled are: I/O1 - I/O2 - I/O3 – I/O4 Set Request packet {"req":{"set":{"DIBC": 4}}} Response {"res":{"set":{"DIBC": 4}}}

Parameter name:	**DISTS**	
Operation Type	Values	Details and examples
get	Number Digital In Status 0 = Low 1 = High	Read the I/O status for the I/O defined as inputs (read also the DIOC parameter). 0 = Low 1 = High They are logical status, because in the I/O configuration web page it is possible to invert the digital level value. The I/O defined as Outputs return the correspond bit at zero value. Example: I/O7, I/O3 and I/O2 are defined as outputs and the others as inputs. In I/O8, and I/O6 have an high input level and a low level in the other inputs. Any I/O has the inversion level mask disabled except for the I/O1. I/O7, I/O3, I/O2 = 0 because they are outputs I/O8, I/O6 = 1 because they have an high input level I/O5, I/O4 = 0 because they have a low input level I/O1 = 1 because it has a low input level, but its inversion level mask is enabled The returned byte is: 10100001b = A1H = 161 Get Request packet {"req":{"get":{"DISTS":""}}} Response {"res":{"get":{"DISTS":161}}}

Parameter name:	**DOSET1,…,DOSET8**	
Operation Type	Values	Details and examples
get, set	Number Read or Write the status from the Output1 to the Output8 0 = Low 1 = High	Read the I/O status for the I/O defined as outputs (read also the DIOC parameter). 0 = Low 1 = High They are logical status, because in the I/O configuration web page it is possible to invert the digital level value. The I/O defined as Inputs return a value equal to zero. DIOSET1 correspond to the I/O1. The same logical for the others. Get Request packet {"req":{"get":{"DOSET1":""}}} Response {"res":{"get":{"DOSET1":1}}} Set Request packet {"req":{"set":{"DOSET1": 1}}} Response {"res":{"set":{"DOSET1": 1}}}

Parameter name:	**AUVOL**	
Operation Type	Values	Details and examples
get, set	Number Read or Write the audio volume	The audio volume has a rage of values from 0dB to -64dB. The 0dB correspond to the maximum value. The minimum regulation step is 1. Get Request packet {"req":{"get":{"AUVOL":""}}} Response {"res":{"get":{"AUVOL":-20}}} Set Request packet {"req":{"set":{"AUVOL": -32}}} Response {"res":{"set":{"AUVOL": -32}}}

Parameter name:	**AUM**	
Operation Type	Values	Details and examples
get, set	Number Audio Mute 0 = Disable (1 = reserved) 2 = Auto	Manage the audio Mute propriety: Amplifier module; Line In and DSP processing. Disable: the audio is changed through the single devices setup: Amplifier module; Line In and DSP processing. Auto: mute propriety is activated when there aren't any audio DSP processing over 2 seconds and Line In is disabled. Get Request packet {"req":{"get":{"AUM":""}}} Response {"res":{"get":{"AUM": 2}}} Set Request packet {"req":{"set":{"AUM": 0}}} Response {"res":{"set":{"AUM": 0}}}

Parameter name:	**AUADMIXON**	
Operation Type	Values	Details and examples
get, set	Number Audio Line In Mix status 0 = Disable 1 = Enable	Read or Write the Audio Line In Mix status. 0 = Disable 1 = Enable When enabled the NPxx mix the audio coming from the Line In and the audio processing from the internal DSP. Get Request packet {"req":{"get":{"AUADMIXON":""}}} Response {"res":{"get":{"AUADMIXON": 0}}} Set Request packet {"req":{"set":{"AUADMIXON": 1}}} Response {"res":{"set":{"AUADMIXON": 1}}}

Parameter name:	**AULINL**	
Operation Type	Values	Details and examples
get, set	Number Line In audio level [-3,..,-31]	Read or Write the Line In audio level. Minimum value is -31dB. Maximum level is -3dB. Get Request packet {"req":{"get":{"AULINL":""}}} Response {"res":{"get":{"AULINL": -12}}} Set Request packet {"req":{"set":{"AULINL": -5}}} Response {"res":{"set":{"AULINL": -5}}}

Parameter name:	**AULINL2**	
Operation Type	Values	Details and examples
get, set	Number Line In audio level Mix [-3,..,-31]	Line In audio level when mixed with audio generated from the internal DSP processing. As for the Line In audio Level, the Minimum value is -31dB and the Maximum level is -3dB. But, for this mix level, its level must be less than the value of Line In audio Level: Line In audio level > Line In audio level Mix Get Request packet {"req":{"get":{"AULINL2":""}}} Response {"res":{"get":{"AULINL2": -7}}} Set Request packet {"req":{"set":{"AULINL2": -10}}} Response {"res":{"set":{"AULINL2": -10}}}

Parameter name:	**RELSET**	
Operation Type	Values	Details and examples
get, set	Number Relay Status 0 = Off 1 = On	Read or Write the Relay status. This is the Relay logic value, because in the Relay settings web page, there is the "Logics" parameter, with possible value of "Normal or "Inverted", that confirm or invert the physical value. Warning: when the relay is activated, pay attention if the relative driven hardware can tolerate this condition. Get Request packet {"req":{"get":{"RELSET":""}}} Response {"res":{"get":{"RELSET": 0}}} Relay has logical value Off Set Request packet {"req":{"set":{"RELSET": 1}}} Response {"res":{"set":{"RELSET": 1}}}

Parameter name:	**SENSON**	
Operation Type	Values	Details and examples
get, set	Number Sensor 0 = Disable 1 = Enable	Read or Write the sensor status. Get Request packet {"req":{"get":{"SENSON":""}}} Response {"res":{"get":{"SENSON": 0}}} Sensor status is disabled Set: enable the sensor functions Request packet {"req":{"set":{"SENSON": 1}}} Response {"res":{"set":{"SENSON": 1}}}

Parameter name:	**NTPSYNCNOW**	
Operation Type	Values	Details and examples
set	Number	Execute the NTP synchronization, to the specified server in the "Date&Time settings" web page. This operation start only if the NTP services is enabled. After this command it is necessary to wait 5 seconds, in order to conclude the sync process. Set Request packet {"req":{"set":{"NTPSYNCNOW": 1}}} Response {"res":{"set":{"NTPSYNCNOW": 1}}}

Parameter name:	**SD**	
Operation Type	Values	Details and examples
get	Number 0 = unplugged 1 = plugged	Read the micro SD status: - Plugged or not - SD size [Kbyte] - SD size free [KByte] Get Request packet {"req":{"get":{"SDPRESENT":"","SDSIZE":"","SDFREE":""}}} Response {"res":{"get":{"SDPRESENT": 1,"SDSIZE": 3922,"SDFREE": 1082}}} Micro SD is plugged. It has a total size of 3922KByte, that 1082Kbyte are free.

Parameter name:	**Read music audio tracks**	
Operation Type	Values	Details and examples
get	Strings This command return the list of audio files inside the directory "music"	Suppose that in the SD card there are N audio files, in the "music" directory. Request packet {"req":{"get":{"MUSIC_ITEMS":""}}} Response {"res":{"get":{"MUSIC_ITEMS": [{<info file 1>},..,{<info file N>}]}}} The generic <info file J> has the information regarding the generic audio file at position number J, in the file system, inside the micro SD card. <info file J> = "<file-nameJ>": [LJ, GJ, "<artistJ>", "<titleJ>","<albumJ>"] <file-nameJ> = file-name of audio file at position number J, inside the micro SD card LJ = length in seconds of this audio file, with name = file-name GJ = Genre index of this audio file, with name = file-name. See the list in the Appendix 1 <artistJ> = Artist's name of this audio file, with name = file-name <titleJ> = Title of this audio file, with name = file-name <albumJ> = Album of this audio file, with name = file-name There are some limits in the maximum length for the strings about the file-name, the artist, the title and the album. If they will exceeds these limits, the string will be cut out with the character "@". This character is also used when the Id3tag value is empty. Maximum length for the strings: Max_File_Name_Length = 50 Max_Artist_Length = 30 Max_Title_Length = 50 Max_Album_Length = 30 The maximum TCP-IP payload buffer length, that could be received, from the NPxx device, is 1024 bytes. The maximum TCP-IP payload buffer length, that could be be transmit, from the NPxx device, is 1400 bytes.

		If the data [{<info file 1>},..,{<info file N>}] exceed the 1400 bytes, it will be sent using more than one packets. For example, if the number of files which info could be contained in 1400 bytes are 136, the first received packet has these contents: {"res":{"get":{"MUSIC_ITEMS": [{<info file 1>},..,{<info file 135>}], "NEXT": 136}}} In order to have the other music files info, it is necessary send a new request with these contents: {"req":{"get":{"MUSIC_ITEMS":"", "NEXT", 136}}} With this new request, the user ask to the NPxx to send the music file info from the next file with index equal to 136, in the file system. This process could continue for different packets until you receive one without the object "NEXT", N , where N is a number. Example: there are 2 audio files in the SD card: “time.mp3” and “ White As now.mp3”. Request packet {"req":{"get":{"MUSIC_ITEMS":""}}} Response {"res":{"get":{"MUSIC_ITEMS": [{"time.mp3": [224, 13, "Culture Club","Time (Clock of the Heart)","The Best"]}, {"White As Snow.mp3": [281, 17, "U2","White As Snow", "No line on the horizon"]}] }}}

Parameter name:		**Read spot audio tracks**
Operation Type	Values	Details and examples
get	Strings This command return the list of audio files inside the “spot” directory	Same roles and process described in the command for “Read music audio track”, but change the command name. Request packet {"req":{"get":{"SPOT_ITEMS":""}}} Response {"res":{"get":{"SPOT_ITEMS": [{<info file 1>},..,{<info file N>}] }}}

Parameter name:		**Read sensor audio tracks**
Operation Type	Values	Details and examples
get	Strings This command return the list of audio files inside the “sensor” directory	Same roles and process described in the command for “Read music audio track”, but change the command name. Request packet {"req":{"get":{"SENSOR_ITEMS":""}}} Response {"res":{"get":{"SENSOR_ITEMS": [{<info file 1>},..,{<info file N>}] }}}

Parameter name:	**Read Playlists**	
Operation Type	Values	Details and examples
get	Strings This command return the list of playlist files inside the directory “playlist”	Get Request packet {"req":{"get":{"PL_ITEMS":""}}} Response {"res":{"get":{"PL_ITEMS": [{<info PL file 1>},..,{<info PL file N>}] }}} The generic <info PL file J> has the information regarding the generic playlist file at position number J, in the file system, inside the micro SD card. <info PL file J> = "<PL file-nameJ>": [LJ] <PL file-nameJ> = file-name of playlist file at position number J, inside the micro SD card LJ = length in seconds of this playlist file, with name = PL file-name The maximum length for the file-name are 50 characters. If it will exceeds these limit, the string will be cut out with the character "@". If the data [{<info PL file 1>},..,{<info PL file N>}] exceed the 10KByte, it will be sent using more than one packets, as described in the “MUSIC_ITEMS” command. Example: there are 3 playlist files in the SD card: “Culture Club.m3u”, “The Police.m3u” and “U2 No Line on Horrizon.m3u”. Request packet {"req":{"get":{"PL_ITEMS":""}}} Response {"res":{"get":{"PL_ITEMS": [{"Culture Club.m3u": [3695]},{"The Police.m3u": [3307]},{"U2 No Line on Horrizon.m3u": [3227]}] }}} With a different layout, the response packet is {"res":{"get":{"PL_ITEMS": [{"Culture Club.m3u": [3695]}, {"The Police.m3u": [3307]}, {"U2 No Line on Horrizon.m3u": [3227]}] }}}

Parameter name:	**Play audio track**	
Operation Type	Values	Details and examples
cmd	String This command manage the play functions of an audio file, inside the directories “music”, “spot” and “sensor”. Pause Play Stop playonly	This command has 3 parameters: "TYPE": type of command = “play”, “stop”, “pause” "FILE_TYPE": system directory where the file in stored = "music", “spot”, “sensor” "TRACK_NAME": the name of audio file; file-name and extension (“mp3”). Example Play the file “time.mp3”, present in the system directory “music” Request packet {"req":{"cmd":{"TYPE":"play", "FILE_TYPE": "music", "TRACK_NAME": “time.mp3”}}} Response {"res":{"cmd":”done”}} Example Stop the audio file in play. Request packet {"req":{"cmd":{"TYPE":"stop"}}} Response {"res":{"cmd":”done”}} Example Pause the audio file in play. Request packet {"req":{"cmd":{"TYPE":"pause"}}} Response {"res":{"cmd":”done”}} At the end of audio file, present in the command, the player continue to play with the next file in its file system. playonly At the end of audio file, present in the command, the player stops to produce any audio and wait an user command or event Request packet {"req":{"cmd":{"TYPE":"playonly", "FILE_TYPE": "music", "TRACK_NAME": “time.mp3”}}} Response {"res":{"cmd":”done”}}

Parameter name:	**Play playlist**	
Operation Type	Values	Details and examples
cmd	String This command manage the play functions of a playlist file, inside the directory “playlist” Pause Play Stop Rewind startplaylistonly	This command has 3 parameters: "TYPE": type of command = “startplaylist”, “stop”, “pauseplaylist”, “rewind” "FILE_TYPE": system directory where the file in stored. For the playlist the value must be “playlist”. "PL_NAME": the name of playlist file; file-name and extension (“m3u”). Before to execute the “rewind” function, if the same playlist is in play, it will be stopped. Example Start to play the playlist file “Culture Club.m3u”. Request packet {"req":{"cmd":{"TYPE":"startplaylist", "FILE_TYPE": "playlist", "PL_NAME": “Culture Club.m3u”}}} Response {"res":{"cmd":”done”}} Example Put in pause the playlist file “Culture Club.m3u”. Request packet {"req":{"cmd":{"TYPE":"pauseplaylist", "FILE_TYPE": "playlist", "PL_NAME": “Culture Club.m3u”}}} Response {"res":{"cmd":”done”}} Example Stop to play the current playlist in play. Request packet {"req":{"cmd":{"TYPE":"stop"}}} Response {"res":{"cmd":”done”}} Example Rewind the playlist “Culture Club.m3u”. Request packet {"req":{"cmd":{"TYPE":"rewplaylist", "FILE_TYPE": "playlist", "PL_NAME": “Culture Club.m3u”}}} Response {"res":{"cmd":”done”}} At the end of Playlist file, present in the command, the player continue to play with the next playlist file in its file system startplaylistonly At the end of last audio file, inside the Playlist present in the command, the player stops to produce any audio and wait an user command or event Example Rewind the playlist “Culture Club.m3u”. Request packet {"req":{"cmd":{"TYPE":" startplaylistonly", "FILE_TYPE":"playlist", "PL_NAME": “Culture Club.m3u”}}} Response {"res":{"cmd":”done”}}

Parameter name:		**INTTEMP**
Operation Type	Values	Details and examples
get	Number Internal Micro-controller temperature	Read the internal micro-controller temperature. In comparison to the external chassis temperature (ambient), could be a difference of 10 °C. Request packet {"req":{"get":{"INTTEMP": ""}}} Response {"res":{"get":{"INTTEMP": 26}}} The internal micro-controller temperature is equal to 26°C

Parameter name:		**SWRESET**
Operation Type	Values	Details and examples
set		Execute a software reset inside the NPxx device. This command restart the NPxx device. Request packet {"req":{"set":{"SWRESET": 1}}} Response {"res":{"set":{"SWRESET": 1}}}

Parameter name:		**SUSPENSION**
Operation Type	Values	Details and examples
get		Return the Suspension info status. This is a possible state in Scheduler mode, with Digital I/O Mode in “Priority Message”. In “Priority Message” status, the I/O #8 have a particular role: if it is activated it suspend the present scheduler until it will be disactivated. This particular status could be monitored with the SUSPENSION command: {"req":{"get": {"SUSPENSION": ""}}} The response could be have 2 possible values: 0 = not suspended; 1 = suspension status is on

Parameter name:		**DATETIME**
Operation Type	Values	Details and examples
get		Return the date and time info in a single formatted string Request packet {"req":{"get": {"DATETIME": ""}}} Response {"res":{"get":{"DATETIME":"2021-01-15,08:06:52"}}}

Parameter name:	**INFOPLAY**	
Operation Type	Values	Details and examples
get	Return JSON object with values as string and numbers. In general, this command return the information about audio track, playlist or scheduler that is in play in this moment. If one or more of these elements are not in play, the relative key-value will be empty.	Request packet {"req":{"get": {"INFOPLAY": ""}}} The generic response is {"res":{"get": {"INFOPLAY": {"status":<status>,"scheduler":<scheduler name>, "playlist":<playlist name>, "trackname":<audio track name>, "trackinfo": [<Len>,<genre>,<artist>,<title>,<album>] }}}}} Where <status > could have these values: ["stop", "play", "pause", "suspension" (this last one only in scheduler mode)] <Len> is the song's length, in seconds The key-values <scheduler name> and <playlist name> could be empty in cases the Device Configuration does not have these elements, like in StandardPlayer and AdvancePlayer modes. In PlaylistSequence or AdvancePlaylist could be empty the <scheduler name> value. In the other cases, the "scheduler", "playlist" and "trackname" have the name that Player has in focus, or ready to play. In case no audio track is in play, the response is {"res":{"get":{"INFOPLAY":{"status":"stop"}}}} If Device Configuration is Standard Player, and the audio track "A church is burning.mp3" is in play, the response is: {"res":{"get": {"INFOPLAY": {"status":"play","scheduler":"","playlist":"", "trackname": "A church is burning.mp3", "trackinfo": [203,80,"Simon e Garfunkel","A church is burning","The essential CD02"]}}}} In the same case, but with Device Configuration equal to AdvancePlaylist, the response is: {"res":{"get": {"INFOPLAY": {"status":"play","scheduler":"","playlist": "my_simon_PL.m3u", "trackname": "A church is burning.mp3", "trackinfo": [203,80,"Simon e Garfunkel","A church is burning","The essential CD02"]}}}} If we are in scheduler mode, the response is: {"res":{"get": {"INFOPLAY": {"status":"play","scheduler":"MyFavouriteSongs", "playlist": "my_simon_PL.m3u", "trackname": "A church is burning.mp3", "trackinfo": [203,80,"Simon e Garfunkel","A church is burning","The essential CD02"]}}}} In Scheduler, if is in play an audio spot, the response is: {"res":{"get": {"INFOPLAY": {"status":"play","scheduler":"From Wednesday2Sunday", "playlist": "", "trackname": "spotLavazza.mp3", "trackinfo": [15,10,"Lavazza","LavazzaCremaGusto","Lavazza2"]}}}}

Parameter name:		**Virtual keypad**
Operation Type	Values	Details and examples
cmd		These commands reproduce the same event you have when you press the buttons present in the NP10 model: “Previous”, “Stop”, “Play” and “Next”. If no audio file is in play the Previous become Audio-Decrement and Next become Audio-Increment. The functionality of any virtual key (“Previous”, “Stop”, “Play” and “Next”) is associated to the relative Device Configuration activated. {"req":{"cmd":{"TYPE":"VKPprev"}}} {"req":{"cmd":{"TYPE":"VKPstop"}}} {"req":{"cmd":{"TYPE":"VKPplay"}}} {"req":{"cmd":{"TYPE":"VKPnext"}}} For any of these commands the response is: {"res":{"cmd":{"retmsg":"done"}}}

Parameter name:		**Delete File**
Operation Type	Values	Details and examples
cmd		Delete the file indicated in the command. The generic JSON command is {"req": {"cmd": {"TYPE": "deletefile", "FILE_TYPE": <type>, "FILE_NAME": <name> }}} Where <type> is a string and its possible value are ["music", "playlist", "sensor", "spot", "scheduler"] <name> is a string indicated the file name to delete, including its extension. The "FILE_TYPE" field indicate where to find the file name. Example: delete "myaudio.mp3" file in the “music” directory {"req": {"cmd": {"TYPE": "deletefile", "FILE_TYPE": "music", "FILE_NAME": "myaudio.mp3"}}} Example: delete "myspot.mp3" file in the “spot” directory {"req": {"cmd": {"TYPE": "deletefile", "FILE_TYPE": "spot", "FILE_NAME": "myspot.mp3"}}} Example: delete "mysensoraudiofile.mp3" file in the “sensor” directory {"req": {"cmd": {"TYPE": "deletefile", "FILE_TYPE": "sensor", "FILE_NAME": "mysensoraudiofile.mp3"}}} Example: delete "myplaylist.m3u" file in the “playlist” directory {"req": {"cmd": {"TYPE": "deletefile", "FILE_TYPE": " playlist", "FILE_NAME": "myplaylist.m3u"}}} Example: delete "myscheduler.txt" file in the “scheduler” directory {"req": {"cmd": {"TYPE": "deletefile", "FILE_TYPE": "scheduler", "FILE_NAME": " myscheduler.txt"}}} Pay attention: the scheduler file, in the scheduler web page, is showed without the file extension “txt”, but it is present in the player’s file system. WARNING: The delete function could be very critical, because you can delete files inside playlists or schedulers, compromising its behaviors and correct executions.