

Application Note

Extending Router Functionality



© 2024 Advantech Czech s.r.o. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photography, recording, or any information storage and retrieval system without written consent. Information in this manual is subject to change without notice, and it does not represent a commitment on the part of Advantech.

Advantech Czech s.r.o. shall not be liable for incidental or consequential damages resulting from the furnishing, performance, or use of this manual.

All brand names used in this manual are the registered trademarks of their respective owners. The use of trademarks or other designations in this publication is for reference purposes only and does not constitute an endorsement by the trademark holder.

Used symbols



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



Attention – Problems that can arise in specific situations.



Information – Useful tips or information of special interest.

Contents

1. Document Content	1
2. Scripts	2
2.1 HTTP POST Method for a Remote Execution	2
2.2 SMS Handling Using /var/scripts/sms	2
2.3 Send SMS to E-mail	2
2.4 SMS Command 1	3
2.5 SMS Command 2	3
2.6 Send Information Email 1	4
2.7 Send Information SMNP Trap 1	4
2.8 Send Information Email 2	4
2.9 Send Information SMNP Trap 2	5
2.10 Automatic Reboot	5
2.11 Switch Between WAN and PPP	6
2.12 How to Use an Unsupported Serial Converter Chip	6
3. Router Apps	7
3.1 Recommended Tools	7
3.2 SDKs and Cross Compiler Available	8
3.3 Directory Structure	8
3.3.1 Internal Archive Structure	9
3.3.2 Information Files in /etc Directory	10
3.3.3 Configuration Files in /etc Directory	11
3.3.4 Scripts in /etc Directory	11
3.3.5 Web Interface Files in /www Directory	14
3.4 Programming	14
3.4.1 Actions – Add, Update, Delete and Scripts Call Order	15
3.4.2 Hardware Interfaces	16
3.4.3 Firewall Integration	22
3.4.4 Libraries and Dependency	25
3.5 CPU and Toolchains	25
3.5.1 CPU Architecture	25
3.5.2 Crosscompilation – Toolchains and Flags	25
3.6 Constraints	26
3.7 Programming for S1 Products	27
3.7.1 Extending Read-Only Filesystem	27
3.7.2 Adding JavaScript and CSS to Web Administration	27
3.7.3 Changing System Configuration	27
4. Custom Firmware Compilation	28
4.1 Preparing Build Environment	28
4.2 Building Opensource Components	28
4.3 Installing Built Components	28
4.4 Important Note on Running Custom Firmware	28
5. Related Documents	30

List of Figures

1	Router Apps Programming Scheme	7
2	Block Diagram of v3 Routers	16
3	The Default Server in the NAT Configuration (for IPv4)	22

List of Tables

1	SDKs Available	8
2	Cross Compilers Available	8
3	io Options	17
4	LED Options	18
5	Sizes of /var/data Directory	18
6	Sizes of /opt Directory	18
7	RAM Memory Parameters	19
8	GPIO Driver Iocontrol Command Codes	20
9	GPIO Port Type Codes	21
10	GPIO Cellular Module Board Type Codes	21
11	CPU Architecture	25

1. Document Content

This manual introduces various methods for enhancing your router's basic capabilities using the firmware itself. Each Advantech router's functionality can be extended, allowing users to customize and optimize their routers for specific needs. The methods described in this document are organized based on their level of complexity, starting from the simplest and progressing to more advanced approaches, and can be divided into these three groups:

1. **Scripts** - A script is a sequence of commands that can be executed via the command line interface (CLI). This method allows users to automate tasks, manage router operations, and extend functionality without needing to modify the router's core software. Scripts are the easiest and most flexible way to add simple custom functions to your router.
2. **Router Apps** - Router apps are pre-compiled applications that can be uploaded to the router. These apps introduce new functionality that may not be present in the default firmware. They are ideal for users who require specific features or capabilities that are not available out-of-the-box but prefer a plug-and-play solution rather than custom development.
3. **Custom Firmware Compilation** - For those who require extensive customization, compiling your own firmware is the most powerful method. This approach enables a wide variety of advanced functions and deep integration into the router's software. However, it is the most complex method and requires knowledge of firmware development, toolchains, and potentially debugging and testing.

Each of these methods offers a different level of control and customization over your router, making it possible to extend its functionality to meet diverse requirements. Whether you're a novice looking to automate simple tasks or an experienced developer aiming to integrate custom solutions into the router, there is a method suited to your needs.

2. Scripts



Note that scripts are not supported by the S1 products. The only way to substitute it is to create a custom Router App. This way is described in the next chapter.

2.1 HTTP POST Method for a Remote Execution

CSRF (Cross-site Request Forgery) protection is implemented in all routers from ICR-OS version 5.3.0. To execute an action remotely, a script with *curl* command can be used. In the next chapter is example of script for sending of SMS remotely via router's web page by HTTP POST method. Visit <https://icr.advantech.com/support/faq> page for more examples.

2.2 SMS Handling Using /var/scripts/sms

Next three Chapters show examples of using "/var/scripts/sms" script. This script located in RAM of the router has to be created in Startup Script (using EOF, see examples below), so the file is on its place in the router even after reboot.

The script "/var/scripts/sms" is called by Mobile WAN connection daemon if there is active *Enable remote control via SMS* in *Configuration* → *Services* → *SMS* section. The script can be used for creation of advanced (your own) control SMS commands of the router. The Mobile WAN daemon passes on following parameters to the script:

- \$0 – name of script itself (in this case "sms") – not passed
- \$1 – can be "1" or "0" (true or false). The value true ("1") is returned if mobile phone number the SMS is received from is filled in the field *Phone Number X* on the SMS Web configuration page. Otherwise it is false ("0").
- \$2 – mobile phone number of the SMS sender
- \$3 to \$9 – words of SMS, separated by space (maximum of seven words)

2.3 Send SMS to E-mail

Send incoming SMS to the email.

Startup Script:

```
EMAIL=john.doe@email.com
cat > /var/scripts/sms << EOF
#!/bin/sh
/usr/bin/email -t \${EMAIL} -s "Received SMS from \${2}" -m "Authorized: \${1},
Text: \${3} \${4} \${5} \${6} \${7} \${8} \${9}"
EOF
```

2.4 SMS Command 1

Implementation of a new SMS command "IMPULSE", which activates binary output OUT0 for 5 seconds. SMS will be processed, if it comes from one of three numbers defined on the web interface or phone number +420123456789.

Startup Script:

```
PHONE=+420123456789
cat > /var/scripts/sms << EOF
#!/bin/sh
if [ "\$1" = "1" ] || [ "\$2" = "$PHONE" ]; then
if [ "\$3" = "IMPULSE" ]; then
/usr/bin/io set out0 1
sleep 5
/usr/bin/io set out0 0
fi
fi
EOF
```

2.5 SMS Command 2

This script implements a new SMS command "PPP", which sets item *Network type*, *Default SIM card* and *Backup SIM card*. PPP command has the following structure:

PPP <AUTO/GPRS/UMTS> <1/2>

The first parameter sets network type. If the second parameter equals 1, *Default SIM card* will be set to primary SIM card. If this parameter equals 2, *Default SIM card* will be set to secondary SIM card.

Startup Script:

```
cat > /var/scripts/sms << EOF
STARTUP=#!/bin/sh
if [ "\$1" = "1" ]; then
if [ "\$3" = "PPP" ]; then
if [ "\$4" = "AUTO" ]; then
sed -e "s/\(PPP_NETTYPE=\).*\/10/" -e "s/\(PPP_NETTYPE2=\).*\/10/" -i
/etc/settings.ppp
elif [ "\$4" = "GPRS" ]; then
sed -e "s/\(PPP_NETTYPE=\).*\/11/" -e "s/\(PPP_NETTYPE2=\).*\/11/" -i
/etc/settings.ppp
elif [ "\$4" = "UMTS" ]; then
sed -e "s/\(PPP_NETTYPE=\).*\/12/" -e "s/\(PPP_NETTYPE2=\).*\/12/" -i
/etc/settings.ppp
fi
if [ "\$5" = "1" ]; then
sed -e "s/\(PPP_DEFAULT_SIM=\).*\/11/" -e "s/\(PPP_BACKUP_SIM=\).*\/12/"
-i /etc/settings.ppp
elif [ "\$5" = "2" ]; then
```



```
sed -e "s/\(PPP_DEFAULT_SIM=\).*\/12/" -e "s/\(PPP_BACKUP_SIM=\).*\/11/"
-i /etc/settings.ppp
fi
reboot
fi
fi
EOF
```

2.6 Send Information Email 1

Send information email about establishing of PPP connection.

Up Script:

```
EMAIL=john.doe@email.com
/usr/bin/email -t $EMAIL -s "Router has established PPP connection.
IP address: $4"
```

2.7 Send Information SNMP Trap 1

Send information SNMP trap about establishing of PPP connection.

Up Script:

```
SNMP_MANAGER=192.168.1.2
/usr/bin/snmptrap -g 3 $SNMP_MANAGER
```

2.8 Send Information Email 2

Send information email about switch binary input BIN0.

Startup Script:

```
EMAIL=john.doe@email.com
MESSAGE="BIN0 is active"

while true
do
/usr/bin/io get bin0
VAL=$?
if [ "$VAL" != "$OLD" ]; then
[ "$VAL" = "0" ] && /usr/bin/email -t $EMAIL -s "$MESSAGE"
OLD=$VAL
fi
sleep 1
done
```

2.9 Send Information SNMP Trap 2

Send information SNMP trap about change state of binary input BIN0.

Startup Script:

```
SNMP_MANAGER=192.168.1.2

while true
do
/usr/bin/io get bin0
VAL=$?
if [ "$VAL" != "$OLD" ]; then
/usr/bin/snmptrap $SNMP_MANAGER 1.3.6.1.4.1.30140.2.3.1.0 u $VAL
OLD=$VAL
fi
sleep 1
done
```

2.10 Automatic Reboot

Automatic reboot at the definition time. (23:55)

Startup Script:

```
echo "55 23 * * * root /sbin/reboot" > /etc/crontab
service cron start
```

2.11 Switch Between WAN and PPP

Switching between WAN and PPP. PPP connection is active, if PING on the defined IP address does not pass through.

Startup Script:

```
WAN_PING=192.168.2.1
WAN_GATEWAY=192.168.2.1
WAN_DNS=192.168.2.1

. /etc/settings.eth

/sbin/route add $WAN_PING gw $WAN_GATEWAY
/sbin/iptables -t nat -A PREROUTING -i eth1 -j napt
/sbin/iptables -t nat -A POSTROUTING -o eth1 -p ! esp -j MASQUERADE

LAST=1
while true
do
ping -c 1 $WAN_PING
PING=$?
if [ $PING != $LAST ]; then
LAST=$PING
if [ $PING = 0 ]; then
/etc/init.d/ppp stop
sleep 3
/sbin/route add default gw $WAN_GATEWAY
echo "nameserver $WAN_DNS" > /etc/resolv.conf
/usr/sbin/conntrack -F
/etc/scripts/ip-up - - - $ETH2_IPADDR
else
/etc/scripts/ip-down - - - $ETH2_IPADDR
/usr/sbin/conntrack -F
/sbin/route del default gw $WAN_GATEWAY
/etc/init.d/ppp start
fi
fi
sleep 1
done
```

2.12 How to Use an Unsupported Serial Converter Chip

Unsupported serial converter chip can be added on the fly using:

```
echo <VID> <PID> >/sys/bus/usb-serial/drivers/ftdi_sio/new_id
```



Example Startup Script for VID 0403 and PID d921:

```
echo 0403 d921 >/sys/bus/usb-serial/drivers/ftdi_sio/new_id
```

3. Router Apps

Router App (formerly *User module*) can be used for special software applications in the Advantech routers. This is to customize the router and to add new features. This guide describes the programming of a Router App so it can work in the Advantech routers. The directory structure of a Router App, programming methods, and technical information are explained to make it easy when programming your own Router App.

The Linux OS is running in the Advantech routers. It is recommended to use the Linux OS for router apps development, but it is not required. You can use C, C++ or Python language to develop the Router App. See the section 3.2 below for SDKs and cross compilers available. The general structure, scripts and general rules used in all router apps development platforms are described in this guide.

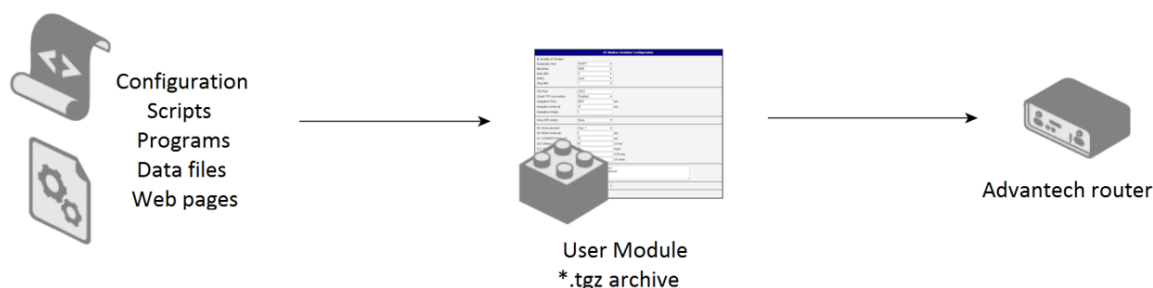


Figure 1: Router Apps Programming Scheme

3.1 Recommended Tools

- Cross compiler according to the platform used (see Chapter 3.2 below).
- Optionally SDK for easier development and usage (see Chapter 3.2 below).

3.2 SDKs and Cross Compiler Available

This chapter lists available SDKs and Cross Compilers that can be used for easier router apps development. The table below states the SDKs.

Language	Router Platform	SDK Download Link
C/C++	v2i, v3, v4, v4i	https://bitbucket.org/bbsmartworx/modulesdk
Python	v2i, v3, v4, v4i	https://bitbucket.org/bbsmartworx/modulesdk

Table 1: SDKs Available

Follow the *README* file in the SDK. It is recommended to download the SDK and look at the examples there when going through the following chapters.

Use the cross compiler mentioned in the table below to compile the SDK and an user module written in C/C++ language. Follow the *README* file instructions included in the compiler.

Language	Router Platform	Cross Compiler Download Link
C/C++	v2i, v3, v4, v4i	https://bitbucket.org/bbsmartworx/toolchains

Table 2: Cross Compilers Available

3.3 Router App Directory Structure

To upload the router app into the Advantech router you need a *.tgz* archive with a single directory in it (archive is packed using *tar* and then compressed using *gzip*) tool. The name of the *.tgz* archive and the directory in it has to be the same. This name can contain up to 24 characters of: 'a'-'z', 'A'-'Z', '0'-'9' and '_'. It is not recommended to use spaces in the names of subdirectories and files.

The <name> directory inside the archive can contain '/etc' subdirectory with the appropriate files in it – see the structure below and the following sections. There can be a '/www' subdirectory if there is a web interface of a Router App, '/bin' subdirectory and any other subdirectories and files you need. All subdirectories and files are optional, you can employ what you need in your Router App.

Router App archive name convention:

<name>.<platform>.tgz, e.g. 'mymodule.v3.tgz'.

Command for creating Router App archive:

```
tar -c --owner=0 --group=0 --mtime="2001-01-01 UTC" --exclude-vcs -C
$MODNAME | gzip -n > $MODNAME.$PLATFORM.tgz
```

3.3.1 Internal Archive Structure

The schema below illustrates the internal structure of the Router App archive.

<name>

—	/etc/	Subdirectory with scripts, information and configuration files.
	—	defaults Default configuration values.
	—	depends List of router apps this app depends on.
	—	init Initial script.
	—	install Script will run during installation process.
	—	ip-up Script executed when WAN connection is established (for IPv4).
	—	ip6-up Script executed when WAN connection is established (for IPv6).
	—	ip-down Script executed when WAN connection is lost (for IPv4).
	—	ip6-down Script executed when WAN connection is lost (for IPv6).
	—	name Human readable name used in the web interface.
	—	requires The lowest compatible version of router's firmware.
	—	settings Actual configuration file. Not in the *.tgz archive.
	—	uninstall Script will run during uninstallation process.
	—	version Version of the Router App showed in the web interface.
—	/bin/	Subdirectory with your auxiliary files, daemons or *.cgi scripts.
—	/www/	Subdirectory with web interface files.

File type legend:

Information files - see Chapter 3.3.2

Configuration files - see Chapter 3.3.3

Script files - see Chapter 3.3.4

3.3.2 Information Files in /etc Directory

depends

There is a list of dependencies (all router apps the app depends on) in this file. The format of the file is one Router App per line and the name of the Router App has to be same as the name of user app's directory <name> in the *.tgz archive.

File content example:



```
Python  
otherModuleName
```

name

This file contains the long human readable app name. It will be shown in the web interface of the router. Following characters are recommended to be used for Router App name: 'a'-'z', 'A'-'Z', '0'-'9' and ' '. If there is no 'name' file, the directory <name> is used instead.

File content example:



```
My Router App
```

requires

There is required minimal version of the router's firmware in this file. It has three numbers format of the router firmware versioning – MAJOR.MINOR.PATCH.

File content example:



```
5.2.0
```

version

The file with app version information. It will be shown in the web interface. The recommended format is the semantic versioning MAJOR.MINOR.PATCH and a date in YYYY-MM-DD format as shown below. If this file is missing, the version of the Router App will not be shown in the web interface of the router.

File content example:



```
1.0.0 (2015-07-15)
```

3.3.3 Configuration Files in /etc Directory


defaults

The default configuration parameters have to be saved in this file. These parameters are used during installation and when RST button on the router is pressed (back to factory defaults reset). The content of this file should be copied by 'init' script (see the next section) into the 'settings' file on install (see below) to enable the backup of configuration of the Router App. You do not need this file if the Router App has no configuration. Variables have to be defined this way:

```
MOD_<name>_<variable_name>=<value>,
```

where MOD stays for 'rotuter app' so it is recognizable when together with rest of the configuration parameters of the router, <name> is the name of the rotuter app (same as the the directory and archive name) and variable_name is the desired parameter name. Please use uppercase letters for the <name> and variable_name.

File content example:



```
MOD_MYMODULE_ENABLED=1
MOD_MYMODULE_PARAM1=0
MOD_MYMODULE_PARAM2=5
MOD_MYMODULE_PARAM3=20
```

settings

This file should not be in the *.tgz archive of the Router App. It should be created during installation by 'init' script. There should be a line copying the 'defaults' file into the 'settings' file in the 'init' script when installing the Router App. See 'init' script in the chapter 3.3.4.

The 'settings' file allows to make the backup of the configuration. It can be backed up together with router's configuration and it can remain on the Router App update.

When backing up the router's configuration, the 'settings' file is added to the router's configuration file and all the parameters are downloaded together in a single *.cfg file. When updating the Router App, the 'settings' file is backed up and the newer version of the Router App looks for the 'settings' file first. It goes back to the 'defaults' file only if there are some new parameters.

3.3.4 Scripts in /etc Directory

init

This is an initialization script. It is called with different parameters in different situations (start of the router, add, update, delete of the Router App, see the chapter 3.4.1). It can be called manually with the desired parameter, too. If there is no 'init' script, nothing happens and nothing is done on the Router App initialization in the given situations. These are the parameters of the script:

- **start** – The 'init' with the 'start' parameter is called automatically when starting the router or after the installation of the Router App.
- **stop** – The 'init' with the 'stop' parameter is called automatically before update or uninstalling the Router App.

- **restart** – The 'init' with the 'restart' parameter is not called automatically – it can be called manually only.
- **status** – The 'init' with the 'status' parameter is not called automatically – it can be called manually only. It is the status whether the Router App is running or not.
- **defaults** – The 'init' with the 'defaults' parameter is called automatically after installing the Router App or when the RST button is pressed. This is to copy the contents of 'defaults' file into the working configuration – 'settings' file.



An example of an 'init' script is shown below. There are just strings returned to inform what is going on in the example. Notice the copy 'cp' at the 'defaults' parameter to enable the backup of configuration. You can find this source code in the 'example1' of our *SDK* documentation.

```
#!/bin/sh

MODNAME=mymodule

case "$1" in
  start)
    echo "Starting module $MODNAME: done"
    exit 0
    ;;
  stop)
    echo "Stopping module $MODNAME: done"
    exit 0
    ;;
  restart)
    $0 stop
    $0 start
    ;;
  status)
    echo "Module $MODNAME is running"
    exit 0
    ;;
  defaults)
    cd /opt/$MODNAME/etc && cp defaults settings
    ;;
  *)
    echo "Usage: $0 {start|stop|restart|status|defaults}"
    exit 1
esac
```



install

This is an installation script. It is executed just after the uploading of the Router App into the router (files copied). See the next chapter [3.4.1](#) for more details on the order of scripts executed during the installation

process.

uninstall

This script is executed during the uninstallation process of the Router App. It is called just after stopping the Router App ('init stop') and just before deleting the files of the Router App. See the next chapter 3.4.1 for more details on the order of scripts executed during the uninstallation process.

ip-up

This script is executed when the WAN connection using IPv4 address is established. It works the same way as Up/Down Script in the router's web interface, but just for the particular Router App. This script is called with following parameters:

```
/opt/mymodule/etc/ip-up <ip-address-of-WAN-interface> <WAN-interface>
```

Below is the example of the script execution for internet connection established via Mobile WAN with IPv4 address 10.40.28.64.



```
/opt/mymodule/etc/ip-up 10.40.28.64 ppp0
```

ip6-up

This script is executed when the WAN connection using IPv6 address is established. This script is called with following parameters:

```
/opt/mymodule/etc/ip6-up <ip6-address-of-WAN-interface> <WAN-interface>
```

Below is the example of the script execution for internet connection established via Mobile WAN with IPv6 address fc00::a40:37.



```
/opt/mymodule/etc/ip6-up fc00::a40:37 ppp0
```

ip-down

This script is executed when the WAN connection using IPv4 address is lost. It is called with the same parameters as the previous 'ip-up' script:

```
/opt/mymodule/etc/ip-down <ip-address-of-WAN-interface> <WAN-interface>
```

Below is the example of the script execution for internet connection lost on Mobile WAN with IPv4 address 10.40.28.64.



```
/opt/mymodule/etc/ip-down 10.40.28.64 ppp0
```

ip6-down

This script is executed when the WAN connection using IPv6 address is lost. It is called with the same parameters as the previous 'ip6-up' script:

```
/opt/mymodule/etc/ip6-down <ip6-address-of-WAN-interface> <WAN-interface>
```

Below is the example of the script execution for internet connection lost on Mobile WAN with IPv6 address fc00::a40:37.



```
/opt/mymodule/etc/ip6-down fc00::a40:37 ppp0
```

3.3.5 Web Interface Files in /www Directory

This directory contains any .html, .cgi or other files of the web interface of the Router App. If there is file index.html, index.cgi etc., it is accessible in the router's web interface in the *Customization* section, *Router Apps*. If there is no 'www' folder, there is no link to the web interface of the Router App and if there is no 'index' file, there is no web interface to show up for the Router App. The directory is linked to this URL address of the router:

```
/opt/mymodule/www → http(s)://<router ip address>/module/mymodule
```



Regarding security you have 2 options – secured with the router's usernames and passwords or unsecured:

1. **Secured:** create a '.htpasswd' file in this 'www' directory with a symbolic link to the file '/etc/htpasswd' where the router's usernames and encrypted passwords are stored. This is the recommended option. Example of the '.htpasswd' file:



```
ln -s /etc/htpasswd .htpasswd
```

2. **Unsecured:** there is no '.htpasswd' file and anyone can access the web interface and files of the router app. It is strongly recommended not to use this option.

3.4 Programming Information

Useful information for programming of router apps can be found in this chapter. There is handling of Router App explained – adding, updating and deleting the Router App – what scripts are called in what order. Access to the hardware interfaces of the router is described. Important note on firewall integration, information on libraries and dependency are written out.

You can use lot of programs and commands already included in the router's operating system. See the *Command Line Interface* [8] Application Note for the documentation or press TAB key twice when connected to the console of the router (via SSH or Telnet). The list of possible commands will show up. You can write <command> --help for more information on that command.



3.4.1 Actions – Add, Update, Delete and Scripts Call Order

Generally you can put anything you need in the shell scripts, but please make sure that the actions executed can be finished within a few seconds. The order of scripts called on different actions is described below. If you want to see the log of scripts called (in the web interface System Log, for debug reason etc.), add this line at the beginning of each script. Here \$0 is a script itself and \$@ are its parameters.



```
/usr/bin/logger -t mymodule "DEBUG: $0 $@"
```

Router App Installation

Installation of the Router App is done by uploading the Router App into the router (*Customization* section). The *.tgz archive is extracted and the Router App directory is copied into the /opt directory of the router's file system. So the path to the Router App files is /opt/mymodule. After files are copied the scripts are called in the order below and with these parameters:

1. *Add or Update* button pressed – *.tgz archive uploaded, extracted and copied into the /opt directory.
2. /opt/mymodule/etc/install – script executed.
3. /opt/mymodule/etc/init defaults – script executed.
4. /opt/mymodule/etc/init start – script executed.

Router App Update

Update is done the same way as adding the Router App, but as the Router App has the same name, the previous running version is stopped first and the settings is backed up, too:

1. *Add or Update* button pressed.
2. /opt/mymodule/etc/init stop – script is executed if the name of the Router App is the same. The configuration file 'settings' is backed up. Then the old Router App files are deleted and the new *.tgz archive is uploaded, extracted and copied into the /opt directory.
3. /opt/mymodule/etc/install – script executed.
4. /opt/mymodule/etc/init defaults – script executed. Now when the 'settings' file is created from 'defaults', it is overwritten by 'settings' file from backup. If there are any new parameters, they are taken from 'defaults'.
5. /opt/mymodule/etc/init start – script executed.

Router App Uninstallation

Deleting of the Router App is done by pressing the *Delete* button next to the Router App you want to delete. These scripts are executed before deleting the files of the module:

1. *Delete* button pressed at the Router App.

2. `/opt/mymodule/etc/init stop` – script executed.
3. `/opt/mymodule/etc/uninstall` – script executed.
4. The whole Router App directory is removed from `/opt` directory of the router.

3.4.2 Hardware Interfaces

The access to the hardware interfaces is described in this chapter. You can use serial interface, all the network interfaces, binary inputs/outputs, user LED, MRAM or eMMC, storage space etc. in your Router App.

See the block diagram for v3 routers in Figure 2.

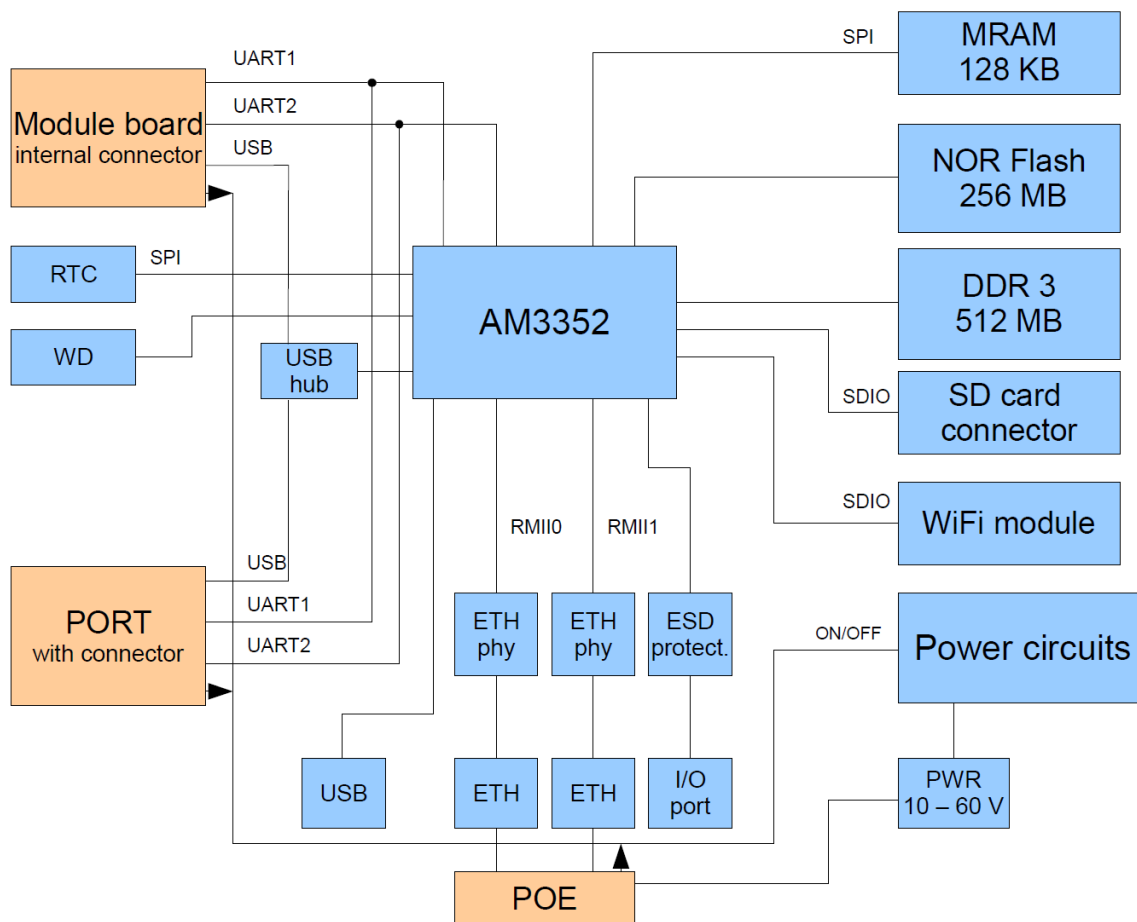


Figure 2: Block Diagram of v3 Routers

Serial Line Interface

This applies to the routers with serial line interface only. Access the serial line as a file since the Linux OS is running in the router. The path to the serial line file in the router's file system is `/dev/ttySn`, where `n` stands for the number of the UART serial port starting with 0 for the first one. Port mapping can vary based on the router platform and the PORT used.

Read the file to get the serial line input and write to this file to send data via serial line. Handle the files using appropriate locks: A Router App can be started as root, which means it can have full access to the system. Access to serial lines should be cared using file check and creation (locks) in directory `/var/lock`. A lock file has to be created in `/var/lock` before opening the serial line. The lock file name contains of 'LCK..' string and device name, e.g. for `/dev/ttyS0` the lock file will be `LCK..ttyS0`. Save the process identifier (PID) of the process running on an open device into this lock file. The PID format is 11 characters long – fill the spaces before the number and add end of the line. (E.g. for process 5634: space, space, space, space, space, space, 5, 6, 3, 4, end of line). The lock file has to be deleted when the work with the interface is finished. There is a function to handle this in our *SDK* library.

Ethernet and Network Interfaces

You can access Ethernet and other network interfaces as a standard Linux network interfaces. Use `ifconfig` command to see and configure the network interfaces in the router. Detailed description of the command can be found in the *Command Line Interface* [8] Application Note.

There can be additional network interfaces in the router, depending on the configuration and tunnels settings.

I/O Interface

You can use the `io` program to control binary outputs and to read binary inputs. It supports reading state of binary outputs and setting state of counters. See the User's Manual for your router for details on binary inputs/outputs. **Note:** Binary inputs/outputs have inverse logic.

Synopsis: `io [get <pin>] | [set <pin> <value>]`

Option	Description
<code>get</code>	Get the state of input
<code>set</code>	Set the state of output

Table 3: io Options



Examples:

```
io set out0 1  Set the state of binary output OUT0 to 1.
io get bin0    Get the state of digital input BIN0.
io get an1     Get the state of analog input AN1 on expansion port XC-CNT.
io get cnt1    Get the state of counter input CNT1 on expansion port XC-CNT.
```

User LED Interface

You can control the USR LED on the front panel of the router via the program `led`.

Synopsis: `led [on | off]`

Option	Description
<code>on</code>	User LED is on
<code>off</code>	User LED is off

Table 4: LED Options



Examples:

`led on` Turn on USR LED.

`led off` Turn off USR LED.

Non-volatile Memory

You can use and access the non-volatile memory, depending on the platform, see Table 5 and Table 6. This memory is accessible in the `/var/data` and `/opt` directories of the router's file system.

Parameter	v2i	v2i with eMMC	v3	v3 with eMMC	v4	v4i
Memory type	NOR	eMMC	MRAM	eMMC	eMMC	eMMC
File system	JFFS2	ext4	JFFS2	ext4	ext4	ext4
Partition size	2 MiB	512 MiB	128 KiB	512 MiB	512 MiB	474 MiB

Table 5: Sizes of `/var/data` Directory

Parameter	v2i	v2i with eMMC	v3	v3 with eMMC	v4	v4i
Memory type	NOR	eMMC	NOR	eMMC	eMMC	eMMC
File system	JFFS2	ext4	JFFS2	ext4	ext4	ext4
Partition size	12 MiB	814 MiB	128 MiB	838 MiB	838 MiB	2.16 GiB

Table 6: Sizes of `/opt` Directory



For the size of the MRAM equal to 128 KiB, it is recommended to use maximally 64 KiB by a Router App, because the router's operating system uses this memory, too.

Notices:

- You can fit more data into the JFFS2 file system, if the data can be compressed well.
- It is recommended to create the Router App `<name>` subdirectory in `/var/data`.
- The `/var/data/<name>` subdirectory is deleted automatically on Router App removal.
- Clean up of other files or subdirectories is up to the author of the Router App.

RAM

See Table 7 for information about RAM sizes for different router platforms. You can use the standard way of dynamic memory allocation (e.g. `malloc` function). Be careful regarding the memory usage – do not deplete all the memory for your Router App.

Parameter	v2i routers	v3 routers	v4 routers	v4i routers
RAM size	128 MB	512 MB	1 024 MB	1 024 MB

Table 7: RAM Memory Parameters

Storage Access – USB Flash and SD Card

Connecting the USB device or SD card works the standard way as in Linux OS. When you connect a USB Flash stick to the router, you can see it in the `/dev` directory. You can use see the details on detected devices using `dmesg` command.

- **USB Flash stick** will typically show up as `/dev/sda1`. You can mount it with the `mount` command. (E.g. `mount -t vfat /dev/sda1 /mnt`).
- Some USB to serial converters are supported. These will show up as `ttyUSB0`, `ttyUSB1` etc. devices.
- **SD Card** inserted in the SD card reader will on the router will show up as `/dev/mmcblk0p1`. You can mount it the standard way. (E.g. `mount -t vfat /dev/mmcblk0p1 /mnt`).

I/O Control – Lower Hardware API

You can use even lower hardware API – Unix I/O control (`ioctl`). This can have better performance in some cases, but it can be harder to implement, too. Here are GPIO driver `ioctl` command codes in the table below with the Shell program alternative or close feature if available.

Variable – Action	Code	Input	Output	Shell
UM_GPIO_GET_MO1_SIM Get index of SIM card in the first cellular module	0x80004202U	0	0 or 1	—
UM_GPIO_SET_LED_USR Set state of LED USR	0x40004203U	0 or 1	0	led
UM_GPIO_SET_OUT0 Set state of output OUT0	0x40004206U	0 or 1	0	io
UM_GPIO_GET_OUT0 Get state of output OUT0	0x80004206U	0	0 or 1	io
UM_GPIO_GET_BIN0 Get state of input BIN0	0x80004207U	0	0 or 1	io
UM_GPIO_GET_PORT1_TYPE Get type of expansion port 1	0x80004209U	0	Code in Table 9	status ports
UM_GPIO_GET_PORT1_OVRL Get information on port 1 MBUS overload	0x8000420AU	0	0 or 1	—
UM_GPIO_GET_PORT2_TYPE Get type of expansion port 2	0x8000420BU	0	Code in Table 9	status ports

To be continued on the next page

Continued from the previous page

Variable – Action	Code	Input	Output	Shell
UM_GPIO_GET_PORT2_OVRL Get information on port 2 MBUS overload	0x8000420CU	0	0 or 1	—
UM_GPIO_GET_MO1_TYPE Get type of the first module board	0x8000420EU	0	Code in Table 10	—
UM_GPIO_GET_TEMPERATURE Get internal temperature	0x80004211U	0	Integer number in Kelvin	status sys
UM_GPIO_GET_VOLTAGE Get supply voltage	0x80004212U	0	Integer number in milli-volts	status sys
UM_GPIO_SET_PORT1_SD Shutdown expansion port 1	0x40004214U	0 or 1	0	—
UM_GPIO_GET_PORT1_SD Get shutdown of expansion port 1	0x80004214U	0	0 or 1	—
UM_GPIO_SET_PORT2_SD Shutdown expansion port 1	0x40004215U	0 or 1	0	—
UM_GPIO_GET_PORT2_SD Get shutdown of expansion port 2	0x80004215U	0	0 or 1	—
UM_GPIO_GET_MO2_SIM Get index of SIM card in the second module	0x80004216U	0	0 or 1	—
UM_GPIO_GET_MO2_TYPE Get type of the second module board	0x80004219U	0	Code in Table 10	—
UM_GPIO_GET_MOD_IDX Get index of selected module	0x8000421AU	0	0 or 1	—
UM_GPIO_GET_BIN1 Get state of input BIN1	0x8000421BU	0	0 or 1	io

Table 8: GPIO Driver Iocontrol Command Codes

On GET codes – you get typically boolean output where 1 means "yes" and 0 means "no", except for binary inputs/outputs – these have reversed logic. If an error occurs, -1 value is returned. The codes and variables can be found in our SDK library, The variables should be named as mentioned in the table above for proper work. There are board types on output codes in the tables below:

Output Code	Variable – Port Type
0x00	UM_GPIO_PORT_TYPE_EMPTY
0x02	UM_GPIO_PORT_TYPE_RS232
0x03	UM_GPIO_PORT_TYPE_RS485
0x04	UM_GPIO_PORT_TYPE_MBUS
0x05	UM_GPIO_PORT_TYPE_CNT
0x08	UM_GPIO_PORT_TYPE_ETH
0x0A	UM_GPIO_PORT_TYPE_WMBUS
0x0B	UM_GPIO_PORT_TYPE_RS422
0x0F	UM_GPIO_PORT_TYPE_NONE
0x11	UM_GPIO_PORT_TYPE_WIFI
0x12	UM_GPIO_PORT_TYPE_SDCARD
0x13	UM_GPIO_PORT_TYPE_DUST
0x20	UM_GPIO_PORT_TYPE_SWITCH

Table 9: GPIO Port Type Codes

Output Code	Variable – Module Board Type
0x00	UM_GPIO_MODULE_TYPE_EES3
0x01	UM_GPIO_MODULE_TYPE_EU3
0x02	UM_GPIO_MODULE_TYPE_MCXXXX_VWM10
0x03	UM_GPIO_MODULE_TYPE_PHS8
0x04	UM_GPIO_MODULE_TYPE_MCXXXX
0x05	UM_GPIO_MODULE_TYPE_VWM10
0x06	UM_GPIO_MODULE_TYPE_GOB13K
0x0A	UM_GPIO_MODULE_TYPE_MCXXXX_MCXXXX
0x0F	UM_GPIO_MODULE_TYPE_NONE

Table 10: GPIO Cellular Module Board Type Codes

3.4.3 Firewall Integration

If you want to use a TCP or UDP server in your Router App (or generally any program listening on TCP or UDP port), read this chapter carefully – there are information on how your Router App should handle the firewall in the router.

There is `iptables` program integrated in the router. It is used for Firewall and NAT rules processing.

There is the *Send all remaining incoming packets to default server* item (see Figure 3) in the NAT configuration of the router (separately for IPv4 and IPv6). If enabled (and the IP address is filled in), it will apply the Firewall and NAT rules first and the rest of incoming packets are sent to the configured default server. It ignores the TCP/UDP port your Router App is listening on. Therefore the Router App should add the `iptables` rules for itself during the installation process and remove them on its uninstallation. The best way to do it is in the 'init' script.

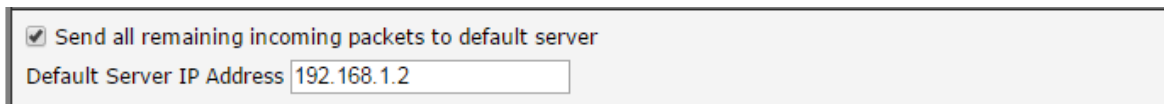


Figure 3: The Default Server in the NAT Configuration (for IPv4)

The example of the 'init' script adjusting the `iptables` rules is shown below. There are `add_chain()` and `del_chain()` functions and then the usual 'init' script continues with the `case` switch. Note that the script below is shortened, the rest of the parameters is skipped in this example.

The `iptables` rules are added in the `add_chain()` function so the Firewall can accept it and so the NAT will not send it to the default server. The `add_chain()` function is then called by the 'init start'. It has parameters e.g. `mod_my module tcp 1000` as you can see from the example below. Here 1000 is the TCP port number defined in the 'settings' file of the Router App. Now when the packet comes to TCP port 1000, it is accepted even if there is default server set in NAT configuration of the router.

The `del_chain()` function is called on 'init stop' likewise. It's parameter is `mod_my module` as you can see in the example below. This is to remove the `iptables` rules on the Router App removal (or restart, or manually on 'init stop').

```
MODNAME=mymodule
MODEXEC=mymoduled

add_chain() {
  /sbin/iptables -N $1 || return
  /sbin/iptables -A $1 -p $2 --dport $3 -j ACCEPT
  /sbin/iptables -A in_mod -j $1
  /sbin/iptables -t nat -N $1
  /sbin/iptables -t nat -A $1 -p $2 --dport $3 -j ACCEPT
  /sbin/iptables -t nat -A pre_mod -j $1
  if [ -f /sbin/ip6tables ]; then
    /sbin/ip6tables -N $1 || return
    /sbin/ip6tables -A $1 -p $2 --dport $3 -j ACCEPT
    /sbin/ip6tables -A in_mod -j $1
    /sbin/ip6tables -t nat -N $1
    /sbin/ip6tables -t nat -A $1 -p $2 --dport $3 -j ACCEPT
    /sbin/ip6tables -t nat -A pre_mod -j $1
  fi
}

del_chain() {
  /sbin/iptables -D in_mod -j $1
  /sbin/iptables -F $1
  /sbin/iptables -X $1
  /sbin/iptables -t nat -D pre_mod -j $1
  /sbin/iptables -t nat -F $1
  /sbin/iptables -t nat -X $1
  if [ -f /sbin/ip6tables ]; then
    /sbin/ip6tables -D in_mod -j $1
    /sbin/ip6tables -F $1
    /sbin/ip6tables -X $1
    /sbin/ip6tables -t nat -D pre_mod -j $1
    /sbin/ip6tables -t nat -F $1
    /sbin/ip6tables -t nat -X $1
  fi
}
```

continue on next page

continued from previous page

```

case "$1" in
start)
    echo -n "Starting module $MODNAME: "
    . /opt/$MODNAME/etc/settings
    [ "$MOD_EXAMPLE5_ENABLED" != "1" ] && echo "skipped" && exit 0
    add_chain mod_$MODNAME tcp $MOD_MYMODULE_PORT 2> /dev/null
    /opt/$MODNAME/bin/$MODEXEC &
    RETVAL=$?
    [ $RETVAL = 0 ] && echo "done" || echo "failed"
    exit $RETVAL
    ;;
stop)
    echo -n "Stopping module $MODNAME: "
    killall $MODEXEC 2> /dev/null
    del_chain mod_$MODNAME 2> /dev/null
    RETVAL=$?
    [ $RETVAL = 0 ] && echo "done" || echo "failed"
    exit $RETVAL
    ;;
*)
    echo "Usage: $0 {start|stop|restart|status|defaults}"
    exit 1
esac

```

There are `in_mod` and `pre_mod` parameters in iptables rules in functions `add_chain()` and `del_chain()`. Here is the iptables structure used in the router so you know when `in_mod` and `pre_mod` rules are applied. Note that there is many more rules nested in the structure, but only the ones applicable for router apps are shown in the structure below:

- mangle PREROUTING
- nat PREROUTING
 - pre (WAN interfaces only)
 - pre_mod - ACCEPT rules for installed router apps
 - mod_...
 - mod_...
 - mod_...
- nat POSTROUTING
- filter INPUT
 - in
 - in_mod - ACCEPT rules for installed router apps
 - mod_...
 - mod_...
 - mod_...
- filter FORWARD

3.4.4 Libraries and Dependency

To maintain the proper work of the Router App after the router's firmware update, observe these two recommendations for libraries and dependencies:

- Do not link the libraries dynamically. Use the static link with your Router App only.
- Do not use libraries from the file system of the router, except for *glibc* library.

The reason is that the libraries in the router's firmware can change and vary in the updated firmware versions. The Router App should be independent on the libraries of the router's firmware so it can work properly after the firmware update.

If you write your Router App in the C language – you can use *glibc* library from the router's file system (located in `/lib` directory in the router). Only use the functions up to the 2.0.6 version from *glibc* library. This is to maintain the compatibility within all firmware versions since there is *glibc 2.0.6* library in all versions of the router's firmware.

3.5 CPU and Toolchains

3.5.1 CPU Architecture

There are CPU parameters for different router platforms listed in the Table 11.

Parameter	v2i routers	v3 routers	v4 routers	v4i routers
CPU	SAM9X60	AM3352	ARMv8-A	ARMv8-A
Architecture	arm 5TEJ	arm v7	arm v8	arm v8
Core	ARM926EJ-S	Cortex-A8	Cortex-A72	Cortex-A53
CPU power	660 DMIPS	2000 DMIPS	4.7 DMIP-S/MHz	2.3 DMIPS/MHz

Table 11: CPU Architecture

3.5.2 Crosscompilation – Toolchains and Flags

This is applicable if you are crosscompiling the Router App written in C or C++. It is recommended to download and use toolchains offered in chapter 3.2.

You can use other crosscompiler, too. Use these flags for successful crosscompilation, based on the router's platform:

v2i routers – flags for crosscompilation for v2i routers:

```
-march = armv5te
-mtune = arm926ej-s
-mfloat-abi = soft
```

v3 routers – flags for crosscompilation for v3 routers:

```
-march = armv7-a
```

```
-mtune = cortex-a8  
-mfpv = vfpv3  
-mfloat-abi = softfp
```

v4 routers – flags for crosscompilation for v4 routers:

```
-march=armv8-a+crc+crypto  
-mtune=cortex-a72
```

v4i routers – flags for crosscompilation for v4i routers:

```
-march=armv8-a+crc+crypto  
-mtune=cortex-a53
```

3.6 Constraints

- The space in '/opt' directory, where router apps are stored, is limited, see [Table 6](#) for the size of different platforms.
- For a platform having the /var/data folder of 128 KiB size, use only 64 KiB for the smooth run of Router App and the router's firmware. See [chapter 3.4.2](#) for more details.
- You can load more data into the '/opt' directory if compressed (the amount of data depends on the data itself – how well it can be compressed). The '/opt' directory is not erased during the router's firmware update.

3.7 Programming for S1 Products

3.7.1 Extending Read-Only Filesystem

The root filesystem on S1 Products is read only and cannot be changed. It can however be extended using Router Apps that are installed as an read-only overlay over /opt and /usr.

Other partitions cannot contain data only. The persistent /var and volatile /run and /tmp are mounted as „noexec“ without an executable permissions. Executable files can be added only as Router Apps.

The example8 and example9 in the ModulesSDK demonstrate how to add new files to the /usr. Build the example using:

```
cd modules/example8
make PLATFORM=RBv2i-S1
```

This creates a .raw file, which can be installed as a standard Router App.

3.7.2 Adding JavaScript and CSS to Web Administration

The web server uses a strict Content-Security-Policy [<https://content-security-policy.com/>]. JavaScript and CSS styles must be inside a <style> tag with a nonce value.

The ModulesSDK version 2.1.1 provides um_html_js and um_html_css functions that can be used to correctly include JavaScript and CSS styles. For example:

```
um_html_js("document.f.username.focus();\n");
```

3.7.3 Changing System Configuration

Each configuration file has an integrity check. To change configuration from a script, create a partial backup file (e.g. backup.cfg) and then execute:

```
openssl sha512 backup.cfg | awk '{print "INTEGRITY=CFG_HASH=\"$2} ' >> backup.cfg
sudo restore backup.cfg
```


4. Custom Firmware Compilation

4.1 Preparing Build Environment

You will need parts of the build environment whether you're building a Router App (User Module) or any open-source component of ICR-OS. For a proper start, you need a 64-bit Linux-based OS or a virtual machine running it.

The instructions are tested on Ubuntu and Debian. For rpm-based distributions, adjust the commands for your package manager. Fedora is also a fine choice. If you're using another distribution, you're likely skilled enough to adapt.

Development resources are published on [Bitbucket](#). The first step is obtaining a cross compiler to produce binaries for the router, described in chapter [3.2](#).

4.2 Building Opensource Components

If you want to (re-)build an open-source component of ICR-OS, fetch the matching source code for your firmware version from the [source code page](#). To find your ICR-OS version, check *System Information* on the router's default page under *General* in the left menu. The firmware version follows the format: major.minor.patch (e.g., 6.1.5). The corresponding source file would be firmware-6.1.5-src.tar.

Further instructions are in the README file in the archive. All patches to open-source components are included as stand-alone patches for easy identification.

You can build a complete firmware image, but the device will refuse unsigned firmware. However, replacing open-source components, upgrading tools, or libraries is possible via SSH as root. Be cautious: replacing critical files, like the Linux kernel, can cause the device to stop booting, with no easy fix.

These actions are your responsibility, and running custom firmware comes with risks. Please read the note on custom firmware before proceeding.

4.3 Installing Built Components

After following instructions the newly created files should be copied to the filesystem and overwrite original ones.

For the S1 Products the router app needs to be created according to chapter [3.7.1](#).

4.4 Important Note on Running Custom Firmware

- Small changes to open-source utilities can have unforeseen impacts on the system, potentially rendering features unusable or causing router malfunctions.
- Advantech CZ R&D is not responsible for effects caused by custom firmware. Changes may invalidate certifications and guarantees associated with the device.

- The product consists of multiple layers (HW, SW, certifications), and altering any layer creates something different from the original.
- Custom firmware without proprietary components will lack features and may not fully support all device hardware.
- Running custom firmware voids the device warranty.
 - Certifications printed on the labels and the Declaration of Conformity are no longer valid. Stickers for telecommunication certifications must be removed, as custom firmware may override hardware limits.
 - Incorrect hardware handling (e.g., wrong power-down sequence) can reduce device lifespan and damage hardware.
 - If distributing the device, remove the Advantech label, as it no longer reflects the original manufacturer. Contact business representatives for branding with your own OS.
- Custom firmware is unsupported by Advantech Czech s.r.o. If the device becomes unbootable (e.g., bootloader failure or MRAM data corruption), it may be unusable or un-updatable to ICR-OS. Repairing such a device will incur a support fee, as it is out of warranty.
- Uploading custom firmware to a device running ICR-OS will trigger a warning due to the lack of a digital signature. You can proceed at your own risk, but this voids the warranty. Note, digital signatures apply only to v3 generation devices.

5. Related Documents

You can obtain product-related documents on the **Engineering Portal** at icr.advantech.com.

To access your router's documents or firmware, go to the [Router Models](#) page, locate the required model, and select the appropriate tab below.

Documents that are common to all models and describe specific functionality areas are available on the [Application Notes](#) page.

The **Router Apps** installation packages and manuals are available on the [Router Apps](#) page.

If you are interested in further options for extending router functionality, either through scripts or custom Router Apps, please see the information available on the [Development](#) page.