Unattended Broadcasting System
User Documentation

Nik Reiman - nik@aboleo.net

October 30, 2003

Contents

Chapter 1

Introduction

The Uninterrupted Broadcasting System (UBS for short) is a tool that enables radio stations to broadcast in realtime without a physical DJ being present in the studio. Due to the constraints of smaller radio stations not being able to have live DJ's on the air 24 hours a day, the UBS provides a good solution to continue broadcasting during these hours. The UBS was originally created for WMHD, 90.7 FM, which is the radio station hosted by Rose-Hulman Institute of Technology (http://www.rose-hulman.edu, http://wmhd.rose-hulman.edu). The original concept of the UBS was proposed as a senior group project, and was implemented in Java/Tomcat with a MySQL backend. Unfortunately, the complexity and size of this project became prohibitive to its completion by the original group, and due to time constraints, this version of the project was re-written in C.
Currently, the UBS is capable of performing all tasks necessary for FCC radio compliance, as well as performing shows that are enjoyable to listen to. This includes picking a good variety of music, keeping to a strict show schedule, playing station identification tags at the top of every hour, and logging of everything played on the air. The UBS can also be configured to perform a number of other tasks useful in running a radio station, such as song requests, and special events. Many features of this sort are not directly implemented by the UBS, but the UBS it provides an easy frontend for other scripts or programs to do so. The UBS system currently runs under the Unix operating system family, and has been tested on the following platforms:

- Linux
- Mac OS X (10.2)

Future ports will include:

- Microsoft Windows NT/2K/XP

Operating systems that are not planned for support include:

- Solaris (poor sound support/sound hardware)

- Mac OS 8-9 (lack of development tools)
- Microsoft Windows 9x (obsolete)
- SCO UnixWare (licensing issues)

Be sure to check the release notes for the version of the UBS, as they may indicate changes in supported platforms. The UBS is also known to compile on BSD, but has not been extensively tested on this platform.
The UBS is constituted of three main daemons, each with a separate set of responsibilities:

ubs-sched Schedules music files to be played. This daemon is the largest and most complex of the three. It has no part in actually playing anything on the air; rather, it reads in a schedule of shows, and formulates a weekly lineup divided into 30-minute blocks. When this daemon detects that the music queue is depleted, it will add more files to the queue from the appropriate show.

ubs-play Takes music files given by ubs-sched, and plays them on the air. This daemon does not actually play the files, however. It instead pulls files from the queue (provided by the ubs-sched daemon), and plays them with an external media player, such as sox, mpg123 or ogg123.

ubs-event Runs special events for the UBS by way of executing individual event modules. This includes anything from cleaning up files to playing particular files on the air. In the case of the later, the ubs-event daemon can send a signal to the ubs-play daemon, interrupting it from reading the next file in the queue and instead playing a file determined by an event module. This is necessary for playing regular events, such as public service announcements or station identifications.

Since the UBS was designed with modularity in mind, a particular station configuration is not confined to these three program daemons. User contributed modules can run alongside the default daemons, performing other tasks such as process monitoring, web-based status, or anything else necessary to the station's requirements. A number of user-contributed modules are located in the UBS source distribution (see chapter, "Installing the UBS"), but be cautioned that these daemons may not be supported, and may also cause the UBS to malfunction.

Chapter 2

Installing the UBS

Currently, the UBS is only distributed as a source tarball, which means that the target system must have the standard set of development tools (gcc, make, ld) to build software by hand. The basic installation process for the UBS is as such:

1. `./configure`
2. `make`
3. `make install`
4. Optionally build and install extra modules in the `contrib` directory of the source tree (see the file `contrib/README` in the source distribution about available modules).
5. Create and edit the `ubs.conf` configuration file

Although binary distributions of the UBS are planned in the future, they will probably not be available until the codebase has reached a greater level of maturity. By default, the UBS is installed into `/usr/local/ubs`. If a different installation path is desired, specify this by passing `--prefix=PATH` to the configure script.
After building the UBS, install the executables by running `make install`. This will create the UBS prefix directory, as well as several subdirectories. By default, the following directories are created underneath the prefix directory:

bin Where the actual UBS binaries go. By default, the main UBS control program is simply called "`ubs`", and any daemons are in the format "`ubs-name`", where "name" describes the function of the module. If custom modules are being used, it is important that they follow this convention so they are visible to the main UBS control program.

etc Location for configuration files. Upon a default installation, two files are placed in this directory: `ubs.conf.default` and `help.conf`. Do not edit

`help.conf`, as it is used by the UBS shell for its help system. However, a syntactically correct copy of `ubs.conf` must be found in this directory in order for the UBS to start. It is recommended to copy the default file to `ubs.conf` and edit the file according to the particular site configuration (see chapter "Configuring the UBS").

include Directory for UBS header files. Only needed for building UBS modules (see chapter "Writing UBS Modules" for more information).

lib Place for the UBS libraries, which are currently built as statically linked files. Although it is not essential for these libraries to be installed in the system, they are necessary for building any extra modules using the UBS API.

log Where the UBS logs all activities. By default, only two logging targets are specified, one for songs played on the error, and another for debugging information. However, daemons are not confined to logging to these files, and their logging locations will generally be specified in the `ubs.conf` file. It is advised to keep all log files in this directory so that the UBS shell has access to their contents.

tmp Storage location for temporary files, such as cached data, and other scratch files created by UBS daemons.

var Information about the UBS daemons is stored here, such as the queue, the process ID's of the running daemons and their respective status, and the song currently being played on the air.

Note that actual music directories are not accounted for here. The UBS is capable of serving music from any location on the system, as long as it has read permissions to these files. However, it is not necessary to place music files for the UBS underneath the installation tree.
The UBS source distribution also comes with other daemons or event modules (used by the ubs-event daemon) that may be handy in setting up the UBS. While these modules are not essential to the operation and setup of the UBS, they may be useful to some users. These events are not built by default with the `make` command, and must be built by hand. This can be done by entering the `contrib` directory of the UBS source tree, and running `make` there. Currently, the list of contributed modules includes:

Daemons

ubs-monitor Checks to see if the UBS daemons have crashed or malfunctioned, and sends alerts via pager or email.

Events

pick_song Picks a random filename from a directory of music files. Useful for station ID or similar events.

Also, the UBS comes with a variety of graphical frontends that help control the various daemon processes. While the UBS comes with a command-line interface capable of controlling the daemons (see chapter "Running the UBS"), graphical user interfaces may be easier to operate for many users. Also, some interfaces may help the UBS to be accessible via the web, for easy remote access. These frontends are not built by default, and are stored in the `frontends` directory of the UBS source tree. Currently included frontends are:

php A set of PHP functions that allow the UBS to be controlled from a remote web browser.

Future frontends that are planned include:

ncurses A console based replacement for the shell, with a bit more interactivity.

gtk X-based GTK client

cocoa OS X client

Chapter 3

Configuring the UBS

Before the UBS can be run, it needs to have a configuration file which specifies default settings for the system. Unless the location to such a file is specified from the command line (see chapter, "Running the UBS"), this file is `PREFIX/etc/ubs.conf`. If this file does not exist, or contains an error, the UBS will fail to start. A default configuration file is included with the UBS, and is installed as `ubs.conf.default`. This file can be renamed to `ubs.conf`, and edited as necessary. The file is well commented, and should be fairly straightforward.
The configuration file has a relatively simple format. Settings are specified, one per line. Any line beginning with the '#' symbol is treated as a comment, and ignored by the UBS. All other lines are in the following format:

```
context.subcontext = "argument"
```

Here, the "context" describes the section of the particular setting. The `global` context is read by all daemons, and each daemon can have its own context, which is suitable for particular settings not applicable to other daemons. The "subcontext", in turn, describes the particular parameter that is to be set. For instance, the configuration line:

```
play.media = "mpg123 -g 0 -q \%s"
```

will set the "media" variable for the "play" context (which is parsed by the ubsplay daemon) to be the program "mpg123" with the arguments given above. In particular, the song's filename will also be substituted in for "%s". Note that most things in the configuration file can be left commented out, in which case the UBS will resort to the built in default settings. The following variables are recognized by the UBS's global context:

errorlog Where to log program errors. (Default: "log/error.log")

loglevel Determines how much information is to be sent to the error logs. Each loglevel also encompasses all levels less than its value, which means

that the "error" level will also print out messages marked as "status" and "emergency". (Default: "2") The recognized loglevels are:

0 Emergency messages and critical errors only.

1 Status messages.

2 Non-fatal error messages.

3 Debugging information. This level will produce a great deal of logging output and is recommended for developers only.

prefix Where the UBS looks for its relevant subdirectories. Only used upon program initialization. (Default: "/usr/local/ubs")

queue Location of queue file for songs to play. (Default: "var/queue")

songlog Where to log songs that have been played on the air. (Default: "log/playlist.log")

The following commands are recognized by the "play" context, which is read in by the ubs-play daemon:

media The name of the media player to use for all files. Accepts substitution of "%s" for the song filename. (Default: "ogg123 -q %s")

killmedia Determines whether or not to stop the media player upon program exit. Acceptable values are "yes" or "no". If this value is not set, when a "stop" command is issued, the UBS itself will terminate, but it will leave the song playing on the air until it is finished. This can be useful for stations that want to avoid sudden stops, but it also means that a sudden restart of the UBS may play two songs on the air at once. If killmedia is set to be on, then the media player will be immediately stopped when ubs-play exits. (Default: "no")

The `ubs.conf` file also contains the schedule of shows, which is parsed by the ubs-sched daemon. This *must* be specified in the configuration file, or else the UBS will not run. In order to set up a schedule of shows, the following configuration option needs to be set:

```
sched.numshows = "?"
```

Where '?' corresponds to the number of total shows. Each show must then be specified with the following format (not necessarily in this order):

```
showN.name = "name"
showN.start = time
showN.end = time
showN.repeat = "frequency"
showN.dir = "directory"
showN.type = "type"
```

Where 'N' is the show number, starting with '1'. If any one of these values are left blank, the show will be considered invalid and the scheduling daemon will not run. For each one of the shows, the context is simply the word "show" combined with the particular number of the show, starting at the number 1. Thus, if three shows are given in `sched.numshows`, then they would be given contexts of `show1`, `show2`, and `show3`, respectively. The order that these shows are given doesn't matter, but all three shows must be completed at some point in the configuration file.
Each show may have the following parameters, as specified above:

banner (Optional) If specified, then the argument given will be queued as the first song when the show rotates on the air. This can be helpful for having a certain announcement be played to announce an upcoming show.

end Time the show is to stop playing. Same format as start time.

dir The directory which to use for music for this show. This will also recursively include music in all subdirectories of this directory. This can also be the name of an M3U playlist, if the show is to be playlisted (see the "type" variable).

name A brief description of the show. Used only for informational purposes; can be any string.

repeat How often the show is to repeat. Only the "daily" and "weekly" frequencies are currently supported. If the show is to be weekly, then the "when" directive must also be used.

start What time the show is supposed to start, in 24-hour military time. The time format can be given in the format HH:MM, or simply HH. Currently, the UBS schedules shows in even 30 minute blocks.

type Type of show to schedule. Value can either be "random", or "m3u". In the case of the former value, the "dir" parameter must point to a directory tree containing music files, which will be queued at random. For type "m3u", the "dir" parameter must be the name of a m3u playlist, which will then be queued up exactly as it appears.

when (Only needed if show is "weekly") Specifies what day of the week the show should occur. Recognized values are "sun", "mon", "tues", "wed", "thurs", "fri", and "sat".

Other directives for the sched context include:

miscdir Location to queue music from if no valid show is found for the current timeslot. (Default: "misc", which is a subdirectory of the UBS prefix)

searchback How far back to look in the logfiles for potential matches. For example, if this value is set to 10, then no two songs will be queued up by the same artist, song title, or filename until at least 10 other songs have been played. (Default: 30)

For events to be used by the ubs-event daemon, a similar structure is used to that of the shows. In the configuration file, the following directive must be set:

```
event.numevents = "?"
```

Where '?' again corresponds to the number of events that will be declared in the configuration file. The format of events is similar to that of shows; each event is given a context in the format of "eventN", where 'N' is the number of the particular event. Each event must contain the following parameters:

```
eventN.name = "program arguments"
eventN.time = time
eventN.logfile = "logfile"
```

As in the show configuration, an event must have all of the above parameters to be a valid. Subcontexts for the particular events are:

logfile Where to log success of the event. Note that this is not by default a file in the logging directory, and this must be specified with a path as "`log/name.log`" to be stored in the UBS's logfile location.

name The event to run. Unlike the name given for shows, this corresponds to the event to run. The program must be the name of an executable located in `PREFIX/bin`. Arguments to this program are optional, but permitted.

time When to run the event. If given as a flat number (such as "30"), the event will be played every hour at that time. Otherwise, a specific time can be given (such as "12:30"), and the UBS will only play that event at 12:30pm, once a day.

Note that the UBS configuration file allows for additional, user-defined contexts. This can be very useful when writing custom modules. Note that most UBS daemons simply only parse for the "global" context and their own respective context. Additional contexts can be safely ignored.

Chapter 4

Running the UBS

Once the UBS has been successfully installed and configured, it can be launched in several different ways, from either a particular graphical frontend or the included command line utility. At the time of writing this draft, no fully functional graphical frontends exist for the UBS, so this document will focus primarily on the UBS shell. The UBS shell is a command line interface, similar to that of a unix command line interface. Although the UBS shell lacks much of the functionality in most unix shells, it is a sufficient interface for controlling various parts of the UBS.
To start the UBS shell, simply call the UBS executable from a regular unix command line. A transcript of this might look as such:

```
shell$ cd /usr/local/ubs
shell$ ./bin/ubs
Welcome to the UBS.  Use the 'help' command for more info.
ubs-0.15>
```

The shell currently recognizes approximately a dozen commands. For a complete list of commands, type `help` at the shell's command prompt. Each individual command also has its own specific help, which is available by running `command -h`. Each command also recognizes a number of different output formats, which are useful in various situations. The most commonly used output formats are:

-e Regular English output. This is the default output format.

-w HTML output. Useful for webpages, or web-based scripts accessing the UBS shell.

-c Comma separated value output. Also useful for scripting.

-s Silent; no output is produced.

To start the UBS daemons, the `start` command is used. The start command may take in the name of a single UBS daemon, in which case it will start only the

one respective daemon. If no arguments are given, the shell will start all known daemons, which is anything in the form of `ubs-name` in the `bin/` directory of the UBS installation. An example session of starting and stopping the UBS might look something like this:

```
ubs-0.15> start
ubs-event: Started
ubs-sched: Started
ubs-play: Started
ubs-0.15> stop
ubs-event: Stopped
ubs-sched: Stopped
ubs-play: Stopped
```

Other commands recognized by the UBS include:

events Displays information about all known events as specified in the ubs.conf file.

help Print out more help about a particular command. If no specific command is given, all known commands are printed out.

log Gives information about logfiles, and can also display the tail portion of a particular logfile. The log command must be passed the name of a logfile, which can be acquired from the "-l" option. Known arguments include:

- -n Number of lines to seek back in the logfile.
- -l List all known logs.

queue Displays information about the queue, and can also add files to or clear out the current queue. Known arguments include:

- -d Display the current queue (default action).
- -a Append the filename given after this argument to the end of the queue.
- -p Prepend the filename given after this argument to the front of the queue.
- -z Clear out all entries in the queue.

restart Issues a stop and start command to all known daemons if no arguments are given, or a single daemon specified on the command line.

sched Shows the schedule for the current week, and what shows are to be played in which timeslots.

search Frontend to the UBS search engine, which is capable of locating filenames based on querying ID3 tags in all known music files. Will return all files which match either by filename or metadata (ogg/id3 tag) based on a query given from the command line. Known arguments include:

-r Recache the data for the search engine. This must be done on a regular basis, since the UBS caches the music metadata in a single file, rather than searching all music files each time a query is done.

shows Displays information about the schedule of shows. Known arguments include:

-n Only display information for the show currently playing on the air.

-p Display information for the song currently playing on the air.

start Starts the UBS daemons (all if no argument is given, a single one otherwise), launching them into the background. Note that if the shell is exited, the daemons will continue to run.

status Displays the current status of each daemon.

stop Stops the UBS daemons or a single daemon if given as an argument.

tag Prints out information from ID3/Ogg Vorbis tags in music files.

version Show the current version of the UBS.

exit Quit the UBS. This will *not* terminate any UBS daemons that are running.

Chapter 5

UBS Troubleshooting

Coming soon.

Chapter 6

Writing UBS Modules

One of the design features of the UBS is that new program modules can easily be added to the system. There are fundamentally two types of modules that can be built for the UBS; standalone daemons and events. The former refers to code that helps the UBS run, and remains running as long as the UBS is active. Events, on the other hand, need not continue running. Rather, they are launched by the UBS (generally by the ubs-event module) to perform a specific task, and exit when that task is completed. Although the program structure of a daemon and a module are very similar as far as the UBS is concerned, this document shall provide an example of each.

6.1 UBS Daemons

The UBS only imposes a few rules on what it considers to be a valid daemon. The module name must be prefixed with a "ubs-" (which is how the shell finds all the program modules it should launch on startup), and must initialize itself with the functions provided in UBS library. A sample daemon might look something like this:

```
/* ubs-sample.c - A sample UBS daemon. */

/* Needed for most UBS functions */
#include "ubs.h"

int main(int argc, char *argv[]) {
 /* Need to initialize the global table.  If any local
    configuration tables are used, they must also be
    initialized here. */
 ubs_table_init(&GLOBAL);

 /* Define the program name, and set up default global
    variables. */
```

```
 ubs_init(argv[0]);

 /* Read the ubs.conf file into the global table. */
 read_config(DEF_CONFIG, "global", &GLOBAL);

 /* See if this daemon is already running, and exit if
    it is, as not to spawn two copies of it. */
 if(check_running() == FAIL) {
  return FAIL;
 }

 /* Steps needed for proper daemonization. */
 if(fork()) {
  exit(0);
 }
 setsid();
 chdir(ubs_table_data(&GLOBAL, "prefix"));
 umask(0);

 while(1) {
  /* Main algorithm goes here. */
 }

 return OK;
}
```

To build this program, compile by running:

```
gcc -o ubs-sample -IPREFIX/include ubs-sample.c PREFIX/lib/libubs.a
```

where `PREFIX` is the installation path of the UBS. This will build a program called `ubs-sample`, which can be copied to `PREFIX/bin`, where it will be launched with all the other UBS modules on program startup.
The above program simply initializes itself, reads the UBS configuration file, and exits. The steps taken by this program are as follows:

1. Initialize all UBS table structures. This must be done before writing any data to any ubs_table. If the program has additional variable tables (for reading its own contexts in the ubs.conf file), these tables must also be initialized here.

2. Call ubs_init with the program name, which is stored in argv[0]. This sets the global table up with the default set of variables (in case any variables are left undefined in the ubs.conf file), as well as binds a number of signals and initializes global variables in the UBS.

3. Read in the configuration files. This is done by calling read_config, which will store all data in the table referenced in its third argument. To access

this data, the ubs_table_data function may be used. For more information about this, see the programmer's API reference.

4. Call check_running to see if a module by this name is already running. If so, it is probably a bad idea to launch another one.

5. To be a "proper" unix daemon, a few steps must be done to launch the process into the background. By forking, and killing the parent, the child process is left in the background, and the user is returned to the command prompt. After forking off, the child process (now the UBS main process) must call setsid to be the process leader, and chdir to the UBS root directory. Setting the umask to 000 is also a good practice. At this point, UBS initialization is complete, and the daemon can run uninterrupted.

6.2 UBS Event Modules

Now, a more complicated example of a sample event is given, that will demonstrate basic use of the UBS library functions:

```
/* sample-event.c - A sample UBS event. */

#include "ubs.h"

int main(int argc, char *argv[]) {
 /* Table for variables specific to this event context. */
 ubs_table sample;

 /* Initialize tables. */
 ubs_table_init(&GLOBAL);
 ubs_table_init(&sample);

 /* Initialize program. */
 ubs_init(argv[0]);

 /* Read ubs.conf for global context. */
 if(read_config(DEF_CONFIG, "global", &GLOBAL)) {
  /* If the configuration file can't be read for whatever reason, print
     out an error and bail. */
  console_error("Could not read global context in configuration file", FAIL);
 }
 /* Read ubs.conf for the "sample" context.  In this case, users can
    define extra variables that will only be read by this program, in
    the form of sample.variable = value.  These variables will be safely
    ignored by the rest of the UBS. */
 if(read_config(DEF_CONFIG, "sample", &sample)) {
  console_error("Could not read sample context in configuration file", FAIL);
```

```
}

/* This module will only have one recognized context, which will be
   sample.text, and will contain a message that will be printed out to
   the UBS debug log.  If this is not defined, then print out an error
   warning, but otherwise continue program execution. */
if(ubs_table_exists(&sample, "text")) {
 log_error_msg(LOG_DEBUG, "Message from sample event: '%s'",
   ubs_table_data(&sample, "text"));
}
else {
 ubs_error_msg(LOG_ERROR, "'text' variable undefined"");
}

 return OK;
}
```

In this example program, a specific configuration variable is read from the ubs.conf file, and printed out to the UBS system logs. In order for a module to define its own context, all it needs to do is call read_config, with the second parameter being the name of the context to parse (which is "sample", in this case). This data is stored in a ubs_table structure, which must be initialized before the call to read_config.
This program also demonstrates how logging is handled by the UBS. Although the UBS provides mechanisms to write to specific logfiles, with custom data fields, for most debugging and error messages a simple call to ubs_error_msg will suffice. This function takes in two arguments, the first being the logging priority (which can be LOG_EMERG, LOG_STATUS, LOG_ERROR, or LOG_DEBUG, for emergency, status, error, and debug. respectively). The second parameter is a format string followed by any arguments which are to be substituted into the string. Currently, the only recognized data types for the format string are '%s' for a string, and '%d' for an integer.

Chapter 7

Conclusion

Should any problems or bugs be found with the UBS, I encourage reports to be emailed to me at nik(a)aboleo.net, or submitted online at the UBS bug reporting website, which is located at http://www.aboleo.net/software/ubs/bugs. Thanks for using the UBS.