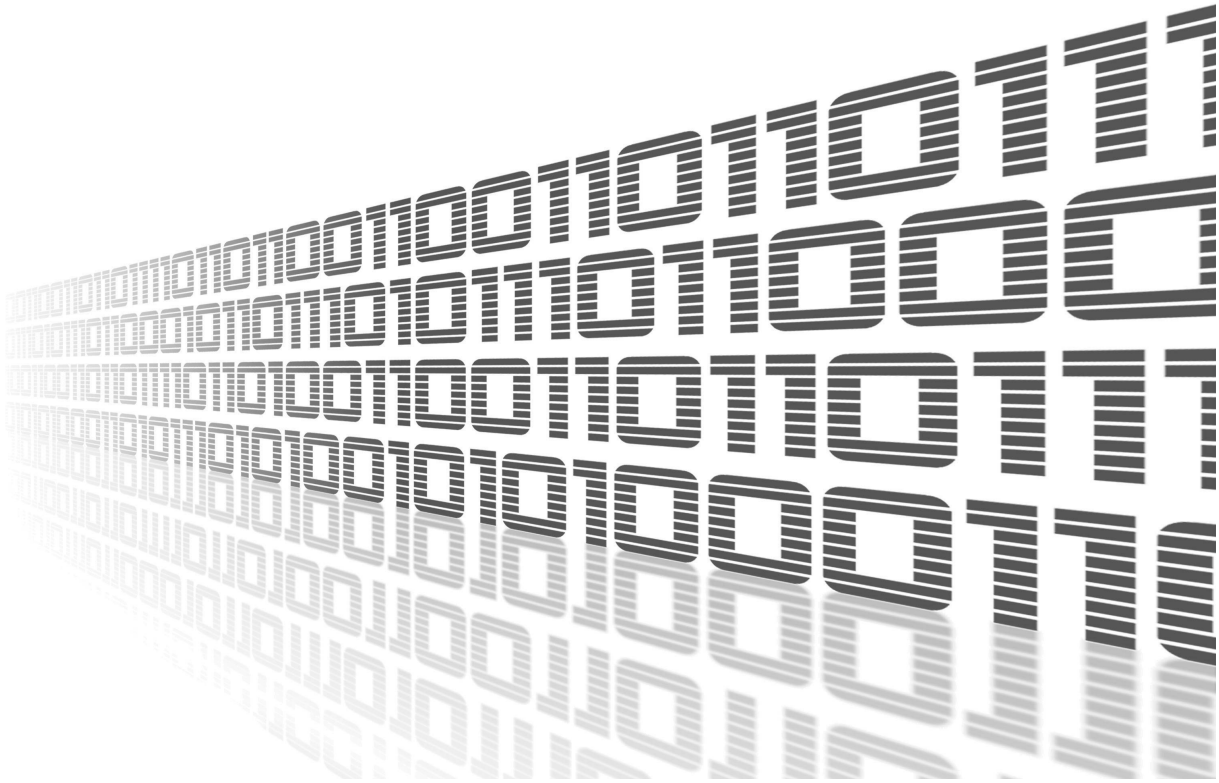
RouterApp

User Module

Node.js

APPLICATION NOTE



ADVANTECH

Used symbols

Danger – Information regarding user safety or potential damage to the router.

Attention – Problems that may arise in specific situations.

Information or notice – Useful tips or information of special interest.

Example – Example of function, command or script.

Advantech Czech s.r.o., Sokolska 71, 562 04 Usti nad Orlici, Czech Republic
Document No. APP-0080-EN, revised on September 13, 2022. Released in the Czech Republic.

Contents

1. Node.js User Module

User module *Node.js* is not contained in the standard router firmware. Uploading of this user module is described in the Configuration manual (see Chapter Related Documents). **This user module is only compatible with v3 and v4 platform routers!**

1.1 Web Interface

Once the installation of the module is complete, the module's GUI can be invoked by clicking the module name on the User modules page of router's web interface.

Left part of this GUI contains menu with General menu section. General menu section contains only the Licenses containing the list of all licenses for Node.js itself and also related Router Application and Return item, which switches back from the module's web page to the router's web configuration pages. The main menu of module's GUI is shown on Figure 2.

Figure 1: Menu

1.2 Introduction

The *Node.js* node is a proprietary server-side JavaScript runtime environment node available for Advantech cellular routers. This node is used by Advantech modules written in JavaScript, but can be used by any other third-party JavaScript application for routers administration and maintenance.

Router module contains this nodes addition to buil-in nodes:

- **node-authenticate-pam** – asynchronous PAM authentication for NodeJS,
- **when.js** – Promises/A+ and `when()` implementation, including a complete ES6 Promise shim,
- **router node** – a proprietary node for Advantech's cellular routers described in this document in detail.

1.3 Building the Custom Nodes

An official way how to build and install a node is using `npm` command. However, it is not possible to find it on our routers as the router is embedded device with limited resources and some nodes require complex building environment and high performance because of other languages than JavaScript.

Fortunately, it is easy to prepare a node on a PC with Linux and then copy it to the router. For more details see: https://icr.advantech.cz/products/software/user-modules#node-red in the chapter 4.5 of Node-RED Application Note.

2. Router Node

This part of the document is dedicated especially to programmers.

Router node (named "router") provides access to router specific functions and hardware. You can load the *Node.js* node in your code by require("router"), for example:

```
var r = require("router");
```

We will use the *r* variable from this example to access all the properties in the next examples in this notes.

Simple Example of Router Node Use

In the next figure is an example of loading the *Node.js* node.

```
# NODE_PATH=/usr/lib/node_modules/ node
> var r = require("router");
undefined
> console.log("Welcome to Advantech router " + r.productName);
Welcome to Advantech router ICR-323x
undefined
>
```

Figure 1: Loading the Node.js Node

2.1 Node Properties

2.1.1 productName

Read-only string variable loaded with router's product name. Example of usage:

```
console.log(r.productName);
```

Output: *SPECTRE-v3T-LTE*

2.1.2 productModel

Read-only string variable loaded with router's model indication. Example of usage:

```
console.log(r.productModel);
```

Output: *Output: ICR-3201W*

2.1.3 platformCode

Read-only string variable loaded with router's platform code. It is supported by routers of v3 and v4 production patform. Example of usage:

```
console.log(r.platformCode);
```

Output: V3

2.1.4 serialNumber

Read-only string variable loaded with router's serial number. Example of usage:

```
console.log(r.serialNumber);
```

Output: ACZ1100000322054

2.1.5 firmwareVersion

Read-only string variable loaded with router's firmware version. Example of usage:

```
console.log(r.firmwareVersion);
```

Output: 6.2.1 (2019-10-16)

2.1.6 RTCBatteryOK

Read-only boolean variable loaded with router's RTC battery state. True means OK, false means bad. Example of usage:

```
console.log(r.RTCBatteryOK);
```

Output: true

2.1.7 powerSupply

Read-only decimal number variable loaded with router's power supply voltage. Example of usage:

```
console.log(r.powerSupply + " V");
```

Output: 11.701 V

2.1.8 temperature

Read-only integer number variable loaded with router's internal temperature in Celsius degrees. Example of usage:

```
console.log(r.temperature + " °C");
```

Output: 39 °C

2.1.9 usrLED

Write-only boolean variable for control router's "USR" LED. Example of usage:

```
r.usrLED = true;
```

Sets USR LED to ON (lighting).

2.1.10 bIn

Read-only array with values on router's binary inputs. Array has the items related to number of the binary inputs. E.g. the router has BIN0 and BIN1 so array has valid indexes 0 and 1. The array items can have values 0 or 1. Example of usage:

```
console.log("The secondary binary input: " + r.bIn[1]);
```

Output: The secondary binary input: 0

2.1.11 bOut

Array related to router's binary outputs. It is similar as B_IN but you can also write values. Writen value change output state. Example of usage:

```
console.log(r.bOut[0]);
```

Output: 1

```
r.bOut[0] = 0;
```

Sets the first binary output to 0.

2.1.12 XBus

Object for working with X Bus. X Bus is a proprietary bus for communication between processes. E.g. you can subscribe informations which network interface go up/down or SMS from a mwan daemon. You can also send/subscribe your own topics between your applications.

```
XBus.publish(topic, payload, store=false)
```

Sends message with topic String and payload String to X Bus. Example of usage:

```
r.xBus.publish("watchdog/proc/myapp", "Timeout: 300");
```

Sends to the system watch request to watch your "myapp" application. The application must send this message regulary no later then period defined in the previous message (300 s in this example). Timeout 0 stops watching.

```
XBus.subscribe(topic, callback)
```

Subscribes to get messages with topic. Example of usage:

Function:

```
xbus.subscribe("status/mobile/mwan0", (msg) => {console.log(msg.payload);});
```

Asynchronous output:

```
Registration: Home Network
Technology: LTE
Signal-Strength: -88 dBm
Signal-Quality: -8 dB
```

```
XBus.unsubscribe(topic)
```

Unsubscribe from topic. Example of usage:

```
r.XBus.unsubscribe(id);
```

Stops receiving info abour registration to network from previous example.

```
XBus.list()
```

Lists stored messages. Example of usage:

```
r.XBus.list();
```

Output:

```
[ 'iface/ipv4/mwan0/config',
  'iface/ipv4/mwan0/running',
  'iface/ipv4/mwan1/config',
  'iface/ipv4/mwan1/running',
  'status/mobile/mwan0',
  'status/mobile/mwan1',
  'watchdog/proc/bard',
  'watchdog/proc/bard6',
  'watchdog/proc/mwan1d',
  'watchdog/proc/mwan2d',
  'watchdog/proc/mwanxd' ]
```

XBus.read(topic)

Read stored message from XBus. Example of usage:

```
r.XBus.read("iface/ipv4/mwan0/config");
```

Output:

```
Up: 1
Iface: usb0
Address: 10.184.131.221
Gateway: 192.168.253.254
DNS1: 217.77.165.211
DNS2: 217.77.165.81
```

2.1.13 configuration

Object containing the router configuration. User can read a configuration item by geting a object property and write a configuration item by setting a object property. The object keys are the same as configuration keys as in the setting files. It is possible to look for a requested key name in related setting file. The firmware configurations are placed in the `/etc/settings.*` files. Router App's configuration are placed in the `/opt/*/etc/settings` files. The Router Report (Web UI: Status / System Log / Save Report) contains a full list of the current configuration and may be it is the easest way how to find the requested configuration key.

If a given key does not exist a read value is undefined and a written value cause exception (in strict mode). It is not possible to add a new non-existing configuration item, only to modify an existing one. The all configuration values are treated as strings. If user need to work with a different type he must convert it himself. Node does not perform any value validation. The user is responsible for sending the correct values. Examples:

```
console.log("Current router AP SSID is " + r.configuration["WIFI_AP_SSID"]);
```

For WIFI_AP_SSID=ROUTER_AP in /etc/settings.wifi_ap (or rather in the SSID field in the WiFi • Access Point 1 form) output will be:

```
Current router AP SSID is ROUTER_AP
```

```
r.configuration["ETH_IPADDR"]="192.168.99.1"
```

Changes the IP addres on eth0 interface

NOTE: A new configuration is only written. If user wants it to apply to the running environment restarting the router or the related service is necessary. For example above it is possible to use the following shell command:

```
/etc/init.d/eth restart
```

3. Related Documents

[1] User Modules: **icr.advantech.cz/user-modules**
[2] JS Foundation: **https://nodered.org/**

You can obtain product-related documents on *Engineering Portal* at *icr.advantech.cz* address.

To get your router's *Quick Start Guide*, *User Manual*, *Configuration Manual*, or *Firmware* go to the *Router Models* page, find the required model, and switch to the *Manuals* or *Firmware* tab, respectively.

The *Router Apps* installation packages and manuals are available on the *Router Apps* page.

For the *Development Documents*, go to the *DevZone* page.