Technical Manual of the `Universal Server`: US-Main

Organisation: Copyright (C) 2019-2025 Olivier Boudeville

Contact: about (dash) universal-server (at) esperide (dot) com

Creation date: Saturday, May 2, 2020

Lastly updated: Wednesday, April 30, 2025

Version: 0.0.20

Status: In progress

Dedication: Users and maintainers of the `Universal Server`, US-Main.

Abstract: The Universal Server (US-Main), part of the umbrella project of the same name, is a multi-service daemon in charge of the automation (monitoring, scheduling and performing) of various computer-based tasks, such as the proper management of the server itself or, in the future, of home automation.

We present here a short overview of these services, to introduce them to newcomers.

Table of Contents

Overview

We present here a short overview of the general automated services offered by our so-called *Universal Server*, to introduce them to newcomers. These services are implemented by US-Main, which relies notably on US-Common.

Beyond this document, the next level of information about US-Main is either to browse the US-Main API documentation or simply to read the corresponding source files, which are intensely commented and generally straightforward.

The project repository is located here.

Layer Stack

From the highest level to the lowest, as summarised here, a (free software) stack involving the Universal Server usually is like:

- the *Universal Server* services themselves (i.e. this US-Main layer)
- [optional] the *Universal Webserver*, i.e. US-Web (for any future web interaction)
- US-Common (for US base facilities)
- [optional] Ceylan-Mobile (for 3G connectivity, notably SMS sending, relying on the Gammu library)
- [optional] Ceylan-Seaplus (prerequisite of Ceylan-Mobile, for a bridge from Erlang to the C language)
- Ceylan-Oceanic (for the support of home automation, through the integration of Enocean devices and actuators; relies on our fork of erlang-serial)
- Ceylan-Traces (for advanced runtime traces)
- Ceylan-WOOPER (for OOP)
- Ceylan-Myriad (as an Erlang toolbox)
- Erlang/OTP (for the compiler and runtime)
- GNU/Linux

The shorthand for `Universal Server` (a.k.a. `US-Main`) is `um`.

Facilities Provided by this US-Main Layer

These are mainly **administration-related services**.

> **Note**
>
> In the course of the presentation of these facilities, related configuration information is given; it is to be applied when Configuring US-Main.

Home Automation

The US-Main server offers house automation services based on the Enocean protocol and associated devices (sensors and actuators).

These services are implemented thanks to our Ceylan-Oceanic library; please refer to it for a better understanding of EURIDs, base identifiers, telegrams and so on.

A key point is to define the Oceanic settings, so that the various devices involved can be monitored and correctly interpreted.

Presence Simulator

The goal is to give to any outside observer the illusion that a building is currently inhabited, for example by switching light(s) on and off as if actual people were busy in such a premise.

For that, US-Main will control an (Enocean) smart plug (itself able typically to toggle a well-chosen, low-consumption lamp visible from outside), based on intra-day logical presence slots. Such a slot is defined by two milestones, a start and a stop one (respectively to switch the smart plug on and off), which may be either a (fixed) intra-day `time()` (hence `{Hour,Minute,Second}`), or the `dawn` and `dusk` symbolic deadlines (atoms), which are then recomputed each day (based on the current date, and on the longitude and the latitude specified by the user for the location of the premise of interest).

The general principle, if the so-called "smart-lighting" feature is enabled, is to switch on a lamp during the busy hours of the day iff no natural light can be expected. All possible cases are expected to be covered (even, for extreme latitudes, days without dawn and/or dusk).

If smart lighting is not enabled, then during a slot of simulated presence, the actuator will be triggered unconditionally (that is whether daylight shall be expected or not).

By default, the following single intra-day logical presence slots, controlling a single set of actuators, are defined, for a simulated presence with smart lighting:

- from `TMorningStart = 7:30 AM` to `TMorningStop = 8:30 AM`, unless any dawn is to happen concurrently
- from `TEveningStart = 6:30 PM` to `TEveningStop = 11:45 PM`, unless any dusk is to happen concurrently

More generally, the user is free, through the `presence_simulation_settings`, either to select a default policy or to specify their own presence program (constant presence or absence, or a list of presence slots), with or without smart lighting.

For example, in one's US-Main configuration file of interest, possibly named `foobar-us-main-for-production.config`, as referenced in the `us_main_config_filename` key of `us.config` (both files being typically in `/etc/xdg/universal-server`)
one may go for the default program:

```
% The EURID of any device (typically a push button or a double rocker)
% that serves to indicate whether someone is at home:
%
{ presence_switching_actuator, "004584A6" }.

% The settings in terms of presence simulation:
{ presence_simulation_settings, { default, undefined } }.
```

If a user-specific program is preferred, here with a single slot, with default source and target EURIDs, and with smart lighting:

```
{ presence_simulation_settings, [
 { presence_simulation_setting, undefined, undefined,
   [ { { 6, 30, 0 }, { 8, 0, 0 } }, { { 19, 30, 00 }, { 23, 30, 00 } } ], true } ] }.
```

Note that US-Main manages gracefully DST (*Daylight saving time*).

Alarm System

The principle is to monitor a set of (Enocean) sensors (typically opening detectors for doors, windows, etc.), and to report whenever a problem is detected.

The US-Main must be first told when the alarm system shall be active; this is typically when no one is at home, which can be notified to US-Main either by executing a script (e.g. `leaving-home.sh`, in charge also for example of locking all running computers) or by pushing an (Enocean) foot switch located at the front door. Then, after a configurable delay, the alarm system will be enabled.

All sensors are then monitored, any opening being reported.

Other events will be reported, whether or not the alarm system is active:

- when an undeclared sensor is first seen
- if a known sensor vanishes (e.g. it has been destroyed, or it ran out of power)
- if a jamming attempt is detected

When any abnormal event occurs, US-Main logs it and may typically sends to the user notifications, by SMS and/or by e-mail.

Monitoring of Host Sensors

The objective here is to track the various (and numerous) sensors of interest that most modern computers include; should abnormal feedback be detected, it is to be automatically reported thanks to the communication gateway service.

The US Sensor Manager tracks automatically many **hardware sensors**; at start-up it detects the main available ones, regarding:

- **temperatures** at various locations: the CPU socket, the CPU package and cores themselves, any APU, the motherboard, the chipset, ACPI, some disks (e.g. NVME); in the future, adding GPU and RAM modules is considered
- the **speed of the fans** known of the motherboard (as opposed to any case fan that would be directly connected to the power supply and that would remain invisible)
- **chassis intrusion**, should such sensors be available

(other sensors like batteries, network or USB interfaces, etc. are at least currently ignored, as their measurements are mostly voltage levels)

From then, the sensor manager periodically monitors the various measurement points exhibited by such sensors: it does its best to filter bogus values, to detect abnormal changes and to report to the user any related issue.

Preparing the Setup

The monitoring done by this server relies on the `sensors` executable (typically `/usr/bin/sensors`, obtained generally from a package of the same name and relying on lm-sensors). One may install the `i2c-tools` package as well for DIMM information (see R2 below).

The `sensors-detect` script must have been run once by root beforehand (select then only the default, safer options, by hitting Enter repeatedly or simply use its `--auto` option), in order to configure sensors.

Sensor configuration is typically stored in `/etc/sensors3.conf`, and must exist prior to running the US-Main server.

Mode of Operation of the Sensor Manager

Once the sensor manager is started, **temperatures** are periodically tracked (i.e. the currently reported one, plus minimum, maximum, and average since start) and compared to thresholds (any critical temperature as reported by the chips, and also ones set by our sensor manager itself in order to trigger alarms).

Abnormal temperatures (that is, going above - or even below - relevant thresholds) are then automatically timestamped and reported to the user by the US logic (i.e. notified in traces with appropriate severity, and possibly sent to the user thanks to emails and/or SMS, see the communication gateway service).

Similarly, any **fan** that would stop whereas not being PWM[1] is reported, and the same applies should an **intrusion** happen.

Many sensors report bogus values; the US Sensor Manager does its best to filter them out appropriately. This includes temperatures outside of any realistic ranges and an intrusion being reported right from US-Main startup (whereas, supposedly, it had not happened already).

[1] PWM stands for Pulse-width modulation; the speed of these fans can be controlled by their power source (typically the motherboard).

Temperature monitoring Temperatures are monitored based on all the sensors that are supported by lm-sensors (notably the motherboard and CPU ones). Many sensors report, even when they are correctly tuned, bogus values, and are more like very poor random generators (see how to mute them).

The sensor manager considers that, when it starts, most temperatures are under control. So it will consider that any too low or too high temperature reported is bogus (refer to the {low,high}_bogus_temperature_threshold defines).

In the future, extra information sources could be used:

- Hard Disk Drives, thanks to hddtemp, libatasmart, udisks2 or smartmontools
- DIMM Temperature sensors (see R2)
- GPU, thanks to XNVCtrl for NVidia ones, or ADL SDK for ATI ones

Refer to R5 for further details.

Note that Platform Controller Hub (e.g. pch_cannonlake-virtual-*, pch_skylake-virtual-*, etc.) are Intel's single-chip chipsets; they tend to run hotter than CPUs.

They may be reported as autonomous first-level entries, or as measurement points of the motherboard.

Fan Control The rotation speed of the fans can be measured thanks to lm-sensors as well.

Note that not all fans are known of the motherboard, notably the ones that are directly controlled by the user through a button (e.g. stop/low/high) will remain invisible to all programs.

Currently the sensor manager is not able to discriminate between fixed-speed fans and PWM ones.

The pulses attribute (e.g. fan2_pulses) tells how many of such pulses are generated per revolution of the fan.

Chassis Intrusion In this last case, prior to launching the US server, one may try to reset them; for example, as root:

```
$ ls -l /sys/class/hwmon/hwmon*/intrusion*
-rw-r--r-- 1 root root 4096 Jul 11 19:30 /sys/class/hwmon/hwmon3/intrusion0_alarm
-rw-r--r-- 1 root root 4096 Jul  8 21:46 /sys/class/hwmon/hwmon3/intrusion0_beep
-rw-r--r-- 1 root root 4096 Jul 11 19:30 /sys/class/hwmon/hwmon3/intrusion1_alarm
-rw-r--r-- 1 root root 4096 Jul  8 21:46 /sys/class/hwmon/hwmon3/intrusion1_beep
$ echo 0 >| /sys/class/hwmon/hwmon3/intrusion1_alarm
$ cat /sys/class/hwmon/hwmon3/intrusion1_alarm
1
```

As shown, this may not succeed.

Muting Faulty Sensors Some sensors are hopelessly flawed and are bound to raise false alarms at any time.

Once they triggered a sufficient number of them, the safest route is to mute them, which can be done thanks to the `us_sensor_monitoring` entry of the US-Main configuration file.

Let's suppose that a sensor whose identifier is `{_SensorType=nct6792, _SensorInterface=isa, _SensorNumber="0a20"}` shall have its measurement point `AUXTIN1` be muted, and that one wants to disable another one, `{acpitz, acpi, "0"}`, as a whole (i.e. all its measurement points).

It can be done with the following configuration entry:

```
{sensor_monitoring, [

   % Under that key shall be specified a list of
   % {sensor_id(), 'all_points' | [user_specified_point()]} pairs
   % in order to mute the sensors / measurement points that are known to be
   % bogus:
   %
   {muted_measurements, [
       {{nct6792, isa, "0a20"}, ["AUXTIN1"]},
       {{acpitz, acpi, "0"}, all_points}
                             ]}
                    ]}.
```

Other Related Technical Information To access information regarding a given sensor, `psensor` may be used: open the preferences of the sensor (click on its name in the main window), and select the menu item *Preferences*, and look at the *Chip* field. See this link for more information.

The `sensors` tool is reporting values found in the Linux virtual file system directory, in `/sys/class/thermal/thermal_zone*/{temp,type}` for temperatures.

Examples:

- `Package id 0` is your (first) CPU
- `dell_smm-virtual-0` is your CPU fan, managed by your system firmware
- `acpitz-virtual-0` (*ACPI Thermal Zone*) is the temperature sensor near/on your CPU socket; this sensor can be unreliable
- `coretemp-isa-0000` measures the temperature of the specific cores

See the many comments in class_USSensorManager.erl for more details.

See also the following resources:

- R1: interpreting the output of `sensors`
- R2: the `lm_sensors` documentation of Arch Linux
- R3 and R4: `lm-sensors` tips and tricks
- R5: information about `psensor`
- R6: an example of preparation/tuning of one's sensors

Contact Directory

The US Contact Directory server allows US-Main to track information regarding US contacts, for various purposes, including for the US communication gateway.

Contact File Format

Contact files are ETF files that contain a range of information about persons and organisations of interest.

Each non-commented line of these files shall be of the following format:

```
-type contact_line() :: { UserId :: user_id(),
   FirstName :: ustring(), LastName :: ustring(), NickName :: ustring(),
   Comment :: ustring(), BirthDate :: maybe( ustring() ),
   LandlineNumber :: maybe( ustring() ), MobileNumber :: maybe( ustring() ),
   PrimaryEmailAddress :: maybe( ustring() ),
   SecondaryEmailAddress :: maybe( ustring() ),
   PostalAddress :: maybe( ustring() ),
   Roles :: [ role() ] }.
```

A typical contact line could then be:

```
{1, "James", "Bond", "007", "MI6 Agent 007", {17,5,1971},
 "+44 9 81 47 25 40", "+44 6 26 83 37 22", "james.bond@mi6.uk.org",
 undefined, undefined, [administrator, secret_agent]}.
```

See also our test contact ETF file as a full example thereof.

Contact File Location

The path to a contact file can be either specified as an absolute one, or as a relative one - in which case it will be deemed relative to the US configuration directory.

They may be mere symlinks pointing to contact files kept in VCS in other locations.

Communication Gateway

The purpose of the US Communication Gateway is to enable (possibly two-way) exchanges with the US users.

Such communication is not to happen frow a web-based medium (see US-Web for that), but through alternate modes such as SMS (relying then on Ceylan-Mobile, itself relying on Ceylan-Seaplus) and/or e-mails (relying then on the corresponding services of Ceylan-Myriad).

For that, the correspondance between a US role (e.g. `administrator`) and actual user information is established thanks to the contact directory service.

Network Support Monitoring

This service allows to ensure that the local host (on which US-Main is running) enjoys a functional **network support**, in terms of:

- ICMP probes (ping)
- Internet (IP) connectivity
- DNS resolution

This is checked by ensuring periodically that a set of target hosts, specified as direct IP addresses and/or DNS names, can indeed be interacted with through the network.

Of course any issue (typically outage of a given network service) is then reported by appropriate means (i.e. by SMS rather than by email then).

Remote Monitoring of Online Services

The purpose here is to monitor online services (typically websites) provided by networked peers.

Each service is tracked based on a set of information:

- protocol: `http`, `https`, maybe in the future `ftp` or alike
- base hostname, specified as a DNS name or an IP address
- possibly a resource designator (e.g. a specific URL) for the actual checking

Next Services

The following services are planned (some day) for addition:

- UPS (Uninterruptible Power Supply) monitoring, to be notified whenever a related event happens (typically a power failure from the electrical grid)

Configuring US-Main

Note

We discuss configuration before installation, as the settings of interest shall be defined prior to deployment, notably so that adequate permissions can be set on the installation, according to the user under which US-Main is intended to run.

General US Configuration

Even if these are just conventions, we consider that a US-Server can be deployed either in:

- **development** mode: by a regular user, with no specific permissions, directly from the user account, and with direct process control

- **production** mode: by dedicated US(-Main) user and group, with low privileges, in the system tree (`/etc`, `/opt`, `/var`), and controlled through systemd

All US-related configuration files respect the ETF file format.

US-Common-level Configuration

The US-Main server is part of our "Universal Server" infrastructure, and as such relies on the base US-Common configuration settings.

As the US-related configuration files are heavily commented, proceeding based on examples is the simplest approach[2].

Refer for that at this us.config example.

Location of the US Configuration Files

The US configuration files will be searched through various filesystem locations[3] in turn:

- first in any directory designated by the `XDG_CONFIG_HOME` environment variable (default value: `$HOME/.config`)
- second in any of the directories listed in the `XDG_CONFIG_DIRS` environment variable (default: `/etc/xdg`;, directories being separated by `:`)

So the main, overall US configuration file, `us.config`, is found based on this series of directories that are gone through in an orderly manner; the first directory to host such a file becomes *the* (single) US configuration directory; let's name it `US_CFG_ROOT`.

This directory, in which generally all configuration files for a US-Main deployment lie is, with our conventions, typically either `~/.config/universal-server/` (in development mode), or `/etc/xdg/universal-server/` (in production mode).

The other US-related configuration files (e.g. any `us-main.config`) are then referenced directly from the main one (`us.config`), and are designated from there through specific keys, here `us_main_config_filename`; the associate value may be either an absolute path, or one relative to the aforementioned US configuration directory (which is recommended).

US-Common Configuration

So the general, user-specified configuration file for US servers is named `us.config`, and the content of this example us.config should be pretty self-explanatory.

Note that, as US-Main supports an `epmd_port` entry in its own configuration file and has its own default port, such an entry should not be set in the general `us.config` file, as its value would not be taken into account anyway - it would just be misleading.

[2] The second best approach being to have directly a look at the code reading them; for `us.config`, see class_USConfigServer.erl.

[3] As explained notably in class_USConfigServer.erl.

Setting User and Group

For operational use, we recommend to create at least a US specific user (to be set in the `us_username` entry of one's `us.config` file) - and possibly additionally a US-Main specific one, in order to better compartmentalise the accesses to the US-Main resources.

For example, provided that a US-specific group (named for instance `us-group`) has already been created (otherwise, do it with `groupadd us-group`), a `us-main-user` user can be defined, and made belonging to that group:

```
# Only sftp allowed if relying on a rssh (package available on the AUR) shell,
# or prefer just /bin/false:
#
$ useradd --create-home --shell /bin/rssh -g us-group us-main-user
# (if wanting to change afterwards: 'chsh --shell /bin/false us-main-user')
```

If having a regular user (e.g. named `bond`) who shall be also able to operate on US-Main deployments (e.g. to monitor their traces locally), it can be added to that group as well: `usermod -a -G us-group bond` (if this user is logged in, this will not be effective before the logout of this user or the user runs `newgrp us-group`).

Finally, the `us.config` US-Common configuration file and the US-Main one it references shall have the right user and group; for example:

```
$ chown us-main-user:us-group /etc/xdg/universal-server/*.config
```

Permissions of Configuration Files

Note that, as these files might contain sensitive information (e.g. Erlang cookies), they shall be duly protected and, in terms of permissions, we should have `640` for them.

So, supposing that the `us.config` designates, in its `us_main_config_filename` entry, `foobar-us-main-for-production.config` as the name of the US-Main configuration file[4], we could have:

```
-rw-r----- 1 us-user      us-group [...] us.config
-rw-r----- 1 us-main-user us-group [...] foobar-us-main-for-production.config
```

`chmod` and `chown` shall thus be used accordingly.

Finally, if being able to rely on multiple, different US configuration directories is necessary in order to be able to launch multiple US-Main instances (these configurations can then register the managers involved - typically the US-Main Configuration server - under different names), only a single *node* can be used per EPMD instance as, by default, such a node is named `us_main` (so two US-Main instances would then clash at this level).

As a consequence, in order to run multiple US-Main instances (e.g. one in production, one for testing) on a given host, they shall be multiplexed on

[4] They should better be in the same `US_CFG_ROOT` directory (discussed above), and may be symbolic links.

different EPMD ports (see the `epmd_port` key of the US-Main configuration file; as a result, different `universal-server` configuration directories shall then be relied upon).

Specifying the US-Main Configuration File

In `us.config` , an `us_main_config_filename` entry can be specified in order to designate the US-Main configuration file that shall be used; for example:

```
{us_main_config_filename, "us-main-for-tests.config"}.
```

US-Main-level Configuration

This US-Main configuration file designated by `us.config` concentrates the settings of all the facilities presented above; it is additionally used by the US-Main scripts, notably in order to start, stop, kill, monitor, control a designated US-Main server.

Here also, proceeding based on examples in the simplest approach[5], thanks to the comments they include.

Refer for that at this US-Main example.

Test Configuration Files

Versions of the US and US-Main configuration files may be used for testing purposes and/or for examples.

For example, as root:

```
$ cd /etc/xdg/universal-server/
$ ln -s $US_COMMON_ROOT/priv/conf/us.config
$ ln -s $US_MAIN_ROOT/priv/conf/us-main-for-tests.config
```

Of course in the general case one will devise one's configuration files according to one's context and needs.

Specifying a US-Main Specific EPMD Port

For US-Main, the default EPMD port is `4507` (preferred to the default, more general US-level EPMD port, `4506`).

This helps compartmentalising the nodes, and then US-Main (possibly including its EPMD daemon) can be restarted without affecting the other Erlang applications that may be running concurrently.

The user may specify an EPMD of their liking, providing it neither clashes nor is filtered by the local firewall. This can be done by specifying in the US-Main configuration file (instead of in `us.config`) an `epmd_port` entry.

For example:

```
{epmd_port, 4666}.
```

[5] The second best approach being to have directly a look at the code reading them, see, for the US-Main configuration, class_USMainConfigServer.erl.

Communication Gateway

The communication gateway will rely on Ceylan-Mobile to send SMS. For that, a suitable 3G device (typically a USB key) will have to be used.

As mentioned in this section, proper permissions must apply, so that the user (e.g. `us-main-user`) running US-Main is able to interact with the 3G device (e.g. `/dev/ttyUSB-my-3G-key`).

So `gpasswd -a us-main-user uucp` can be executed as root (and possibly `us-main-user` may run `newgrp uucp` to update its group identification).

Installing US-Main: Current Stable Version & Download

As mentioned, the single mandatory prerequisite of the Universal Server is US-Common, which relies on Ceylan-Traces, which implies in turn Ceylan-WOOPER, then Ceylan-Myriad and Erlang.

We prefer using GNU/Linux, sticking to the latest stable release of Erlang (refer to the corresponding Myriad prerequisite section for more precise guidelines[6]), and building the Universal Server from sources, thanks to GNU `make`.

The Universal Server `master` branch is meant to stick to the latest stable version: we try to ensure that this main line always stays functional (sorry for the pun). Evolutions are to take place in feature branches and to be merged only when ready.

Automated Installation & Native Deployment (recommended)

This procedure relies on our *native* build/run system, which is the only one that is now officially supported in the context of US-Main, and involves only two steps (prepare/deploy), on the target host.

Step #1, preparation, is to:

- create a `/etc/xdg/universal-server` directory in which a relevant US configuration file (`us.config`) is added, referencing a suitable US-Main configuration file (e.g. `foobar-us-main.config`) located in the same directory; refer to the Configuring US-Main section
- ensure that our deploy-us-main-native-build.sh script in available on that host (in a US-Main clone, it is located in `priv/bin`), possibly simply at the root of one's normal user
- ensure that no prior server is running at the target port(s) (otherwise specify the `--no-launch` option below); to be sure that no previous US-Main instance lingers, one may execute `systemctl stop us-main-as-native-build.service` and/or run our `priv/bin/kill-us-main.sh` script

[6] Note that servers like US-Main have a machinery meant to be executed as root before switching to an application-specific user. As a result, Erlang is best available system-wide (as opposed to only a regular, non-privileged user).

After deployment, the installation itself will be located by default in the `/opt/universal-server/us_main-native-deployment-YYYYMMDD` (timestamped) directory.

Step #2, deployment, just consists in executing the aforementioned `deploy-us-main-native-build.sh` script, as a normal user, and with any relevant option (list them with `-h` or `-help`).

If having specified the `--no-launch` option, then once deployed refer to Launching US-Main - otherwise a launch attempt has been done.

Alternatively to step #2: Using Cutting-Edge Git

This is more or less a manual, more limited version of the US-Main deployment script.

Once Erlang is available, it should be just a matter of executing:

```
$ git clone https://github.com/Olivier-Boudeville/Ceylan-Myriad myriad
$ cd myriad && make all && cd ..

$ git clone https://github.com/Olivier-Boudeville/Ceylan-WOOPER wooper
$ cd wooper && make all && cd ..

$ git clone https://github.com/Olivier-Boudeville/Ceylan-Traces traces
$ cd traces && make all && cd ..


# Possibly:
$ git clone https://github.com/Olivier-Boudeville/erlang-serial
$ cd erlang-serial && make && DESTDIR=. make install && cd ..

$ git clone https://github.com/Olivier-Boudeville/Ceylan-Oceanic oceanic
$ cd oceanic && make all && cd ..


# Also possibly:
$ git clone https://github.com/Olivier-Boudeville/Ceylan-Seaplus seaplus
$ cd seaplus && make all && cd ..

$ git clone https://github.com/Olivier-Boudeville/Ceylan-Mobile mobile
$ cd mobile && make all && cd ..


$ git clone https://github.com/Olivier-Boudeville/us-common
$ cd us-common && make all && cd ..

$ git clone https://github.com/Olivier-Boudeville/us-main
$ cd us-main && make all && cd ..
```

Rebar3-based Build

Note

With rebar3 we encountered *a lot* of difficulties regarding build and release. So, at least for the moment, we dropped the use of rebar3 and focused instead on our native build/run system, which is perfectly suitable and fully sufficient for our needs. We do not plan to restore the rebar3 build anymore (contributions are welcome though - but be aware that the dependency management is bound to be tricky).

One may prefer relying on rebar3, even if it is by far the less frequent approach taken here.

If wanting to be able to operate on the source code of the dependencies, appropriate symbolic links may be defined in a `_checkouts` directory created at the root of `us-main`, these links pointing to relevant Git clones (typically sibling ones).

We do not use anymore the rebar3 support, but left the elements that were used in that context, for further reference.

Using OTP-Related Build/Runtime Conventions

As discussed in these sections of Myriad, WOOPER, Traces and US-Common, we added the (optional) possibility of generating a Universal Server *OTP application* out of the build tree, ready to result directly in an *(OTP) release*. For that we relied on rebar3, relx and hex.

Then we benefit from a standalone, complete Universal Server.

For more details, one may have a look at:

- rebar.config.template, the general rebar configuration file used when generating the Universal Server OTP application and release (implying the automatic management of Myriad and WOOPER)
- rebar-for-hex.config.template, to generate a corresponding Hex package for Universal Server (whose structure and conventions are quite different from the previous OTP elements)

Launching US-Main

We suppose here that the US-Main configuration has already taken place (otherwise the guidelines in the Configuring US-Main section shall be applied).

The US-Main management procedure relies on switching to a configured (UNIX) user (meant to have the lowest privileges, and generally created for the sole purpose of running a given US server), which requires the corresponding US-Main scripts to be executed as root first.

We recommend running US-Main thanks to a native build (rather than as a release), and using for that the start-us-main-native-build.sh script, either directly (for testing) or through systemd (for actual use; the preferred stable solution).

In this last case, first a symbolic link pointing to this script shall be typically created in the `/usr/local/bin` directory of the host of interest. Then the server is to be triggered based on `/etc/systemd/system/us-main-as-native-build.service` (see us-main-as-native-build.service).

For example:

```
# Prepare:
$ cd /usr/local/bin
$ ln -s /opt/universal-server/us_main-latest/us_main/priv/bin/start-us-main-native-bui
$ cd /etc/systemd/system
$ ln -s /opt/universal-server/us_main-latest/us_main/priv/conf/us-main-as-native-buil

# Trigger:
$ systemctl start us-main-as-native-build
$ journalctl --lines=200 --unit=us-main-as-native-build.service --follow
$ systemctl enable us-main-as-native-build
```

Then one may run `priv/bin/monitor-us-main.sh` to browse its traces live at any time.

Like notified in the start-up message:

```
-- Starting US-Main natively-built application as user 'stallone' (EPMD port: 4507)..
       Executing application us_main_app.beam as a service (second form)
Write pipe '/tmp/launch-erl-1103261.w' found, waiting 2 seconds to ensure start-up is

 **************************************************************
 ** Node 'us_main' ready and running as a daemon.
 ** Use 'to_erl /tmp/launch-erl-1103261' to connect to that node.
 ** (then type CTRL-D to exit without killing the node)
 **************************************************************
 (authbind success reported)
```

one may use `to_erl` to connect directly; just remember that exiting the interpreter as usual (Ctrl-C twice) thus means *killing* that node; prefer Ctrl-D (once); see this section for details.

For further information one may also refer to the US-Main shell scripts, which cover various administration-related tasks (deploying, starting, monitoring, stopping an US-Main server).

Apparently, in some cases (not always), stopping the US-Main server (typically with `systemctl stop us-main-as-native-build.service`) will *not* unregister it from its EPMD, which will not be stopped either. As a consequence, any next launching of the US-Main server is bound to fail after a time-out, and the VM log file (e.g. `/opt/universal-server/us_main-native/us_main/log/erlang.log.1`) will confirm that a node with the same name (`us_main`) already exists (and thus prevents the new launch).

In this case, the best solution is to kill that lingering EPMD (e.g. `epmd -port 4507 -kill`) or, more brutally, to run the `kill-us-main.sh` script (it will kill both any US-Main and its EPMD) - and then to restart US-Main.

Troubleshooting US-Main

The recommended procedure is to inspect:

- to run first the monitor-us-main.sh for any (possibly remote) host of interest
- the result of systemctl status us-main-as-native-build.service
- the one of journalctl --pager-end --unit=us-main-as-native-build.service
- /var/log/universal-server/us-main/us_main.traces, if any
- the latest /opt/universal-server/us_main-latest/us_main/log/erlang.log.*
- any content in /var/log/universal-server/us-main/

Should the launch fail at startup before any information can be reported, the best course of action is to launch US-Main by oneself (possibly as root to avoid errors being reported regarding user and group rights).

For example:

```
# This may as well point directly to a development source tree:
$ US_MAIN_ROOT=/opt/universal-server/us_main-native-latest/us_main

$ cd ${US_MAIN_ROOT}/src

$ make us_main_exec
```

The information reported on the console may suffice, notably thanks to the error reports; otherwise one may inspect the content of ${US_MAIN_ROOT}/priv/for-testing/log/us_main. for further information.

For deeper troubleshooting, the goal is to mimic the behaviour of us-main-as-native-build.service, i.e. the action of the start-us-main-native-build.sh script.

So it can be run, as root, to further investigate.

A common cause of failure, if the launch time-out triggered, is having already an instance of US-Main running, thus preventing any new launch thereof. This may be checked for example thanks to ps -edf | grep beam.smp | grep us_main.

Monitoring US-Main

Local Monitoring

Based on the US-Main EPMD port that was preferred, checking whether an instance is running is as simple as:

```
$ export ERL_EPMD_PORT=4507 ; epmd -names
epmd: up and running on port 4507 with data:
name us_main at port 50002
```

The various logs may also be looked at.

Then, if needed, executing kill-us-main.sh will kill any live US-Main instance and unregister it from its EPMD (without killing any EPMD daemon).

Remote Monitoring

The priv/bin/monitor-us-main.sh script shall be run for that.

Of course it must be told the target host, EPMD port, cookie, etc.; this information shall be held by a configuration file, by default named us-main-remote-access.config.

As an example:

```
% This is a configuration file to enable the interaction with a remote US-Main
% instance (e.g. a production server to be reached from a client host),
% typically to monitor its traces.

% The hostname where the US-main instance of interest runs:
{ us_main_hostname, "hurrivan" }.

{ remote_vm_cookie, 'e08a-baa4-bcb5-6b44edbd603b-hurricane' }.

% If ever it was overridden for the server:
%{ us_main_config_server_registration_name, xxx_us_main_config_server }.


% A range of TCP ports that may be used for out-of-band inter-VM communication
% (not using the Erlang carrier; e.g. for send_file):
%
% (this range must be typically in-line with any one set at the level of
% firewalls)
%
{ tcp_port_range, { 40000, 45000 } }.
```

Support

Bugs, questions, remarks, patches, requests for enhancements, etc. are to be reported to the project interface (typically issues) or directly at the email address mentioned at the beginning of this document.

Licence

The Universal Server is licensed by its author (Olivier Boudeville) under the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of this license, or (at your option) any later version.

This allows the use of the Universal Server code in a wide a variety of software projects, while still maintaining copyleft on this code, ensuring improvements are shared.

We hope indeed that enhancements will be back-contributed (e.g. thanks to merge requests), so that everyone will be able to benefit from them.

Credits

Many thanks to David Alberto for his kind sharing in terms of computation of latitude-based daylight durations (in French).

Please React!

If you have information more detailed or more recent than those presented in this document, if you noticed errors, neglects or points insufficiently discussed, drop us a line! (for that, follow the Support guidelines).

Ending Word

Have fun with the Universal Server!