Harc: Home Automation and Remote Control

Table of contents

1. Revision history

Date	Description
2009-07-18	Initial version.

2. Introduction

Since 2006 I have been writing software, designed file formats, and classified remote control signals, revolving around infrared remote control and home automation. It presently consists of around 18000 lines of source code in Java and XML. Very much code has been junked or re-written. Since I do not have an immediate goal, and my possibility to work on the project is limited, I have decided to make the outcome available.

As a working project name, as well as in the Java module names, I have been using the name *Harc*, which is simply an acronym for "Home Automation and Remote Control". Unfortunately, the name is far too generic to register as an Internet domain. There is even a Sourceforge project named Harc, (inactive since 2002).

It is proposed to call a user or developer of the system an harc-eologist.

The present document is aimed more at a high-level description of the system, rather than being a user's manual. It describes most aspects of Harc at most very roughly.

Warning:

This is unfinished, experimental software, aimed at the expert user. "Non-programmers" will probably not find anything of use herein. No warranty of any kind is given.

3. Overview of the system

The "system" consists of a number of data formats, and Java classes operating on those formats. It has been the goal to formulate file formats allowing for a very clean description of infrared protocols, the commands of devices to be controlled (in particular, but not exclusively audio- and video equipment), networking components, topology of an installation, as well as macros. From these very general universal formats configuration files in other formats, used by other systems, can automatically be generated.

There is also a quite universal GUI that gives access to most of the functionality within the Java classes. This has been written to demonstrate the possibilities for the expert user.

Convenience and accessibility for the novice user has not been a priority. (Such user interfaces are conceivable, however.)

Directly supported communication hardware and software are GlobalCache GC-100, IRTrans LAN (only the LAN version is supported directly), RS232 serial communication through the GlobalCaché or ser2net, TCP sockets, HTTP, RF 433 and 868 MHz through EZcontrol T10 as well as through an IR->RF translator like Conrad Eggs or Marmitek pyramids. Indirectly, through LIRC, a vast number of IR senders are supported.

4. Data model

Next a somewhat technical description of the file formats will be given. These are all defined in the form of XML files with a rather strict DTD. The discussion to follow will focus on the concepts, not on the details. Of course, the semantics of the files are defined by the DTD file.

Necessary theoretical background on IR signals can be found for example in this article.

4.1. The command names

Harc has higher requirements on the data quality of its input files than other related projects. For example, in LIRC, a configuration file consists of a device name, and a number of commands with associated IR signals. The names of the commands there are in principle completely arbitrary, and it appears to be common to try to follow the vendor's naming. In Harc, there is instead a fixed (but of course from the developer extensible) list of command names, intended to uniquely denote a command for a device. A command name in principle is a verb, not a noun, and should describe the action as appropriate as possible. There is, for example, no command `power`, instead there are three commands: `power_on`, `power_off`, and `power_toggle` (having the obvious semantics). Also, a command which toggles between play and pause status may not be called `play`, but should be called `play_pause`.

The names are defined in the XML file `commandnames.xml`, from which a Java enum `command_t` is created through an XSLT style-sheet `mk_command_t.xsl`. Further rules for command names are found as comments in that file.

4.2. The protocol files

By "(infrared) protocol" we mean a mapping taking a few parameters (one of those a "command number", one a "device number", sometimes others) to an infrared signal. A protocol file is an XML file describing exactly how the IR signal is made up from the parameters. The format is quite close to an XML version of the "IRP notation" of the JP1-Project. It is a machine readable description on how to map the parameters into a modulated infrared signal, consistent with a technical description. The protocol is identified by its name. A protocol takes a certain number of parameters, although

sometimes one is defaulted.

At his point, the reader may like to compare this prose description of the protocol we (and the JP1 project) call `nec1` with the XML code in `nec1.xml`. Note that our description, using an arbitrary subdevice number, corresponds to the author's "Extended Nec protocol".

The naming of the different protocols is of course somewhat arbitrary. In general, I have tried to be compatible with the JP1-Project.

It can be noted that the supported radio frequency protocols are nothing but IR-signals with the carrier consisting of infrared 950nm light substituted by suitable radio carrier, typically of 433 MHz.

Ideally, most users should not have to worry with the protocol files. This is only necessary when introducing a device with a IR-protocol that has not yet been implemented. At the time of this writing the 17 protocols have been implemented, this covers the most important ones, but not all known to e.g. the JP1 project.

4.3. Device files

A device file is an XML file describing the commands for controlling a device. In Harc a device file truly describes the device and its commands, stripped from all information not pertaining to the very device, like key binding on a remote, button layout, display name, the IR blaster it is connected to, location, IP-address, MAC-address, etc. (This is in contrast to many other systems, like Pronto CCF-files or JP1 device updates).

There may be many different types of commands for the device, like IR, RF signals (this is, at least for the few cases presently supported, nothing else but IR signals with the infrared light as carrier replaced by an radio signal, for Europe of 433 or 868 MHz frequency), commands over serial RS232 interfaces or TCP sockets, or utilizing a WEB API. Also "browsing" the device (pointing a Web browser to its www server), pinging and sending WOL-packages are considered commands, as well as suppling power, sometimes "in reverse direction" (like a motorized blind going up or down). Possibly the same command can be issued over different media. Some commands may take arguments or deliver output. For this (and other) reasons, care should be taken to use the "correct" command names, not just a phrase the manufacturer found cool. Commands are grouped in *commandsets*, consisting of commands having its *type* (ir, serial, tcp,...), device number etc in common.

IR signals within a device file may contain codes in Pronto CCF format in addition (or instead) if the structured information (protocol, device number, command number etc). Actually, "exporting in XML format" means generating an XML file augmented with the raw CCF codes. In may cases, also so-called cooked Pronto codes (Background, written by remotecentral) are included, as well as JP1 protocol information.

The device configuration file is processed by an xinclude-aware parser, allowing a certain

include-file structure, that can be used to structure the data.

4.3.1. Example

As an example, consider the Oppo DV-983HDVD player with serial support. This is supported by Harc with the file `oppo_dv983.xml`. Its commands can be downloaded directly from the manufacturer (hats off!), both the infrared and the serial commands. As can be found in the spreadsheet on the IR code, the device uses the previously mentioned `nec1` protocol, with device number equal to 73. This corresponds to the first command set in the mentioned device file. The serial commands form another commandset, subdivided into *commandgroups*, depending on whether they take an argument and/or deliver output. Note that some commands (for example `play`) are available both as IR and as serial commands.

Other interesting examples are the `*_dbox2.xml` files (referring to the German dbox with the open source tuxbox software) , each containing two (`sagem_dbox2.xml`, `philips_dbox2.xml`), or three (`nokia_dbox2.xml`) different infrared command sets as well as an elaborate web-api command set. Another very interesting example is the Denon A/V-Receiver `denon_avr3808.xml` having several infrared command sets using the `denon` protocol (which, ironically, is called the "Sharp protocol" by the firm Denon), as well as several command sets using the `denon_k` (Denon-Kaseikyo protocol). Then there is a large number of "serial" commands, available through both the serial box as well as through the (telnet) tcp port 23.

4.3.2. Importers

Since Harc is so picky with command names and their semantics, the value of an import facility is limited — necessary information is simply not there (or is wrong). There exists a large number of IR signal data in the Internet (for example from LIRC configuration files, JP1 device updates, or the large collection (mainly CCF) of files on Remotecentral. Presently, Harc has "importers" for Pronto/CCF and JP1's device upgrades in RemoteMaster format. I "sort-of" wrote a LIRC-to-CCF translator a few years ago, possibly I will finish it someday. However, the importers have as their goal to create a first iteration of a device file (not even guaranteed to be valid XML!) to be tweaked manually.

4.3.3. Exporters

Writing an exporter is in principle easier. Harc presently can export the IR signals of a device in CCF format, LIRC-format (either a particular device, or all devices connected to a particular LIRC server defined in the home file), JP1's device upgrades in RemoteManager format, as well as the rem-files used by IRTrans. Individual IR-signals can be exported in wav-format for usage with an audio output driving an IR LED after full wave rectification, see for example this article This feature is presently not available through the GUI.

Many other things are possible. I have had some success creating a program that, given an XML configuration file, creates a full JP1-type image that can be flashed on a URC-7781 (that is, not just one or a few device updates).

4.4. The "home file"

The protocol and device files described up until now are a sort of universal data base — common and invariant to every user, at least in principle. In contrast, the "home file" (possibly the name is not very well chosen) describes the individual setup ("home"). It is a good idea to think of the device files as class definitions, classes which can be instantiated one or more times, in that one or more devices of the same class are present in the home configuration, each having its individual (instance-)name.

It is instructive to have a look at the supplied file `home.xml` at this point. In the home file the different devices are defined as class instances. They can be given alternate names (aliases) and groups can — for different purposes — be defined. For example, this can be useful for generating GUIs taking only a certain group of devices into account. Gateways are defined: a gateway is some "gadget" connecting some other media together, for example, the GlobalCache (among other things) connects the "tcp connector" on the local area network (`lan`) to its output connectors, which may be e.g. an infrared blaster or stick-in LED controlling an IR-device. Devices that can be controlled declare the said device/connector combination as a "from-gateway", or indirectly as a from-gateway-ref (using the XML idref facility). (Yes, there are a lot of details here which ideally sometime should be described in detail.) Thus, a routing is actually defined: how to send commands to the device in question. Note that there may be many, redundant, paths. The actual software is actually using this redundancy, thus implementing a certain failure-safeness. The actual from-gateways, and their associated paths, are tried in order until someone succeeds in sending the command. (Of course, only to the extent that transmission failures can be detected: non-reachable Ethernet gateways are detected, humans blocking the way between an IR-blaster and its target are not...).

Also the interconnection between AV devices can be described here, see the example. Thus, it is possible to send high-level input selecting commands like "turn the amplifier `my_amplifier` to the DVD player `my_dvdplayer`", and the software will determine what command (IR or other) to send to what device. (This is later called "select mode".)

There is a great potential in this concept: Consider for example a "Conrad Egg transmitter", which for our purposes is nothing but IR->RF gateway. Assume that a IR stick-on emitter is glued to the egg, and connected to a Ethernet -> IR gateway. If there is, say a RF controlled Intertechno switch, interfacing with an electric consumer, it is possible to just issue the command for turning the electric consumer on or off, and the software will find out that it has to send the appropriate IR signal to the IR gateway.

However, writing the configuration file is a job for the expert...

4.5. Macros

A simple macro facility has been implemented. This is presently in the form of one XML file, see the example file `macros.xml`. The syntax was inspired by Lisp.

Macros may call commands (XML element `command`), wait (`delay`), "print" a message (`message`), call other macros (also recursively) (`macrocall`), use select-mode (`select-src`) and contain conditionals (`cond`, using the syntax of the Lisp `cond`). The recursion facility eliminates the need for a loop construct.

For the convenience of programs acting on the macro file, macros can be bundled into groups (XML element `macros`). They may be "public" or "private", the latter can only be called from other macros.

Macros can be executed by the macro engine (`macro_engine.java`). However, also other usages are possible: Being all XML, XML transformations are conceivable which generate all sort of output, such as creating shell scripts, generating HTML pages, or macro definitions for smart remote controls like Netremote (using its extension language Lua).

Macros with arguments are presently not implemented.

5. Basic Java classes

There is a large number of Java classes operating on the data objects. Some classes operates on protocols, some on device classes (through device files), some on device instances in the sense of the home file. In most cases when it is sensible to call use the class individually, it contains a `main`-method, i.e. can be called from the command line. In general, there are a number of arguments. A usage message can be generated in the usual GNU way, using `--help` as argument.

6. Program usage

The main entry point in the main `jar`-file is called `Main`. Its usage message reads:

```
        harc --version|--help
        harc [OPTIONS] [-g|-r|-l [<portnumber>]]
        harc [OPTIONS] <macro>
        harc [OPTIONS] <device_instance> <command> [<argument(s)>]
        harc [OPTIONS] -s <device_instance> <src_device_instance>
where OPTIONS=-A,-V,-M,-h <filename>,-t
ir|rf433|rf868|www|web_api|tcp|udp|serial|bluetooth|on_off|ip|special,-m
<macrofilename>,-T 0|1,-# <count>,-v,-d <debugcode>,-a <aliasfile>, -b
<browserpath>, -p <propsfile>, -z <zone>,-c <connection_type>
```

Using the `-g` (as well as no argument at all, to allowing for double clicking the `jar`-file) starts Harc in GUI mode, described in the next section. Invoking Harc with the `-r`, `-l`

portnumber starts the readline and port listening mode respectively. Otherwise Harc will run in non-interactive mode, executing one command or macro, and then exit.

6.1. Non-interactive mode

If there is only one argument, it is considered to be a macro, and it is attempted to execute it. (Using "?" as argument, the available macros will be listed.) The `-s` option enables the select mode, described previously. Otherwise, the arguments are considered as a device instance name, followed by a command name, and optionally by arguments for the command. If the command name is missing or "?", the possible command names for the device will be listed.

The remaining options are as follows:

-A
switch only audio on target device (if possible)
-V
switch only video on target device (if possible)
-M
use so-called smart-memory on some devices
-h *home-filename*
use *home-filename* as home file instead of the default one
-m *macro-filename*
use *macro-filename* as home file instead of the default one
-t *type*
prefer command of type *type* regardless of ordering in home file (if possible)
-T *zero_or_one*
for codes with toggles (like RC5), set the toggle value to the argument.
-v
verbose execution.
-d *debug code*
set debug code. See `debugargs.java` for its precise meaning. Use -1 to turn on all possible debugging.
-a *aliasfile*
Normally, aliases (allowing the software accept e.g. "enter" and "select" as synonyms for "ok") are taken from the official `command.xml`. This option allows the usage of another alias file.
-b *browserpath*
Allows using an alternative path to the browser used to invoke `browse`-commands, instead of the default one.
-p *propsfile*
Allows using an alternative properties file, instead of the default one.

The following options apply only to the select mode

-z *zone*

Select for zone *zone* (if possible)
-c connection
Prefer connection type *connection* for the selection (if possible)

6.2. Readline mode

The "Readline mode" is an interactive command line mode, where the user types the commands one at a time. If GNU readline is available, the extraordinary facilities of GNU readline allows not only to edit present command and to recall previous commands, but also for an intelligent completion of relevant names for macros, devices, and commands. If GNU readline is not available, Harc's "readline mode" will still work, only these "comfort features" are missing. The semantics of the typed command are like the non-interactive arguments. There are also some extra commands, introduced by "--"; these in general correspond to the command line options described above. The normal command line options are ignored.

6.3. Port listen mode

Starting Harc in port listening mode starts a multithreaded server, listening for commands on the TCP port given as argument, default 9999. The server responds to a connection on that port and spawns off a new thread for each connection. It listens to commands on that port, sending output back. The semantics of the command line sent to the server is the same as for the non-interactive invocations, with the addition of the commands `--quit`, which makes the session/thread close, and `--die`, which in addition instructs the server not to spawn any more threads, but to die when the last session has ended.

6.4. The GUI

The present GUI was not designed for deployment. It does not offer a user friendly way for allowing a nontechnical user to control his home or home theater. Rather, the goal was a research-type GUI, to allow the expert user to access to most of the functionality of the Java classes, without having to look in the Javadoc class documentation.

Hopefully, in the near future, there will be one or more "cool" GUIs for the system. This need not be additions to the present system, but rather integrations with other technologies and projects, like Openremote.

The main properties of the present GUI will be described next.

The GUI consists of a title bar with pull-down menus for File, Edit, Options, Misc., and Help. These are believed to be more-or less self explanatory. There are six panes, that will be described in order. Many interface elements have a short tool-text help messages, which are displayed when the cursor is hoovering above the element. The lower part of the main window is occupied by "the console". The latter is a read-only "pseudo paper roll console", listing commands, tracing- and debugging information as directed by the user's selections, as well as command output and error messages.

Except for the mandatory about-popup (which is of course non-modal!), popups are not used.

The GUI resides almost completely within the file `gui_main.java`. It was designed using the Netbeans IDE version 6.5.

6.4.1. The Home/Macros pane

This pane corresponds to using Harc through the Home configuration file. Devices, using their instance names as defined in the home configuration file are sent commands, possibly with *one* argument, possibly returning output in the console. (Commands taking two or more arguments cannot be sent through the GUI.) The first row is for sending commands to devices, the second for the select mode, while the third one can execute macros. Note that both the execution of macros and of commands are executed in separate threads.

This pane is the only one coming close to "deployment usage". The other panes can be useful for setting up a system, defining and testing new devices or protocols, or for "research usage".

6.4.2. The Device classes pane

This pane allows for sending *infrared* signals (no other command type!) to the using either a GlobalCache or an IRTrans, that has been selected using the "Output HW" pane, including output connector to use. The home configuration file is not used. The devices are called by their class names.

6.4.3. The IR Protocols pane

This pane has more of research character. For a protocol in the protocol data base, a device number, possibly a subdevice number, and a command number is entered, pressing the "Encode" button causes the corresponding IR code in Pronto CCF format to be computed and displayed. Pressing the send button causes the code to be sent to a GlobalCache or IRTrans that was selected in the "Output HW" pane. Note that it is possible to hand edit (including pasting from the clipboard) the content of the raw code before sending. Whenever there is content in the raw code text area, the decode button can be invoked, sending the content to the DecodeIR library, thus trying to identify an unknown IR signal (if possible).

Log files from the Irscope program (using `.icf` as their file extension) can be imported using the `icf` button.

There presently appears to be some "glitches" in the button enabling code; click e.g. in the "raw code" text area to enable buttons that are incorrectly disabled.

6.4.4. The Output HW pane

This pane has three subpanes: GlobalCache (for selecting the GlobalCache, and its output connector, used on the Device classes and on the IR Protocols pane), IRTrans (dito), and EZControl. The latter is sort of an interactive toolbox for the EZcontrol T10, allowing to send different commands, inquiry the status of one or all of the preselected switches, as well as getting a list of its programmed timers.

6.4.5. The IRcalc panel

This pane is a sort-of spreadsheet for computing IR signals in the Pronto or JP1 context. The exact way it works is left as an exercise for the reader...

6.4.6. The Options panel

This pane allows the user to set a few more options. On-off options are sometimes instead available through the Options pull-down menu.

6.5. Properties

Harc uses a properties file in XML format. For some of the properties there is no sensible access in the GUI. For this reason, it may therefore sometimes be necessary to manually edit this file with a text editor (or XML editor).

7. Interaction with other projects

HARC interacts with other projects within the area. It can be pointed out that in the case of Java projects, Harc uses unmodified jar-files; in the case of shared libraries (`.so` or `.dll`) these are also used in an unmodified state. In no case, Harc "borrows" code from the projects. Also, in this way additional functionality is implemented, none of which is of essential (like import/export of a certain file format). Differently put: should the need arise to eliminate "derivedness", only minor, nonessential functionality will be sacrificed (or needs to be implemented anew).

7.1. LIRC: Linux InfraRed Control

LIRC is a well established, mature, and active free software project. It is used in very many free software projects. It contains support for a large number of infrared senders and receivers, some sane hardware designs, other possibly less sane. There are also a large number of user contributed configuration files for different IR remote controls and devices, in general consisting of leaned commands. A network enabled LIRC server consists of the software running on a host, listening on a network socket, containing one

or more IR transmitter or transmitter channels. A client sends requests to, e.g., transmit a certain command for a certain device type. Since Harc can talk to a network LIRC server (see source in the file `lirc.java`), there is a large number of IR senders that Harc in this way "indirectly" supports. Unfortunately, the configuration files are residing on the LIRC server only; there is no way to request the transmission of a signal the server does not know in its data base. (A patch for this was submitted by myself, but rejected by the maintainer. I plan to make it available on my web server.) From its IR data base, Harc can generate configuration files for LIRC. There is presently no possibility to import LIRC files.

Presently, there is no support for selecting output channels on LIRC servers with multiple IR devices or IR channels. (This can probably be fairly easily implemented using the LIRC server command `SET_TRANSMITTERS` and the `connector` attribute in the `home.xml` file.)

LIRC is licensed under GNU General Public License, version 2 or later. However, Harc is not a derived work; it contains no LIRC code, and is not linked to any libraries. It optionally "talks" to a LIRC server, but this functionality is entirely optional.

7.2. JP1

The JP1 project is a very unique project. It aims at complete control over the remotes made by Universal Electronics (UEIC), which include the brand names "One For All" and "Radio Shack". Through careful study of its hard- and firmware, techniques for custom programming a remote, equipped with a 6-pin connector (on the remote's PCB called "JP1", giving the project its name) was developed. Thus, an off-the-shelf remote can be taken much beyond its original capacities. Most importantly, it can be programmed from a computer to generate "arbitrary" IR-signals.

RemoteMaster is a program for, among other things, creating so-called device updates. These device updates can be produced by Harc, as rmdu-exports. Thus, for an IR-controlled device in the Harc database, a suitable JP1-enabled remote control can be made to send the appropriate IR-signals. (There are some details, that will be documented somewhere else.) RemoteMaster is presented as an interactive GUI program, however, it can also (although this is not supported) be called through a Java API. Harc presently uses version 1.89, which is not the current version. Although it seems to lack all copyright notices, it is referred to as "open source" and GPL.

Another tool from the JP1 project is DecodeIR by John Fine, available under the GNU General Public License, version 2 or later. It consists of a shared library (`DecodeIR.dll` or `DecodeIR.se`) , written in C++, together with a Java wrapper (`DecodeIR.jar`). To build that jar file, also this file is needed. The tool attempts to decode an IR-signal in CCF form into a well known protocol with device number, command number, possibly subdevice number and other parameters. See the IR protocols pane in the GUI.

7.3. IRScope

Together with appropriate hardware, the Windows program IRScope by Kevin Timmerman is very useful to "record", and optionally analyze unknown IR-signals (again, using the same DecodeIR as above). The log files generated by this program can be directly parsed, see the code in `ict_parse.java` or the IR protocols pane. The program is licensed under GNU General Public License, version 2 or later. Harc neither uses code or links to it, and is not a derived work.

7.4. Tonto

Tonto is normally referred to as an alternate configuration ("CCF") editor for the first generation of the Philips Pronto remote controls. It is a free replacement for the original ProntoEdit program, written by Stewart Allen, licensed under the GNU General Public License, version 2. Being written in Java, it runs "everywhere", in contrast to the original Windows-only program. It also contains a Java API. Harc uses the Tonto API (in the form of the file `tonto.jar`) to import CCF files, and to generate CCF files for devices in its data base. (The latter are supposed to be more of a target for aliasing, than a directly usable user interface.) Unfortunately, the project is (essentially) inactive since 2004.

7.5. wakeonlan

Harc uses wakeonlan (licensed under the GNU Lesser General Public License), a small Java library for implementing WOL-functionality.

7.6. Java Readline

The interactive command line uses the Java Readline (licensed under the GNU Lesser General Public License), which of course only makes sense when used in conjunction with the GNU Readline library, which is licensed under GNU General Public License, version 2 or later.

8. Future development

The big missing piece is of course the complete lack of a "pretty" user interface.

Minor improvements:

- Replace the DTD based file formats with scheme based. This should be a real migration to more intelligent formats, not just a syntax change.
- Implement support for multi-transmitter LIRC servers.
- Eliminating the need to configure a LIRC server by finishing and publishing the LIRC patch described above, and making corresponding changes to `lirc.java`.
- Scripting facility, for example with Rhino/Javascript. Should this replace or complement the present macro engine?

Presently, I do not want to commit myself to maintaining Harc. The goal is not, and has never been, to present a full solution for (for example) home automation, but rather to produce some reusable data exchange formats, and some useful tools, that fit with other projects. Nevertheless, I welcome both bug reports, bug fixes as well as enhancements. I also welcome new configuration files for devices and protocols.

I look forward in particular to a cooperation with the Openremote Project.

9. Downloads

Download Harc! Relevant third-party libraries in jar-format are included.

A version of `libJavaReadline.so` for 64-bit Linux can be downloaded here. Precompiled dynamic libraries `libreadline.so` and `libhistory.so` should be available from any Linux distributor, package name probably `readline` or so. Precompiled `libDecodeIR` (version 2.36) can be downloaded for Windows, 32-bit Linux x86, and 64-bit Linux x86. Note that these components are needed only to enable some special functionality of Harc. There is no need to get them just to try things out. Corresponding sources can either be downloaded from the Internet using URLs published in this page, or (as required by the GPL) requested from me.