Creating OWL User Modules

Contents

Changelog

2020-05-19 – Initial document release

THE NAMING OF COPYRIGHTED TRADEMARKS IN THIS MANUAL, EVEN WHEN NOT SPECIALLY INDICATED, SHOULD NOT BE TAKEN TO MEAN THAT THESE NAMES MAY BE CONSIDERED AS FREE IN THE SENSE OF THE TRADEMARK AND TRADENAME PROTECTION LAW AND HENCE THAT THEY MAY BE FREELY USED BY ANYONE.

HIRSCHMANN AUTOMATION AND CONTROL GMBH
STUTTGARTER STR. 45-51
72654 NECKARTENZLINGEN
GERMANY
TEL.: +49 1805 141538

Overview

User modules can be used for a special software applications in Hirschmann OWL routers. This is to customize the router and to add new features. This guide describes

- *programming of a user module so it can work in the OWL routers,*
- *directory structure of a user module,*
- *programming methods and*
- *technical information*

These are explained to make it easy when programming your own user module.

The OWL routers are running an embedded operating system with a Busybox environment as well as a Linux kernel. It is recommended to use a Linux OS for user modules development, but it is not required. You can use C, C++, Python, Lua or Nodejs language to develop the user modules. See the section below for SDKs and cross compilers available. The general structure, scripts and general rules used in all user module development platforms are described in this guide.

SDKs and Cross Compiler Availability

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

- *C/C++*
  https://hirschmann-support.belden.com/downloads/files/owl-gcc-toolchain
- *Python 3*
  https://hirschmann-support.belden.com/kb/articles/application-note-owl-user-module-python3
- *NodeJS*
  https://hirschmann-support.belden.com/kb/articles/application-note-owl-user-module-nodejs
- *NodeRED*
  https://hirschmann-support.belden.com/kb/articles/application-note-owl-user-module-node-red
- *Lua*
  https://hirschmann-support.belden.com/kb/articles/application-note-owl-user-module-lua

The SDKs themselves come with a description document of how to get started. The guide that follows is intended to explain how to package it into the user module format for easy deployment.

User Module Directory Structure

To upload the User Module into the router you need a `*.tgz` *archive with 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 user module, '/bin' subdirectory and any other subdirectories and files you need. All subdirectories and files are optional, you can employ what you need in your user module.

User module archive name convention:

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

Command for creating user module archive:

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

Internal archive structure

The schema below illustrates the internal structure of the user module archive.

<name>
|
|— /etc/ Subdirectory with scripts, information and configuration files.
| |
| |— defaults Default configuration values.
| |— depends List of user modules this user module 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 user module 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
Configuration files
Script files

Information Files in /etc Directory

depends

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

File content example:

```
python3

lua

nodejs
```

name

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

File content example:

```
My  User  Module
```

requires

List the 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:

```
6.2.3
```

version

The file with module 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 user module will not be shown in the web interface of the router.

File content example:

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

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 the `init` script (see the next section) into the `settings` file on install (see below) to enable the backup of configuration of the user module. You do not need this file if the user module has no configuration. Variables have to be defined this way:

```
MOD_<name>_<variable name>=<value>
```

- *where MOD stands for* **user module** *so it is recognizable when combined together with rest of the configuration parameters of the router, for example in the report.*
- `<name>` *is the name of the user module (same as the the directory and archive name) and*
- `<variable name>` *is the desired parameter name.*
- `<value>` *is the value of the configuration variable. Values containing spaces have to be escaped by double quotes.*

File content example:

```
MOD_MYMODULE_ENABLED=1

MOD_MYMODULE_PARAM1=Examplestring

MOD_MYMODULE_PARAM2=”Hello Hirschmann”
```

settings

This file should not be in the *.tgz archive of the user module. It should be created during installation by `init` script. You should write a line copying the ’`defaults`’ file into the ’`settings`’ file in the `init` script when installing the user module. The `’settings’` file allows to make a backup of the configuration. It will be automatically backed up together with router’s configuration and it remains during an update of the user module.
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 user module, the `’settings’` file is backed up and the newer version of the user module looks for the `’settings’` file first. It goes back to the ’defaults’ file only if there are some new parameters.

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 user

module). It can be called manually with the desired parameter, too. If there is no 'init' script, nothing happens and nothing is done on the user module initialization. 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 user module.*
- *stop - The 'init' with the 'stop' parameter is called automatically before update or uninstalling the module.*
- *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 user module is running or not.
- defaults - The 'init' with the 'defaults' parameter is called automatically after installing the module or when the RST button is pressed. The intention 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.

```
#!/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 user module into the router (files copied).

uninstall

This script is executed during the uninstallation process of the user module. It is called just after stopping the user module ('init stop') and just before deleting the files of the user module.

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 user module. 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
```

Web Interface Files in /www Directory

This directory contains any .html, .cgi or other files of the web interface of the user module. If there is file index.html, index.cgi etc., it is accessible in the router's web interface in the Customization section, User Modules. If there is no 'www' folder, there is no link to the web interface of the user module and if there is no 'index' file, there is no web interface to show up for the user module. 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 user database or unsecured:

- *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
```

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

Programming Information

The handling of user modules is explained - adding, updating and deleting the user module – and which scripts are called in which order. Access to the hardware interfaces of the router is described. Important note on firewall integration, information on libraries and dependency, and older firmware compatibility topics are noted.

You can use all of the programs and commands already included in the router's operating system.

See the "User Guide – OWL Commands and Scripts" [1] *documentation or press TAB key twice when connected to the console of the router (via SSH or Telnet) for auto completion.*

The list of possible commands will show up. You can write `<command> --help` *for more information on that command.*

Actions - Add, Update, Delete and Scripts Call Order

Generally you can put anything you need in the shell scripts. 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  \$@"
```

Add User Module - Install

*Installation of the user module is done by uploading the user module into the router (Customization section). The *.tgz archive is extracted and the user module directory is copied into the* `/opt` *directory of the router's file system. So the path to the user module 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.*

[1] https://hirschmann-support.belden.com/kb/articles/user-guide-owl-commands-and-scripts

Update User Module

Update is done the same way as adding the user module, but as the user module 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 user module is the same. The configuration file 'settings' is backed up. Then the old user module 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.*

Delete User Module - Uninstall

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

1. *Delete button pressed at the user module.*
2. `/opt/mymodule/etc/init stop` *- script executed.*
3. `/opt/mymodule/etc/uninstall` *- script executed.*
4. *The whole user module directory is removed from /opt directory of the router.*

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, RAM, storage space etc. in your user module.

Serial Line Interface

The path to the serial line file in the router's file system: `/dev/ttyS0` *or* `/dev/ttyS1`

It is usually /dev/ttyS0 for OWL routers. If you have a version with two serial interfaces (e.g. RS232 and RS485), both ttyS0 and ttyS1 are used. Generally both UART1 and UART2 are connected.

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 user module 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.

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 "User Guide – OWL Commands and Scripts"* [2].

These are some important physical interfaces of the router:

- *eth0: Ethernet 0*
- *eth1: Ethernet 1*
- *pppX/usbX: Mobile WAN connection (cellular module board connection). It is typically ppp0 or usb0, depending on the model of the router. usb0 is the first module on the routers with two modules, usb1 is the second. pppX interface numbering varies.*
- *wlan0 WiFi connection on WLAN variants of the router.*

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

- *get: Get the state of input*
- *set: Set the state of output*

Examples:

[2] https://hirschmann-support.belden.com/kb/articles/user-guide-owl-commands-and-scripts

`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

- *on:* *User LED is on*
- *off:* *User LED is off*

Examples:

`led on` *Turn on USR LED*
`led off` *Turn off USR LED*

User module data

The device provides user data storage. The storage size is different for each model. Please check the datasheet for details. The user module storage is accessible in the `/var/data` *directory of the router's file system. The operating system uses this storage space too, which is why it should not be filled to the maximum by user modules.*

It is recommended to create the user module `<name>` *subdirectory in* `/var/data`*. The* `/var/data/<name>` *subdirectory is deleted automatically on user module removal.*

Cleanup of other files or subdirectories is up to the author of the user module.

RAM Size

The size of the RAM depends on the variant. Please refer to the datasheet for details. 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 user module.

The `/var` *folder is for example created in the RAM on some variants of the router:*

```
# mount

none on /var type ramfs (rw,noatime)
```

Storage Access - USB Flash and SD Card

Connecting the USB device or SD card works the standard way as in most Linux OS. When you connect a USB Flash stick to the router, you can see it in the /dev directory. You can 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 on the router will show up as* `/dev/mmcblk0p1`*. You can mount it the standard way. (E.g.* `mount -t vfat /dev/mmcblk0p1 /mnt`*)*

Firewall Integration

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

The router utilizes the `iptables` *program for Firewall and NAT rules processing.*

There is a Send all remaining incoming packets to default server configuration option 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 user module is listening on.

Therefore the user module 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. 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 are 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 '*`init start`*'. It has parameters e.g.* `mod_mymodule tcp 1000` *as you can see from the example below. Here 1000 is the TCP port number defined in the 'settings' file of the user module. 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. Its parameter is* `mod_mymodule` *as you can see in the example below. This is to remove the iptables rules on the user module 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

}

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 are many more rules nested in the structure, but only the ones applicable for user modules are shown in the structure below:*

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

Libraries and Dependency

To maintain the proper work of the user module 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 user module 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 user module should be independent on the libraries of the router's firmware so it can work properly after the firmware update.

If you write your user module 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.

Older Firmware Compatibility

User modules are supported since firmware 2.1.2 in the router. If you want to keep your user module compatible with all versions of the firmware, use only the functions from glibc 2.0.6 library or lower. If you do not use glibc functions at all, there will be no compatibility issues with the user module. The libstdc++ library is a part of firmware since 5.1.0 and higher.

CPU and Toolchains

CPU and Memory

	OWL 4G, OWL LPWAN, OWL LTE M12
CPU	*AM3352*
Architecture	*arm v7*
Core	*ARM ® Cortex ®-A8*
CPU power	*2000 DMIPS*
RAM	*512 MB*

Crosscompilation - Toolchains and Flags

This is applicable if you are crosscompiling the user module written in C or C++. It is recommended to download and use toolchains offered in page 4. You can use other crosscompilers too. Use these flags for successful cross compilation:

```
-march=armv7-a
-mtune=cortex-a8
-mfpu=vfpv3
-mfloat-abi=softfp
```