Integrating the NuVo Multiroom

In this tutorial we will show how to use **DOMIQ** modules to integrate with any devices using RS-232 transmission protocol. **Serial-2SG** has built-in RS-232 transmission protocol. Following speeds are supported: 9600, 19200, 38400 and 57600 bps and frame formats: 8N1, 8N2, 8E1, 8O1. It is possible to connect up to two **DOMIQ/Serial-2SG** modules to one **Base** module. Typically **Serial-2SG** modules have assigned address 1. If you plan to connect two **Serial-2SG** modules to the Base module, please let us know. The second **Serial-2SG** module will have assigned address 2.

In further part of this tutorial we will describe the integration process of **DOMIQ** system and devices that use RS-232 transmission protocol on the example of **NuVo** multiroom audio system. By reading this manual you will learn how to:

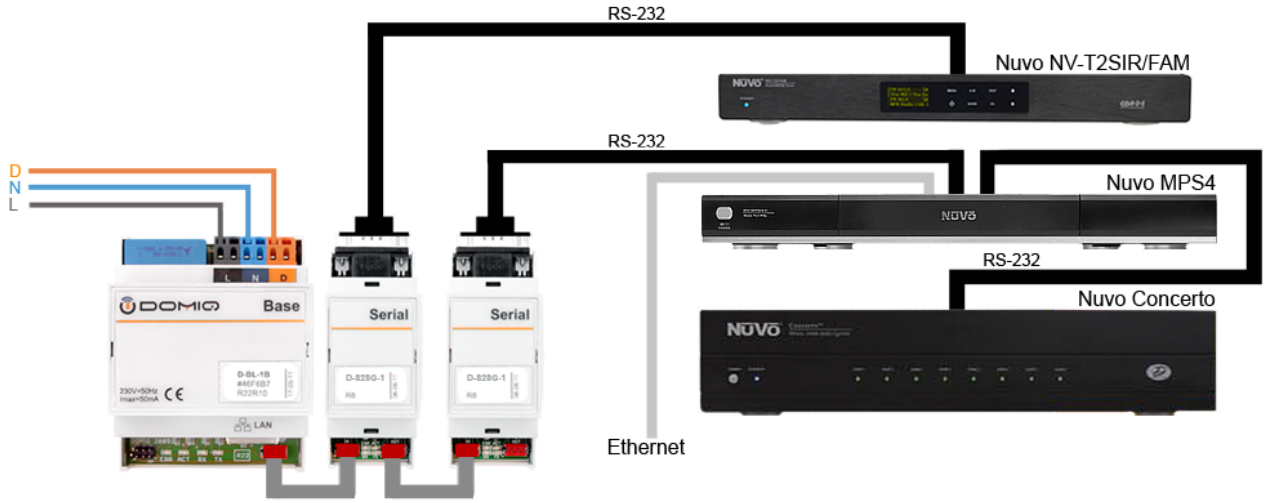
- Use **DOMIQ** modules to send and receive RS-232 messages.
- Integrate multiroom audio system with home automation.
- Process the data read by the **Serial-2SG** module.
- Control multiroom audio system with **Display** visualization.

1. Multiroom NuVo

Multiroom system is an advanced audio system that allows to play audio such as: radio stations and mp3 in several zones simultaneously. Each zone can play sound from any available audio source. The set loaned to us for testing consisted of the following devices:

- **NuVo Concerto** - Audio aplifier. Supports up to 8 zones.
- **NuVo NV-T2SIR/FAM** - A device with two built-in radio tuners. Allows to listen to radio as two independent audio streams.
- **NuVo MPS4** - MP3 files server with four built-in independent audio sources. If the device has connection to the Internet, it is possible to play Internet radio stations. Each source can play different audio files or radio stations. Also if NuVo MPS4 is connected to local network, it is possible to control it using TCP/IP protocol.
- **NuVo NV-I8GCP** - Wall mounted touch panels, allows to control NuVo devices (change settings, choose audio source, volume control, browsing mp3 etc.)

2. Connections



Source: www.nuvotechnologies.com.

For connections marked as **RS-232** standard RS-232 cable with DB-9 connectors is used. This wiring diagram shows only simplified connection of devices. NuVo devices must be connected according to the relevant manufacturer's installation manuals.

3. DOMIQ modules configuration

3.1. DOMIQ/Serial-2SG configuration

Serial-2SG module is configured by setting RS-232 transmission parameters. Following configuration command is used:

`LC.SER.config.<module address>=<transmission speed> <frame format>`

Example:

`LC.SER.config.1=38400 8N1`

Serial-2SG module supports RS-232 transmission protocol at speeds of: 9600, 19200, 38400 and 57600 bps and frame formats: 8N1, 8N2, 8E1, 8O1.

Transmission parameters have to be set each time the **Base** module is restarted. You can achieve that by using the **Logic** tab and placing there a single line of code with the following syntax:

`command("LC.SER.config.<module address>=<transmission speed> <frame format>")`

3.2. Control commands

NuVo multiroom system is fully configurable. With descriptions of NuVo communication protocols and **DOMIQ** modules you can control all available functions offered by the multiroom audio system, such as: on/off for each zone, changing the signal source, control volume, tuning radio tuners, display RDS information etc. and integrate multiroom audio system with home automation.

Two commands are used:

- **Sending commands**: `LC.SER.line.<module address>=<command>`

Example: `LC.SER.line.1=*Z1ON` – turning on of the first zone.

- **Receiving commands**: `LE.SER.line.<module address>=<command>`

if the NuVo MPS4 music server is connected to the Ethernet, it is possible to control it using TCP/IP protocol. Following command is used:

`C.TCP.send.<NuVo MPS4 IP address:port>=<command>`

Example: `C.TCP.send.192.168.10.182:5004=PlayPause` - simulation of pressing *Play/Pause* button.

Full list of available commands can be found in description of communication protocol.

Control command can be send as:

- Result of pressing a button.
- Result of an event.
- Result of a timer.
- Element of logical function.

4. Integration

DOMIQ/Serial-2SG allows for full, bidirectional integration of NuVo multiroom audio system with home automation system. Following are few examples of such integration.

4.1. Music Alarm Clock

Using **Timers** and **Logic** combined with the multiroom system allows you to define an music alarm clock. The automation system will run the multiroom system in selected zones at a specific time, play the favourite music or radio station and set the volume to a specific value. The definition of such alarm clock is a two-step process. The first step is a definition of logical rules. In the second step you will have to set a timer.

4.1.1. Defining Logical Rules

1. Choose **Logic** tab in the **DOMIQ/Base Configurator**.
2. Enter the source code based on the code presented below:

```
function clock()
     command ("LC.SER.line.1=*Z1ON\r*Z1SRC5\r*Z1VOL50") -- Zone 1
     command ("LC.SER.line.2=*T'A'FM98.8")
     command ("LC.SER.line.1=*Z2ON\r*Z2SRC6\r*Z2VOL45") -- Zone 2
     command ("LC.SER.line.2=*T'B'FM103.0")
     command ("LC.SER.line.1=*Z3ON\r*Z3SRC1\r*Z3VOL45") -- Zone 3
     command ("C.TCP.send.<music server IP addres>=PlayPlaylist
<playlist name> True")
end
```

A few words of explanation to the presented code:

Clock function is a set of commands sent to the NuVo devices. The first command is sent to the amplifier. It turns the first zone on, assigns source 5 to the zone and sets the volume to 50.

The second command is sent to the first radio tuner. It sets tuner frequency to 101.0 MHz.

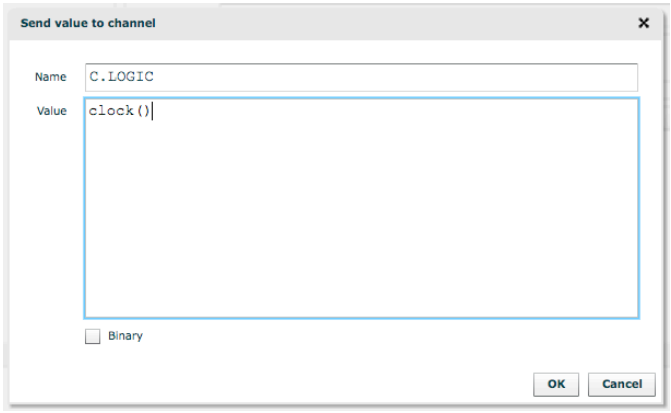
Later we configure zones 2 and 3 in similar fashion to zone 1.

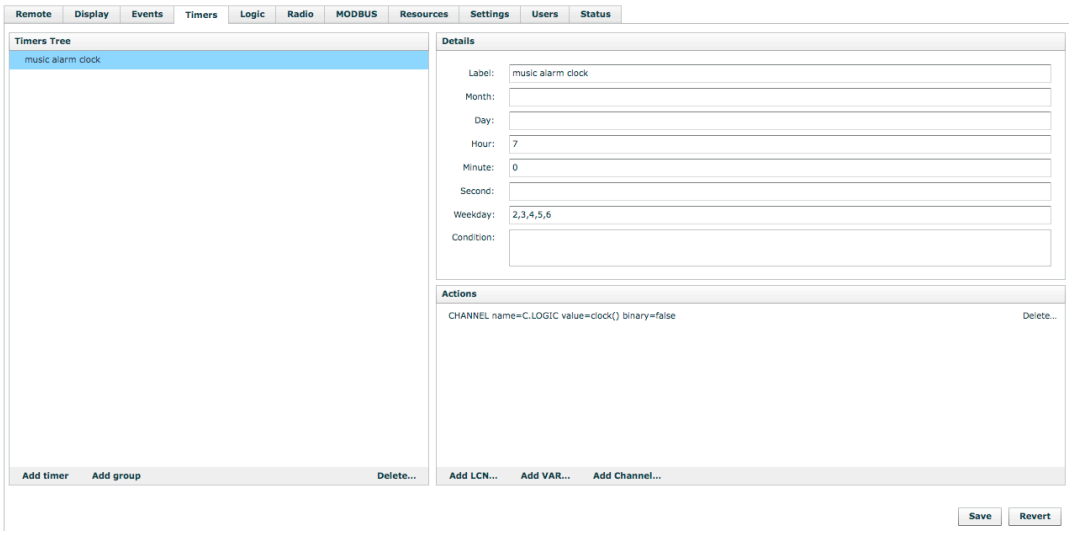
The last command is sent to the music server using TCP/IP protocol. `PlayPlaylist` command runs the playlist with the given name. This function can be freely modified to play sound in other zones or from another source.

4.1.2. Timer Definition

In the desribed example, timer will be triggered from Monday to Friday at 7.00am.

1. Choose **Timers** tab.
2. Click on the **Add** button, to add new timer.
3. Fill **Hour**, **Minute** and **Weekday** fields to define when the timer will be triggered.
4. In the **Actions** box click on the **Add Channel...** button.
5. In the window that appears, in the **Name** field type: `C.LOGIC` and in the **Value** field enter the function name, in this case `clock()`.





5. Parsing messages

DOMIQ/Serial-2SG module buffers the data until the carriage return marker is received. Structure of the data depends on the device to which you connect. In order to display the data in a form readable for user, it is necessary to perform parsing. We provide as an example configurable parser object designed to work in Logic environment.

In order to use the parser, you need to upload the **parser.lua** file to your **Base** module (**Resources** tab, **Scripts** section). The contents of the **parser.lua** file is added as an appendix to this tutorial, and is provided as download on DOMIQ tutorial page.

Parser source code is universal and can be used to parse any messages. The parser compares the received messages with matching patterns. If the message match with the pattern, then parser calls one of the defined function that processes the received data.

Example:

```
import ’parser’
p = parser()
p:add(”ADD(%d)”,function(n) print(n+1) end)
p:add(”SUB(%d)”,function(n) print(n-1) end)
p:parse(”ADD1”)
p:parse(”SUB2”)
```

In the result program will print:

```
2
1
```

You can read more about matching patterns in the "Lua Reference Manual", chapter 5.4.1 at: `http://www.lua.org/manual/5.1/`

More practical approach is to store data in **Base** state. It allows to display data on the **Display** visualizations or in the **Remote** application menu. In further part of this chapter you can read about using parser to display RDS messages on **Display** visualization.

5.1. Parsing RDS messages

RDS messages are sent using **#SsDISPLINE** command, where **s** is a number of a signal source. In order to display RDS messages, paste the following source code to the **Logic** tab:

```
import 'parser'

function nuvo(tab)
  local dev = {}
  local zones = {}
  local tun = assert(tab.tun)
  local amp = assert(tab.amp)
  local pamp = parser()
  local ptun = parser()

  -- Parsing tuner messages
  function tun:onchange(data)
    print("TUN: "..data)
    ptun:parse(data)
```

```
  end
  function amp:onchange(data)
    print("AMP: "..data)
    pamp:parse(data)
  end

  --Parsing #SsDISPLINE command messages
  pamp:add([[#S(%d)DISPLINE(%d),"([^"]+)"]],function(s,l,txt)
    local sid = tonumber(s)
    for k,v in pairs(zones) do
      if v == sid then set("NUVO.line."..k.."."..l,txt) end
    end
  end)

  -- Parsing status messages of each zone
  pamp:add([[#Z(%d),ON,SRC(%d),VOL(%d+),DND(%d),LOCK(%d)]]
  function(z,src,vol,dnd,lock)
    local prefix="NUVO."..z.."."
    set(prefix.."vol",vol)
    set(prefix.."src",src)
    set(prefix.."act",1)
    zones[tonumber(z)] = tonumber(src)
    print ("SRC"..src)
  end)

  -- The same for zone turned off
  pamp:add([[#Z(%d),OFF]],function(z)
    set("NUVO."..z..".act",0)
  end)

  --
  -- HERE PASTE THE SOURCE CODE PRESENTED IN CHAPTER 6.
  --

  return dev
end
```

```
-- Initialization of the driver presented above.
-- Parameters are passed as objects of the two SG modules.
n = nuvo {name='N', tun=use 'SER.line.1', amp=use 'SER.line.2'}
```

A few words of explanation to the presented code:

Presented source code shows is a typical *design pattern* used in programming in Lua, called *creating an object*. The idea of this pattern is to hide all the variables defined **local** in the function body, allowing you to create multiple instace of the code (e.g. support several NuVo multiroom systems), or use parsers for other purposes.

After entering this code, you can proceed to create the visualization screen, where the RDS messages will be displayed:

1. Choose the **Display** tab.
2. Add a new **Screen**, enter the **ID** and choose the background image.
3. Add a **Text** element.
4. In the **Channel** field enter: `NUVO.line.<zone number>.<line number>.`

Example: `NUVO.line.2.3` - displays line No.3 of the zone No.2. The number of lines of information and the lines numbering vary depending on the radio station. NuVo multiroom system allows to display up to 6 line of information. In case of radio stations, the most frequently used are lines from 1 to 4.

5. Repeat steps 3 and 4 depending on the number of lines of information.

In the result you will get:

6. Controlling NuVo

In this chapter we describe how to control NuVo multiroom system using visualizations (**Display** touch panel, **Remote** app). The following functionality will be presented:

- On/off buttons for individual zones.
- Mute buttons for individual zones.
- Tuning radio tuners.
- Volume control.

- Master on/off button.
- Changing source for individual zones.

This functionality extends displaying RDS messages described in chapter 5. Controlling NuVo devices using visualizations is a two-step process. The first step is creating of logical rules. In the second step you will have to create visualization elements. Zones and signal sources numbering may vary depending on the connection of NuVo devices.

6.1. Creating of logical rules

To achieve presented functionality, you need to extend code from the **Logic** tab. Replace below comment in previously entered code:

```
-- HERE PASTE THE SOURCE CODE PRESENTED IN CHAPTER 6".
```

with this:

```
 -- This function turns ON individual zones
function dev:on(z)
  assert(z==1 or z==2 or z==3 or z==4 or z==5 or z==6,
    "Zone between 1 and 6")
  amp:send("*Z"..z.."ON")
end

--This function turns OFF individual zones
function dev:off(z)
  assert(z==1 or z==2 or z==3 or z==4 or z==5 or z==6 or z==nil,
    "Zone between 1 and 6 or nil")
  if z == nil then
    for i=1,6 do
      amp:send("*Z"..i.."OFF")
    end
  else
    amp:send("*Z"..z.."OFF")
  end
end

-- Increasing volume
function dev:volup(z)
  assert(z==1 or z==2 or z==3 or z==4 or z==5 or z==6,
    "Zone between 1 and 6")
  amp:send("*Z"..z.."VOL+")
end
```

```
-- Decreasing volume
function dev:voldown(z)
  assert(z==1 or z==2 or z==3 or z==4 or z==5 or z==6,
    "Zone between 1 and 6")
  amp:send("*Z"..z.."VOL-")
end

-- MUTE
function dev:mute(z)
  assert(z==1 or z==2 or z==3 or z==4 or z==5 or z==6,
    "Zone between 1 & 6")
  amp:send("*Z"..z.."MUTE")
end

-- Changing audio source
function dev:src(z)
  assert(z==1 or z==2 or z==3 or z==4 or z==5 or z==6,
    "Zone between 1 and 6")
  amp:send("*Z"..z.."SRC+")
end

-- Tuning radio tuner
local function tuner(ch,fun)
  assert(ch=='A' or ch=='B')
  tun:send("*T'"..ch.."'"..fun )
end

function dev:seek(ch,dir)
  assert(dir=="+" or dir=="-")
  tuner(ch,"SEEK"..dir)
 end
```

This source code can be modified and extended with new functions.

6.2. Creating visualizations

In order to visualize new functionality, use the screen created in chapter 5. All control buttons are universal and can be used to control individual zones. Just remember to change the zone number in the typed commands. The procedures presented in this chapter (except 6.2.5) control zone 1.

6.2.1. Turning zones On/Off

1. Add a new **OnOff** element.
2. In the **Channel** field type: `NUVO.1.act`.
3. In the **on** field enter: `LOGIC=n:on(1).`
4. In the **off** field enter: `LOGIC=n:off(1).`
5. In order to visualize the status of the zone in the form of light, add a new **Light** element. From the **Theme** list choose **Green**. In the **Channel** field type: `NUVO.1.act`. Place the element in the corner of the **OnOff** button.

6.2.2. Zone muting

1. Add a **Switch**.
2. In the **Label** type short description, for example *Mute*.
3. In Command field type: `LOGIC=n:mute(1).`

6.2.3. Tuner tuning

Tuning is done as a frequency scan from station to station.

1. Add a **Switch**.
2. In the **Label** type short description, for example *Seek+*.
3. In Command field type: `LOGIC=n:seek('A','+').`
4. Repeat steps from 1 to 3. Change **Label** description to *Seek-* and **Command** to: `LOGIC=n:seek('A','-').`

6.2.4. Volume control

1. Add a **Switch**.
2. In the **Label** type short description, for example *VOL+*.
3. In Command field type: `LOGIC=n:volup(1).`
4. Repeat steps from 1 to 3. Change **Label** description to *VOL-* and **Command** to: `LOGIC=n:voldown(1).`

Repeat subsections from 6.2.1 to 6.2.4 for the remaining zones.

6.2.5. Master on/off button

1. Add a **Switch**.
2. In the **Label** type short description, for example *ALL OFF*.
3. In Command field type: `LOGIC=n:off().`

The example of final result: