Dusan Holub
October 2017

B+B SmartWorx
Powered by ADVANTECH

B+B SmartWorx
User Modules Development

www.advantech-bb.com

Agenda

- Purpose and prerequisites
- Environment setup
- Do I need a User Module?
- User Modules' run environment
- User Modules' build environment
- An API overview
- System utilities
- Make a UM step-by step
- Troubleshooting
- Notes and resources

B+B SmartWorx Powered by Advantech

Colors explanation

Highlighting of what we are referring to.

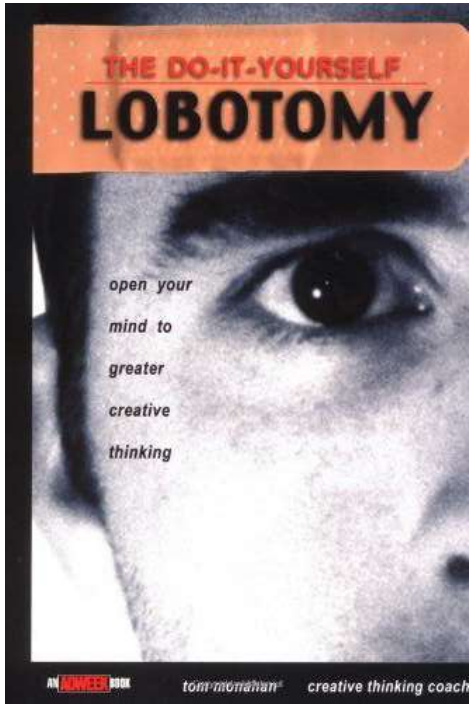
Side notes are for better understanding or readability.

You are supposed to modify only these parts of code, but feel free to modify others if you know what are you doing.

You should always pay attention to this warning.

Purpose of this training is to:

- Help decide if UM or script is needed
- Give an overview of UM run environment and how it is handled by OS
- Show a guideline for user modules development in C for V2 and V3 platforms
- Suggest “do’s” and “don’ts”
- Reccomend the most effective way how to develop your own UM from existing example
- Help with troubleshooting



Prerequisites - Environment

- **Hardware prerequisites**
  - Up to 1 GB of free space in `/opt`
  - Internet access for SDK installation

```
make[3]: Entering directory '/home/dusan/src/ModulesSDK/modules/template/source'
  CC  module_cgi.o
  LD  module_cgi
make[3]: Leaving directory '/home/dusan/src/ModulesSDK/modules/template/source'
make[3]: Entering directory '/home/dusan/src/ModulesSDK/modules/template/source'
make[3]: Leaving directory '/home/dusan/src/ModulesSDK/modules/template/source'
make[2]: Leaving directory '/home/dusan/src/ModulesSDK/modules/template'
make[1]: Leaving directory '/home/dusan/src/ModulesSDK/modules'
dusan@dusan-VirtualBox:~/src/ModulesSDK
$
```

- **Software prerequisites**
  - Decide if you use prepared virtual machine image or install into your existing Linux installation
  - Modern 64 bit Linux (xubuntu-16.04.3-desktop-amd64.iso is used here)
  - Your favorite IDE or a text editor and a console terminal
- **Other**
  - Cca 2 hours for installation (if ready made virtual appliance is not used)

Prerequisites - Know-How

- Understanding of technology you are going to write a module for
- Advantech routers:
  - Configuration and administration
- Linux administration:
  - Familiar with command line
  - Administration of installed packages
  - Networking configuration
- Programming for / on Linux
  - Bash scripting (if needed)
  - C coding
  - Makefile understanding

Environment setup - DIY way 1/2

```
# Sanity check
sudo apt-get update && sudo apt-get upgrade && sudo apt-get autoremove

# GNU C compiler suite, host kernel headers for Virtualbox guest add-ons
sudo apt-get install build-essential linux-headers-$(uname -r)

# for virtual machine only: now install optional guest add-ons and reboot

# Install toolchain prerequisites
sudo apt-get install gawk wget git dpkg rpm texinfo
sudo apt-get install libgmp-dev libmpfr-dev libmpc-dev

# Optional but useful
sudo apt-get install autoconf automake bison cmake flex libtool m4 pkg-config scons
sudo apt-get install mc vim-gtk
```

Environment setup - DIY way 2/2 option A - install

```
# Toolchains installation scripts - see https://bitbucket.org/bbsmartworx/toolchains
# takes > 1 hour
cd ~ && mkdir src && cd src
git clone https://bitbucket.org/bbsmartworx/toolchains.git
cd toolchains && sudo dpkg -i deb/*.deb
# in case of problems, remove /opt/toolchain, resolve issue and make again
```

rpm packages are also available for Fedora etc.

```
# User modules SDK - see https://bitbucket.org/bbsmartworx/ModulesSDK
cd ~/src
git clone https://bitbucket.org/bbsmartworx/modulessdk.git
cd ~/src/ModulesSDK && make
# took few seconds to build them all
```

Environment setup - DIY way 2/2 option B - compile

```
# Toolchain builder - see https://bitbucket.org/bbsmartworx/TCBuilder
# took cca 1 hrs 30 mins on i5, SSD, Virtualbox with 2 CPUS, 4 GiB RAM
cd ~ && mkdir src && cd src
git clone https://bitbucket.org/bbsmartworx/TCBuilder.git
cd TCBuilder && sudo make
# in case of problems, remove /opt/toolchain, resolve issue and make clean && make again


# User modules SDK - see https://bitbucket.org/bbsmartworx/ModulesSDK
cd ~/src
git clone https://bitbucket.org/bbsmartworx/ModulesSDK.git
cd ~/src/ModulesSDK && make
# took few seconds to build them all
```

Environment setup - lazy way 1/3

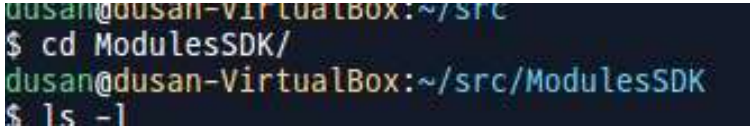
Using the UM development environment, already installed on a virtual machine

- Install Virtualbox
- Import UMDevelopment_1.3.ova
  - Username: **um**
  - Password: **123456**
- All SDKs and libraries are already installed
- Customize the environment
  - Install and configure your favorite text editor or an IDE
  - Customize the environment to suit your needs and habits
- You can use different virtualization software, of course

Environment setup - lazy way 2/3

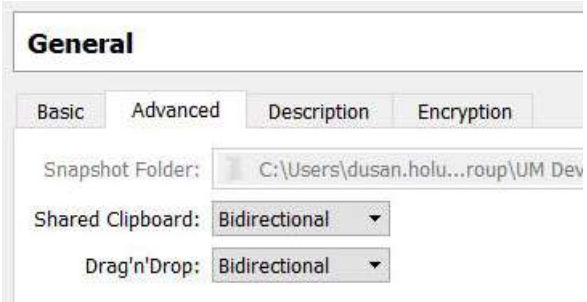
How the virtual appliance has been installed?

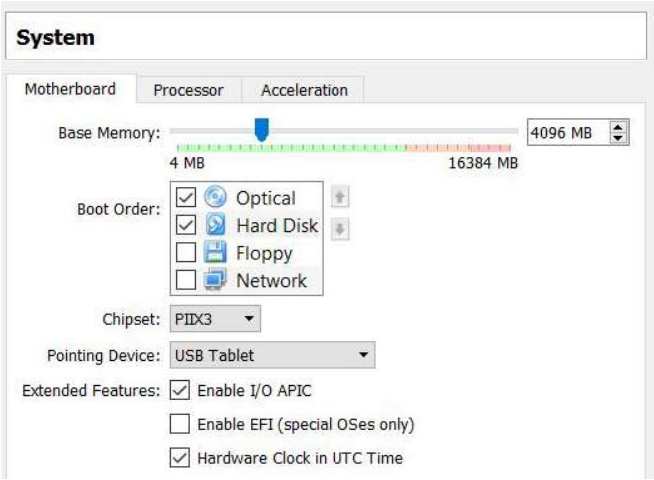
- Vanilla xubuntu-16.04.3-desktop-amd64.iso installation
- **Installation according to previous explanation - both DIY options performed**
- The terminal is tweaked to:
  - bigger size
  - bigger fonts
  - F10 key disabled not to activate terminal menu (for **`mc`**)
- The file manager is set to:
  - Show full file info in one line, instead of icons only
  - Perform single click actions (instead of one click)
- The terminal is set to have a red cursor
- The .bashrc is modified for better prompt. Well, much better. And nicer.

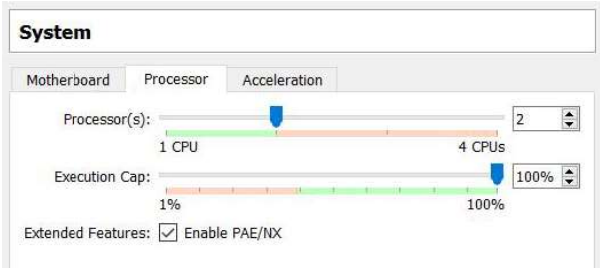


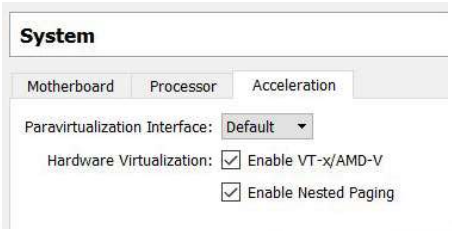
Environment setup - lazy way 3/3

- Virtualbox is set to:
  - automatically mount `c:\Users\_shared` for read and write (**Do not forget to set directory permissions in host!** or change the mountpoint)
  - Bidirectional clipboard between a host and a virtual machine









Do I need a user module? Pros and cons.

A user module:

- Needs an SDK, C cross compiler suite
- Has a GUI via web interface
- Has stored configuration support
- Is not portable
- Is for simple to complex functionality
- Is faster in execution
- Non-interactive development

Script:

- Needs shell scripting tools only
- Has no GUI*
- Has no stored configuration*
- Is portable **
- Is for simple functionality only
- Is slower in execution
- More interactive development - test every command interactively

* Unless you code it on your own
** To some extend

B+B SmartWorx Powered by Advantech

Scripting tips 1/2

- Start needed script in router's *startup script* by

  `/<path>/<yourscriptname> &`

  Or put the whole script into *startup script*

- Use **`/usr/bin/logger -t SCRIPT_IDENTIFIER “$0 Message”`**

  to log messages into syslog on DEBUG level

- See the end of the presentation for useful resources

Scripting tips 2/2

Do not forget preinstalled powerfull linux utilities

- **`grep`** to find line of interest
- **`awk`** to work with column(s) of interest
- **`sed`** to find / replace part of the text

`Note:` **`bash`** has alot to offer - variables, math, loops, input and outputs, ..

User Module run environment 1/2

- UM process are running under **root**
- Every module is installed in it's own **`/opt/<modulename>`** directory
  - **`/opt/<modulename>/bin`** Module binaries
    - **`/opt/<modulename>/etc`**

`install`	script executed **after** UM installation
`uninstall`	script executed **before** UM uninstallation
`ip-up`	script executed **after** WAN connection is established
`ip-down`	optional script executed **after** WAN connection is lost
`init`	module control script

`init start`
is called for all installed modules on router startup

WARNING
These scripts are called by OS one by one for each module. Maximum execution length of a script should be up to few seconds. Do not block OS to execute other scripts!

B+B SmartWorx Powered by Advantech

User Module run environment 2/2

- Every module is installed in it's own **`/opt/<modulename>`** directory
  - **`/opt/<modulename>/www`** Contains:
    - **`CGI`** scripts (usually links to **`../bin/cgi`**)
    - Static web pages
    - Pictures
    - Whatever else related to www

- Logs are in system log: **`/var/log/messages`**

Build environment - filesystem 1/4

```
├── tcbuilder
├── modulessdk
│   ├── images
│   │   ├── ...
│   │   ├── example1
│   │   │   ├── example1.v2.tgz
│   │   │   └── example1.v3.tgz
│   │   └── template
│   ├── library
│   │   ├── obj.v2
│   │   └── obj.v3
│   └── modules
│       ├── ...
│       ├── example
│       │   ├── merge
│       │   │   ├── etc
│       │   │   └── www
│       │   └── source
│       │       ├── obj.v2
│       │       └── obj.v3
│
```

Compiled and packed module binaries are stored here

SDK sources and object files

Root directory for all module sources

Build environment - filesystem 2/4 - merge

```
├── modules
│   ├── example1
│   │   ├── Makefile
│   │   └── merge
│   │       └── etc
│   │           ├── defaults
│   │           ├── init
│   │           ├── install
│   │           ├── ip-down
│   │           ├── ip-up
│   │           ├── name
│   │           ├── uninstall
│   │           └── version
```

- Just copied into module package - remember run environment?
- **defaults** - default parameters
- **init** - module run control (start, stop, restart, status) - **init start** is called on router start
- **(un)install** - OS runs these after installation / before uninstallation
- **ip-down** / **ip-up** - OS runs these after WAN up / after WAN down
- **name** - a name of the module shown in GUI
- **version** - a version of the module shown in GUI after name

Note - files in **`merge`** are merged to router's **`/opt/<modulename>`** → you can use other directories, ie. **`bin`** or **`usr/bin`** to distribute binaries or other files

Build environment - filesystem 3/4 - merge

```
├── modules
│   ├── example1
│   │   ├── Makefile
│   │   └── merge
│   │       └── etc
│   │           ├── defaults
│   │           ├── init
│   │           ├── install
│   │           ├── ip-down
│   │           ├── ip-up
│   │           ├── name
│   │           ├── uninstall
│   │           └── version
```

`uninstall` script should remove everything that **`install`** script created including symlinks.

Build environment - filesystem 4/4 - source

```
├── modules
│   ├── example...
│   │   ├── merge
│   │   └── source
│   │       ├── Makefile
│   │       ├── module.c
│   │       ├── module_cfg.c
│   │       ├── module_cfg.h
│   │       ├── module_cgi.c
│   │       ├── module.h
│   │       ├── obj.v2
│   │       │   ├── module
│   │       │   ├── module_cfg.d
│   │       │   ├── module_cfg.o
│   │       │   ├── module_cgi
│   │       │   ├── module_cgi.d
│   │       │   ├── module_cgi.o
│   │       │   ├── module.d
│   │       │   └── module.o
│   │       └── obj.v3
│   │           ├── module
```

- v2 and v3 binaries are not exchangeable
- Put all your sources and headers in the source directory
- Add your sources to the Makefile

Before you start - 1/2 - versioning system

Set up a versioning system and, please, do backups

- Or configure your own version control system according to your needs
  - The UM SDK git repository is not writable for you → set your own git repository or other versioning system
- Or set raw backups
  - Cloud
  - Other
- Do regular backups. Please.

Before you start - 2/2 - define your globals

See **module.h**

```
#ifndef _MODULE_H_
#define _MODULE_H_

// module title
#define MODULE_TITLE        "Example 5"

// module name
#define MODULE_NAME         "example5"

// prefix for all parameters
#define MODULE_PREFIX       "MOD_EXAMPLE5_"

// name of configuration file
#define MODULE_SETTINGS     "/opt/" MODULE_NAME "/etc/settings"

// name of init script
#define MODULE_INIT         "/opt/" MODULE_NAME "/etc/init"

#endif
```

As seen on GUI

Filesystem friendly name

Unique identifier among ALL modules. Should start with **MOD_** and be uppercase

You can run this file via:
. /opt/<modulename>/etc/settings
And then test environment variables

A User Modules' API Overview

API classes

- User module environment binding (configuration, web interface) allow UM properly settle into a router
- Router peripherals accessibility (serial ports, GPIO)
- Operating system access (file, locking, sockets, systime)
- Helpers (base64 enc/decoder, html builder, SNMP, Python binding)

Note: Use SDK functions where possible instead of your own.

API - Configuration

Configuration stored in filesystem - see **um_cfg.h**, **um_cfg.c**

```
// open and close configuration file
extern FILE *um_cfg_open(const char *config_name, const char *open_mode);
extern void um_cfg_close(FILE *file_ptr);

// put string into configuration file, get string from configuration file (allocate memory for result)
extern void um_cfg_put_str(FILE *file_ptr, const char *name, const char *value);
extern char *um_cfg_get_str(FILE *file_ptr, const char *name);

// put integer into configuration file, get integer from configuration file
extern void um_cfg_put_int(FILE *file_ptr, const char *name, unsigned int value, int store_zero);
extern unsigned int um_cfg_get_int(FILE *file_ptr, const char *name);

// put boolean into configuration file, get as integer
extern void um_cfg_put_bool(FILE *file_ptr, const char *name, unsigned int value);

// put IP address into configuration file, get IP address from configuration file
extern void um_cfg_put_ip(FILE *file_ptr, const char *name, unsigned int value);
extern unsigned int um_cfg_get_ip(FILE *file_ptr, const char *name);
```

Your module - configuration example - header file

```
// module_cfg.h
#ifndef _MODULE_CFG_H_
#define _MODULE_CFG_H_

// configuration
typedef struct {
  unsigned int          enabled;
  char                  *device;
  unsigned int          baudrate;
  unsigned int          databits;
  char                  *parity;
  unsigned int          stopbits;
  unsigned int          port;
} module_cfg_t;

// load configuration from file
extern void module_cfg_load(module_cfg_t *cfg_ptr);

// save configuration to file
extern int module_cfg_save(module_cfg_t *cfg_ptr);

#endif
```

Good practise
Always allow administrator to enable / disable your module

Your configuration parameters

Your module - configuration example - load

```
// module_cfg.c
#include <stdio.h>
#include "um_cfg.h"
#include "module.h"
#include "module_cfg.h"
// load configuration from file
void module_cfg_load(module_cfg_t *cfg_ptr)
{
  FILE                  *file_ptr;

  file_ptr          = um_cfg_open(MODULE_SETTINGS, "r");
  cfg_ptr->enabled  = um_cfg_get_int(file_ptr, MODULE_PREFIX "ENABLED" );
  cfg_ptr->device   = um_cfg_get_str(file_ptr, MODULE_PREFIX "DEVICE"  );
  cfg_ptr->baudrate = um_cfg_get_int(file_ptr, MODULE_PREFIX "BAUDRATE");
  cfg_ptr->databits = um_cfg_get_int(file_ptr, MODULE_PREFIX "DATABITS");
  cfg_ptr->parity   = um_cfg_get_str(file_ptr, MODULE_PREFIX "PARITY"  );
  cfg_ptr->stopbits = um_cfg_get_int(file_ptr, MODULE_PREFIX "STOPBITS");
  cfg_ptr->port     = um_cfg_get_int(file_ptr, MODULE_PREFIX "PORT"    );
  um_cfg_close(file_ptr);
}
// ...
```

API library

Your module globals

Your module configuration functions

Modify this block according to your configuration parameters

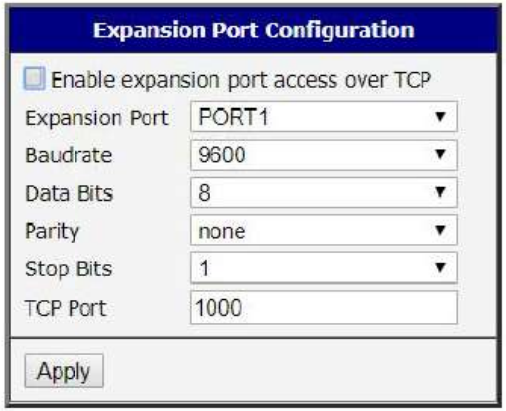
Your module - configuration example - save

```c
// module_cfg.c

// save configuration to file
int module_cfg_save(module_cfg_t *cfg_ptr)
{
  FILE                  *file_ptr;

  if ((file_ptr = um_cfg_open(MODULE_SETTINGS, "w"))) {
    um_cfg_put_bool(file_ptr, MODULE_PREFIX "ENABLED" , cfg_ptr->enabled    );
    um_cfg_put_str (file_ptr, MODULE_PREFIX "DEVICE"  , cfg_ptr->device     );
    um_cfg_put_int (file_ptr, MODULE_PREFIX "BAUDRATE", cfg_ptr->baudrate, 1);
    um_cfg_put_int (file_ptr, MODULE_PREFIX "DATABITS", cfg_ptr->databits, 1);
    um_cfg_put_str (file_ptr, MODULE_PREFIX "PARITY"  , cfg_ptr->parity     );
    um_cfg_put_int (file_ptr, MODULE_PREFIX "STOPBITS", cfg_ptr->stopbits, 1);
    um_cfg_put_int (file_ptr, MODULE_PREFIX "PORT"    , cfg_ptr->port    , 0);
    um_cfg_close(file_ptr);
    return 1;
  }

  return 0;
}
// ...
```



Modify this block according to your configuration parameters.
Check compatibility with **`module_cfg_load`**.

Your module - using configuration

```
// module.c
#include "module.h"
#include "module_cfg.h"

int main(int argc, char *argv[])
{
  module_cfg_t          cfg;            // configuration parameters structure
  char                  device[16];     // full path to device

  // load configuration
  module_cfg_load(&cfg);

  // open serial port
  snprintf(device, sizeof(device), "/dev/%s", cfg.device);
  if (!um_lock_acquire(device, 1)) {
    syslog(LOG_ERR, "device %s is locked", device);
    exit(1);
  }
  fd_com = um_com_open(device, cfg.baudrate, cfg.databits, cfg.parity, cfg.stopbits);
  if (fd_com < 0) {
    syslog(LOG_ERR, "unable to open %s", device);
    exit(1);
  }
  // ...
```

Your configuration structure, load and save functions prototypes

API - Configuration via WEB - functions

Configuration modification via WEB and CGI - see **um_cgi.h**, **um_cgi.c**

```
// start and finish CGI script parameters processing
extern void um_cgi_begin(void);
extern void um_cgi_end(void);

// check if query is ok
extern int um_cgi_query_ok(void);

// get CGI script parameter (raw string)
extern int um_cgi_get_raw(const char *name, char **value_ptr);

// get CGI script parameter (string, integer, bool, IP, MAC)
extern int um_cgi_get_str(const char *name, char **value_ptr, int no_spaces);
extern int um_cgi_get_int(const char *name, unsigned int *value_ptr);
extern int um_cgi_get_bool(const char *name, unsigned int *value_ptr);
extern int um_cgi_get_ip(const char *name, unsigned int *value_ptr, int zero);
extern int um_cgi_get_mac(const char *name, char **value_ptr, int zero);
```

Note: No setters here. Setting parameters is done via web pages only - see following.

Your module - Configuration via WEB - preparation

Configuration modification via WEB and CGI - see **`module_cgi.c`** from example 5

```
#include "um_cgi.h"
#include "um_html.h"
#include "um_process.h"

#include "module.h"
#include "module_cfg.h"

// structure of main menu
static const um_menu_item_t MENU[] = {
  { "Status"        , NULL                 },
  { "System Log"    , "slog.cgi"           },
  { "Configuration" , NULL                 },
  { "Expansion Port", "index.cgi"          },
  { "Customization" , NULL                 },
  { "Return"        , "../../module.cgi"   },
  { NULL            , NULL                 }
};
```

API libraries

Your module globals

Your module configuration structure and functions prototypes

Your module pages - see next slide

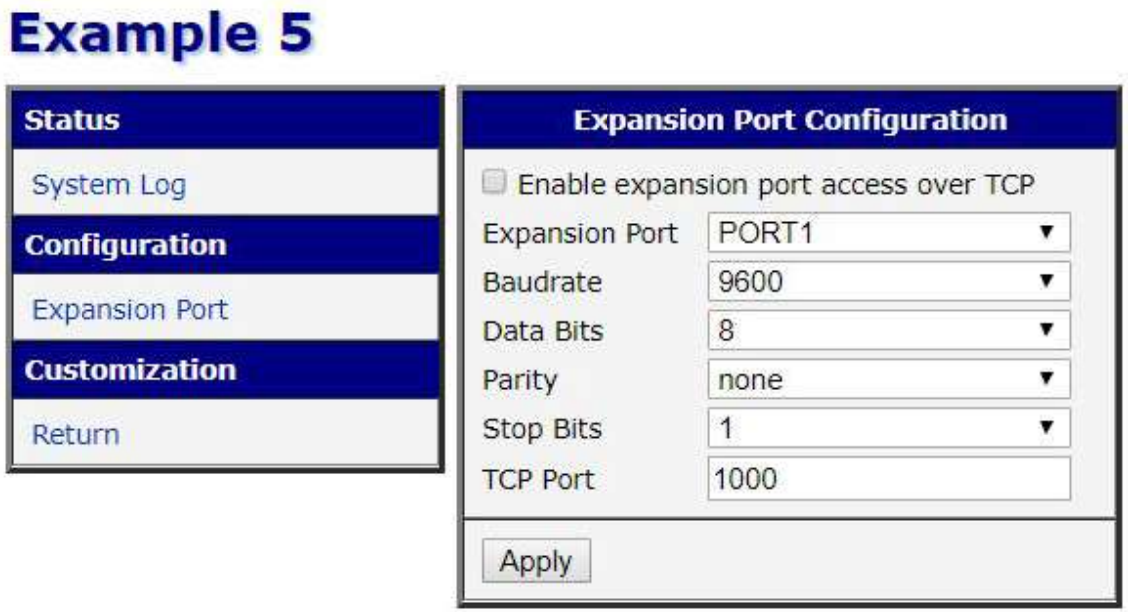
Your module - Configuration via WEB - mapping

Configuration modification via WEB and CGI - see **`module_cgi.c`** from example 5

```
#include "um_cgi.h"
#include "um_html.h"
#include "um_process.h"

#include "module.h"
#include "module_cfg.h"

// structure of main menu
static const um_menu_item_t MENU[] = {
  { "Status"        , NULL                 },
  { "System Log"    , "slog.cgi"           },
  { "Configuration" , NULL                 },
  { "Expansion Port", "index.cgi"          },
  { "Customization" , NULL                 },
  { "Return"        , "../../module.cgi"   },
  { NULL            , NULL                 }
};
```



Your module - Configuration via WEB - options

Module_cgi.c

```
// options for parameter "Expansion Port"
static const um_option_str_t DEVICE[] = {
  { "PORT1", "ttyS0" },
  { "PORT2", "ttyS1" },
  { NULL   , NULL    }
};


// options for parameter "Data Bits"
static const um_option_int_t DATABITS[] = {
  { "8" , 8 },
  { "7" , 7 },
  { NULL, 0 }
};
```

These names will be used in GUI listboxes (see next page)

Use proper functions corresponding to types used

Your module - web page generation - main_index

```
// main function of CGI script "index.cgi"
static void main_index(void)
{
  module_cfg_t  cfg;
  module_cfg_load(&cfg);
  um_html_page_begin(MODULE_TITLE);
  um_html_form_begin(MODULE_TITLE, "Expansion Port Configuration", "set.cgi", 0, JAVASCRIPT_BEGIN, MENU);
  um_html_table(2, 0);
  um_html_check_box("enabled", cfg.enabled);
  um_html_text("Enable module");
  um_html_table(2, 100);
  um_html_text("Expansion Port");
  um_html_select_str("device", cfg.device, DEVICE);
  um_html_text("Baudrate");
  um_html_select_int("baudrate", cfg.baudrate, BAUDRATE);
  ...
  um_html_form_break();
  um_html_table(1, 0);
  um_html_submit("button", "Apply");
  um_html_form_end(JAVASCRIPT_END);
  um_html_page_end();
}
```

Modify to fit your needs

Configuration parameters fill-in

Choose good identifiers

Check proper types

Your module-web page generation - getters/setters

```
static void main_set(void)
{
  module_cfg_t          cfg;
  int                   ok, input_ok;

  um_cgi_begin();
  um_html_page_begin(MODULE_TITLE);
  ok       = 0;
  input_ok = um_cgi_query_ok()                                &&
             um_cgi_get_bool("enabled" , &cfg.enabled    ) &&
             um_cgi_get_str ("device"  , &cfg.device   , 1) &&
             um_cgi_get_int ("baudrate", &cfg.baudrate   ) &&
             // …
             um_cgi_get_int ("port"    , &cfg.port       );

  if (input_ok) {
    if (module_cfg_save(&cfg)) {
      ok = !um_process_exec(MODULE_INIT, "restart", NULL);
    }
  }
  um_html_config_info_box(ok, input_ok, 0, "index.cgi");
  um_html_page_end();
  um_cgi_end();
}
```

Read parameters from GUI

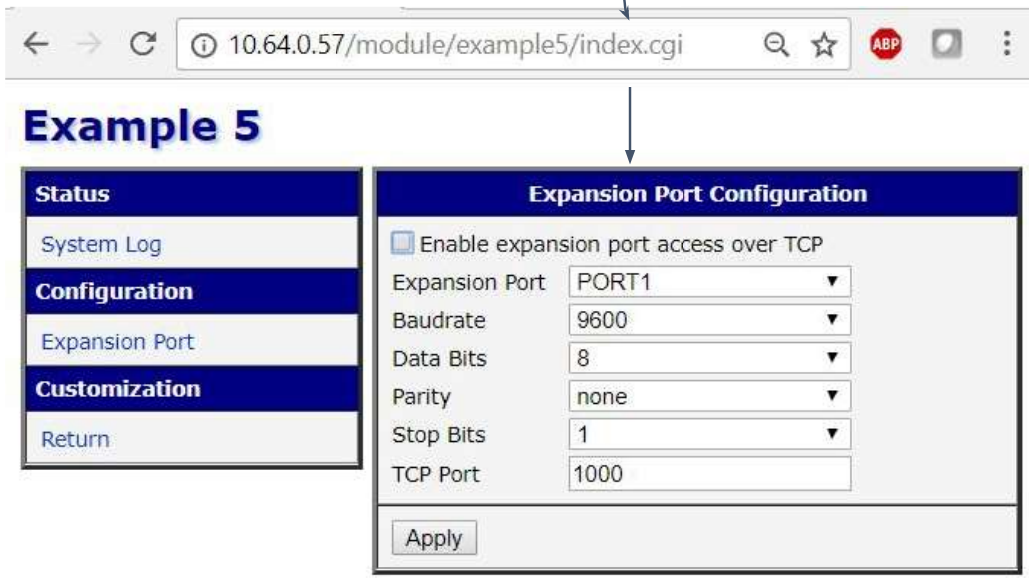
Save to filesystem and restart the module with the new configuration.

Your module - web page generation - main

```
// cgi main function
int main(int argc, char *argv[])
{
  const char            *name;

  if (argc > 0) {
    name = basename(argv[0]);
    if (!strcmp(name, "index.cgi")) {
      main_index();
    } else if (!strcmp(name, "set.cgi")) {
      main_set();
    } else if (!strcmp(name, "slog.cgi")) {
      main_slog();
    }
  }
  return 0;
}
```

Pair URL with C functions to generate proper web pages



Your module - logging

```
// main function excerpt
#include <syslog.h>
int main(int argc, char *argv[])
{
  // open system log
  openlog(basename(argv[0]), LOG_PID | LOG_NDELAY, LOG_DAEMON);
  syslog(LOG_NOTICE, "started");
  // ...
  syslog(LOG_NOTICE, "TCP connection from %s established", inet_ntoa(addr.sin_addr));
  syslog(LOG_ERR, "recv error: %s", strerror(errno));
  // ...
  syslog(LOG_NOTICE, "stopped");

  // no need to close syslog file - OS does so on process termination
  return 0;
}
```

Good practice
Always use **started** and **stopped** messages

API - Peripherals access - functions 1/3

GPIO **um_gpio.h**, **um_gpio.c**
XCCNT **um_xccnt.h**, **um_xccnt.c**

See **um_gpio.h** for full lists

```
// types of expansion ports
#define UM_GPIO_PORT_TYPE_EMPTY              0x00
#define UM_GPIO_PORT_TYPE_RS232              0x02
//…

// types of module boards
#define UM_GPIO_MODULE_TYPE_EES3             0x00
//…
#define UM_GPIO_MODULE_TYPE_NONE             0x0F

// get index of selected module
extern int um_gpio_get_module_idx(void);

// get type of module board
extern int um_gpio_get_module_type(int module);

// get index of selected SIM card
extern int um_gpio_get_module_sim(int module);
```

API - Peripherals access - functions 2/3

```
// get type of expansion port
extern int um_gpio_get_port_type(int port);

// get/set shutdown of expansion port
extern int um_gpio_get_port_sd(int port);
extern void um_gpio_set_port_sd(int port, unsigned int state);

// get state of input / set state of output on expansion board
extern int um_gpio_get_port_input(int port, unsigned int idx);
extern void um_gpio_set_port_output(int port, unsigned int idx, unsigned int state);

// get information about overload on MBUS
extern int um_gpio_get_mbus_overload(int port);
```

API - Peripherals access - functions 3/3

```
// get/set state of output OUT0
extern int um_gpio_get_out0(void);
extern void um_gpio_set_out0(unsigned int state);

// get / set state of output OUT1
extern int um_gpio_get_out1(void);
extern void um_gpio_set_out1(unsigned int state);

// get / set state of input BIN0
extern int um_gpio_get_bin1(void);
extern int um_gpio_get_bin0(void);

// set state of LED USR
extern void um_gpio_set_led_usr(unsigned int state);
```

API - Peripherals access - gpio example

```
#include <fcntl.h>        // open
#include <sys/ioctl.h>    // ioctl

#include "um_gpio.h"

#define UM_GPIO_GET_BIN1            0x8000421BU
// default return value for *_available functions
#define OHC_RETURN_NULL             -10

// This function returns 1 if BIN1 is available and 0 if BIN1 is not available.
int bin1_available(void)
{
  static int result = OHC_RETURN_NULL;

  // read just once per process. Once value is read, just return the stored result.
  if (result == OHC_RETURN_NULL) {
    result = um_gpio_get_bin1() == -1 ? 0 : 1;
  }
  return result;
}
```

Note: See attachments to this presentation for full list of command codes

API - Peripherals access - xccnt in cgi example

```
// xccnt in cgi

if ( um_gpio_get_port_type( 0 ) == UM_GPIO_PORT_TYPE_CNT )
{  um_html_table(1, 110);
   um_html_pre_head( "External IO (PORT1)" );
   um_xccnt_get_features( 0, &features );
   um_html_table(2, 110);
   for (i = 0;i < 4;i++)
   { if ( features & (UM_XCCNT_FEATURE_BIN1 << i) )
     { snprintf(temp, sizeof(temp), "External Input %u:", i + 1);
       um_html_text(temp);
       um_xccnt_get_input(0, i, &value);
       um_html_text(value ? "Off" : "On");
     }
   }
   if ( features & UM_XCCNT_FEATURE_OUT1 )
   { um_html_table(3, 110);
     um_html_text("External Output:");
     um_xccnt_get_output(0, &value);
     um_html_text(value ? "On" : "Off");
     um_html_submit("out1", value ? "Off" : "On");
   }
 }
```

See "**GPIO driver...pdf**" for complete constants reference

API - Operating system access

Process	**um_process.h**,	**um_process.c**
Sockets	**um_socket.h**	**um_socket.c**
Systime	**um_systime.h**	**um_systime.c**
File	**um_file.h**	**um_file.c**

```
// um_process.h
// start a process and wait for its completion
extern int um_process_exec(const char *program, ...);

// start a process in the background
extern pid_t um_process_start(const char *program, ...);

// send a signal to process running in the background
extern int um_process_kill(pid_t pid, int sig);

// check existence of process running in the background
extern int um_process_exists(pid_t pid);

// um_systime.h
// get monotonic system time
extern time_t um_systime(void);
```

API - Operating system access - sockets

```
// um_socket.h
// open TCP socket (server mode)
extern int um_socket_open_tcp_server(unsigned int port);
// open TCP socket (client mode)
extern int um_socket_open_tcp_client(unsigned int ip, unsigned int port);

// open UDP socket (server mode)
extern int um_socket_open_udp_server(unsigned int port);
// open UDP socket (client mode)
extern int um_socket_open_udp_client(unsigned int ip, unsigned int port);

// open UNIX socket (server mode)
extern int um_socket_open_unix_server(const char *path);
// open UNIX socket (client mode)
extern int um_socket_open_unix_client(const char *path);

// close socket
extern void um_socket_close(int sock);

// enable keepalive on TCP socket
extern void um_socket_keepalive(int sock, int time, int intvl, int probes);
```

API - Helper functions

SNMP constants and structures	**um_snmp.h**	**um_snmp.c**
Communication via serial device	**um_com.h**	**um_com.c**
Base64 encoding and decoding	**um_base64.h**	**um_base64.c**

```
// um_com.h
// open and close serial port
extern int um_com_open(const char *device, unsigned int baudrate, unsigned int databits, const char *parity, unsigned int
stopbits);
extern void um_com_close(int fd);
// send and receive data
extern char *um_com_xmit(int fd, const char *str, int timeout);
// wait until transmitter is empty
extern void um_com_drain(int fd);
// set state of RTS and DTR signal
extern void um_com_set_rts(int fd, int state);
extern void um_com_set_dtr(int fd, int state);
// get state of CTS,DSR and CD signals
extern int um_com_get_cts(int fd);
extern int um_com_get_dsr(int fd);
extern int um_com_get_cd(int fd);
```

API - Helper functions - example - lock device

```
  // find out module type
  module_type = um_gpio_get_module_type(0);
  switch (module_type) {
    case UM_GPIO_MODULE_TYPE_EES3:
      dev_name = "/dev/ttyS2";
      break;
    case UM_GPIO_MODULE_TYPE_EU3:
    case UM_GPIO_MODULE_TYPE_MCXXXX_VWM10:
    case UM_GPIO_MODULE_TYPE_MCXXXX_MCXXXX:
    case UM_GPIO_MODULE_TYPE_PHS8:
    case UM_GPIO_MODULE_TYPE_MCXXXX:
    case UM_GPIO_MODULE_TYPE_VWM10:
    case UM_GPIO_MODULE_TYPE_GOBI3K:
      if (modul > 1) {
        dev_name = "/dev/ttyUSB16";
        } else {
          dev_name = "/dev/ttyUSB8";
        }
        break;
// ...
// about to open serial port
if (um_lock_acquire( dev_name, 10 )) {
  syslog(LOG_ERR, "port open failed %i", modul);
  um_lock_release( dev_name );
```

Device mapping to module type

Creating your module from existing example 1/3

1. Create your module directory `~/src/modulessdk/modules/<yourmodulename>`
2. Copy **WHOLE** suitable example from `~/src/modulessdk/modules/example*`
3. Set module name in:
   - `./merge/etc/init`
   - `./merge/etc/name`
   - `./source/module.h`
   - `./source/Makefile`
4. Update version history in `./version.txt`
5. Program parameters load and save in:
   - `module_cfg.h, module_cfg.c`
   - `module_cgi.c`

Modulename: allowed characters are letters, numbers and _ only

!

Creating your module from existing example 2/3

5. Write your functionality into **`module.c`**

Temporary files must be created in RAM (`/var` or `/tmp` - find out by issuing mount command**).**

6. If other than default source files needed, update
`~/src/modulessdk/modules/<yourmodulename>/source/Makefile`

Example:

```
MODULE_EXE = module
MODULE_SRC = module.c module_cfg.c availability.c ping.c data.c

MODULE_CGI_EXE = module_cgi
MODULE_CGI_SRC = module_cgi.c module_cfg.c html_mod.c data.c availability.c
```

Creating your module from existing example 3/3

7. Build the module:

```
$make clean
$make
```

8. Use logging for debugging. If really needed, we have **`gdb`** user module.

9. See **`/var/log/syslog`** for logs

10. **`while (self.happy == FALSE)`**

Hints and tips for better make

- You can use **make DEBUG=1** to cross-compile a User Module with debug symbols for the GDB.
- You can use **make WARN=1** or **make WARN=2** to cross-compile a User Module with additional warning messages from the GCC.
- You can use **make ASAN=1** to cross-compile a User Module with enabled Address Sanitizer.
- You can use **make UBSAN=1** to cross-compile a User Module with enabled Undefined Behavior Sanitizer.
- You can use **make SSP=1** to cross-compile a User Module with enabled Stack Smashing Protector.
- You can use **make upload [ROUTER=<IP address>] [USERNAME=<username>] [PASSWORD=<password>]** to upload User Module directly into the router.
- You can use **make -j8** to speed up cross-compilation significantly on multicore systems.
- You can install ccache to speed up recompilation significantly.
- You can set environment variable GCC_COLORS to enable colorization of GCC output.

System utilities 1/2

`/bin/busybox` linked utilities - see **`/bin`** and **`/bin/busybox`**
Other useful utilities:

`email`	Send email
`gsmat`	Send AT command to active MWAN module
`gsmpwr`	Command GSDM module power
`gsmsms`	Send sms. **`sms`** does the same
`inc`	Increments parameter and returns incremented value to stdout
`io`	Get / set user's binary pin value
`led`	Set user's LED to on | off
`mac`	(Internal - do not use)
`sms`	(Internal - do not use)
`status`	Get status of specified network interface
`snmptrap`	Send an snmp trap

For details issue **`<command> --help`** via **`ssh`** and/or see the attachments

System utilities 2/2

shlock Simple semaphore - internal usage, but might help you

script 1 execution timeline

time

```
# try to get lock
echo "script1 - trying to get lock.."
shlock /var/lock/my_identifier1
echo "script1 - got shlock"
...
exit
```

`shlock` returned immediately

script 2 execution timeline

```
# try to get the same lock as in script1
echo "script2 - trying to get lock.."
shlock /var/lock/my_identifier1

echo "script2 - got lock"
exit
```

`shlock` was waiting till end of script1

Workshop - DIY user module

1. Check **`ModulesSDK`** build ability - **build examples**
2. Make new module as a copy of some example - **give it a new name, define globals**
3. Build your module as is - **test in router**
4. Modify configuration to your needs - **check exported router configuration**
5. Modify functionality - test it, **check logging in syslog**
6. Code, do unit tests, test extensively.
7. Test in operating environment.
8. Optimise logs.
9. Enjoy!

Troubleshooting 1/3

1. **UM does not load default values**
   - Parameter names or types does not match in all of **`etc/defaults`**, **`module_cfg.h`**, **`module_cfg.c`**
   - Init script is corrupted - run it manually from **`/opt/<modulename>/etc`** and see results

2. **UM does not correctly load / save parameters**
   - Parameter types or names in **`module_cgi.c`** does not match to **`etc/defaults`**, **`module_cfg.h`** or **`module_cfg.c`**
   - Check parmeter names and values in report (“Save report” button)

Troubleshooting 2/3

3. UM does not seem to start / stop

- UM has not been started - check init script
- UM crashed - check **`ps`**, run it manually via **`ssh`**, use logging to syslog

4. UM does not do what expected

- Installing old version - check timestamp of module package Always compile from **`~/src/ModuleSDK`** directory
- Name or parameters name mismatch - check if you really have unique module name in parameters
- Use more logging to see what is happening

Troubleshooting 3/3

5. Router is slow

- Check your coding - do not use polling, if possible
- Use **`select`**, **`read`**, for blocking reads
- Use signals instead of polling
- Use **`sleep`** to lower cpu ussage
- Call system utilities instead of your coding.

Final notes

1. Advantech B+B SmartWorx can not support your own modules
2. If you need a module made for you, please contact a local partner of Advantech B+B SmartWorx
3. When designing your module, remember that you are on embedded device with limited memory amount and CPU power
4. Think first, code later
5. WAN data might cost money (dependent on a mobile carrier plan) - test on WiFi if possible

Useful resources

Development

- Advanced **Bash Scripting** Guide http://www.tldp.org/LDP/abs/abs-guide.pdf
- Beej's Guides from **network programming** in C, GNU **debugger** up to dragon killing https://beej.us/guide/
- Make and **makefile** documentation https://www.gnu.org/software/make/manual/make.html

GIT

- Advantech B+B Smartworx **GIT** repository https://bitbucket.org/bbsmartworx
- **GIT cheat sheet** https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet
- **GIT howto** for beginners: http://rogerdudler.github.io/git-guide/

Others

- **Busybox** info https://busybox.net/about.html

Thank you!

I wish you happy coding with great results

http://advantech-bb.com
dusan.holub@advantech-bb.cz

B+B SmartWorx
Powered by ADVANTECH