HARCToolbox

Table of contents

1 Home....3
1.1 Welcome to the HARCToolbox project!....3
1.2 NEWS!!....7
1.3 Interaction with other projects....8
1.4 Impressum....10
1.5 Legal....11
2 Current....11
2.1 Current program and articles....11
2.2 IrScrutinizer documentation....12
2.3 IrpMaster: a program and API for the generation of IR signals from IRP notation....36
2.4 The Girr format for universal IR Commands and remotes....51
2.5 General InfraRed Server Command Language....57
2.6 Architecture Concept....63
2.7 Glossary and terms....69
2.8 HarcHardware....78
2.9 Jirc documentation....79
2.10 Improved LIRC driver for the Raspberry Pi....81
2.11 Raspberry Pi daughter board for IR and RF transmission and reception....84
2.12 Enabling LIRC to send CCF signals not residing on the server....88
3 Old programs and docs....90
3.1 Discontinued projects and old articles....90
3.2 IrMaster documentation....91
3.3 Using and Transforming XML export....108
3.4 Harctoolbox: Home Automation and Remote Control....113
3.5 lirc2xml: Extracting information from LIRC configuration files....126

4 API Documentation.. 130
4.1 HarctoolboxBundle 1.1.3 API.. 130
4.2 Girr 1.0.3 API.. 130
4.3 HarctoolboxBundle 1.1.3 API.. 130
4.4 HarctoolboxBundle 1.1.3 API.. 130
4.5 HarctoolboxBundle 1.1.3 API.. 130
4.6 HarctoolboxBundle 1.1.3 API.. 130
5 Downloads.. 130
5.1 HARCToolbox downloads.. 130
6 All.. 134
6.1 Site Linkmap Table of Contents... 134

1 Home

1.1 Welcome to the HARCToolbox project!

Date	Description
2011-01-10	Initial version. Creation of domain www.harctoolbox.org.
2012-05-01	Incorporated IrpMaster and IrMaster as sub projects, just as lirc2xml (moved from my personal home site) and the LIRC CCF patch (published for the first time). General reconstruction of site. Release of version 0.1.2 of IrpMaster, version 0.1.2 of IrMaster, version 0.7.0 of Harctoolbox (unfinished...), as well as the version 0.1.2 of lirc2xml.
2012-06-07	Updated for the release of version 0.2.0 of IrMaster and IrpMaster. The document on transforming XML new.
2012-08-19	Updated for the release of version 0.3.0 of IrMaster and release 0.2.1 of IrpMaster.
2012-11-18	Updated for the release of version 0.3.1 of IrMaster and release 0.2.2 of IrpMaster.
2014-02-02	Reorganized, for version 1.0.0 of IrScrutinizer, etc.
2014-06-12	Updated for the new releases: version 1.1.0 of IrScrutinizer, version 1.0.1 of IrpMaster, IrMaster, and Girr, version 0.3.0 of Jirc, version 0.9.1 of HarcHardware, version 0.2.1 of GuiComponents.
2014-09-27	Updated for the new releases: version 1.1.1 of IrScrutinizer, version 1.0.2 of IrpMaster, IrMaster, and Girr, version 0.9.2 of HarcHardware, version 0.2.2 of GuiComponents.
2015-04-16	Updated for the new releases: version 1.1.2 of IrScrutinizer, version 1.0.3 of IrpMaster, IrMaster, and Girr, version 0.9.3 of HarcHardware.
2015-09-10	Updated for the new releases: version 1.1.3 of IrScrutinizer.

Table 1: Revision history

> **Note:**
> IrScrutinizer 1.1.3 released. Sources now hosted on Github

1.1.1 History

Since 2006 I have been writing software, designed file formats, and classified remote control signals, revolving around infrared remote control and home automation. The old "main project" (old harctoolbox) is the original project. It deals with descriptions of IR protocols and signals, device descriptions, and descriptions of their interconnection.

In early 2011 I came to the conclusion, that my work on IR protocols was a dead end street. It was better to connect to and use the Internet knowledge, notable the so-called IRP notation, that was excellently formalized by Graham Dixon in early 2010. This started the IrpMaster project, in which the IRP notation was completely implemented (with one exception, which is rather a specification flaw than an omission of practical importance).

For IrpMaster, the goal was correctness and completeness, without any compromises for user friendliness. In particular, it does not contain a graphical user interface. Its command line interface appears as inaccessible to the GUI-centered user. Instead, the project IrMaster was started, to allow a user friendly, interactive access to the functionality of IrpMaster. As a graphical "meta program", it also fulfills some other tasks, see its documentation.

Early 2013 I planned to extend IrMaster with more functionality, in particular the possibility to capture IR signals and to import IR signals from many other sources and file formats. Originally, I planned to extend IrMaster with this new functionality; however, I decided to start on a brand new program, IrScrutinizer, which basically combines almost all that fits in an IR program. One of the reasons was that I was unhappy with the somewhat suboptimal code structure of IrMaster. The first version was released on November 2013. IrMaster was declared as no longer developed, promoted to version 1.0.0, which was released in February 2014.

In the light of these projects, the main project harctoolbox has gone slightly "stale", and it needs to be restructured.

The program *lirc2xml* was posted on the LIRC mailing list in 2009 (without any response :-), and has been residing on my private web page since then. For IrScrutinizer, I wanted this functionality incorporated. A natural way would be to turn lirc2xml to a shared library with JNI interface. However, discouraged by the experiences from DecodeIR, I wanted a pure Java solution, and ported the part of LIRC needed to parse and interpret the configuration files to Java. This is the Jirc project.

LIRC CCF: The idea to modify the LIRC server to allow for directly sent CCF signals was originally sent to me in private mail from Bitmonster (of Eventghost). I made a preliminary version some time later, this was communicated to him and submitted to the then-maintainer Christian Barthelmus, who promptly rejected it (some more details in the documentation). This is its first real "publishing".

There are presently over 80000 lines of source code in Java on this site, code in other languages, or by other authors not counted.

The present work is copyrighted by myself, and available under the GNU General Public License version 3. In the future, it may also be available under additional conditions, so-called dual licensing. (If interested in a commercial license, please contact me.) File formats are in the public domain, including their machine readable descriptions, i.e. DTD and schema files.

As a working project name, as well as in the Java module names, the name *Harc*, which is simply an acronym for "Home Automation and Remote Control", was used. Unfortunately, that name is far too generic to register as an Internet domain. There is even a Sourceforge project named Harc, (inactive since 2002). For this reason the project was renamed to *HARCToolbox* early 2011, and the domain www.harctoolbox.org created.

1.1.2 Overview

IrpMaster is a very advanced renderer of infrared signals. That is, from an abstract description consisting of a protocol name, and some parameter values, it computes the corresponding infrared signal, as a sequence of gaps and flashes. It is intended to be used both from its API, as well as an interactive (command line) program, both offline (generating files etc) as well as in real-time applications (generating signals to be sent to hardware equipment). The program does not attempt to be "user friendly", and does not, e.g., contain a graphical user interface.

IrMaster's main purpose is to be a user friendly user interface to IrpMaster. But it does not end there: it also integrates/interfaces with other projects, like the alternative, "classical" IR renderer Makehex, the DecodeIR "inverse renderer", the analyzeIR library, the Harctoolbox main project etc. It can generate export files in different formats, it can address networked hardware directly (presently GlobalCaché, IRTrans LAN module, and a patched LIRC server. (Please note that *IrMaster* and *Irpmaster* are two different projects. IrMaster depends on IrpMaster, but not vice versa.)

IrScrutinizer is a powerful program for capturing, generating, analyzing, importing, and exporting of infrared (IR) signals. For capturing and sending IR signals several different hardware sensors and senders are supported. IR Signals can be imported not only by capturing from one of the supported hardware sensors (among others: IrWidget, Global Caché, and Arduino), but also from a number of different file formats (among others: LIRC, Wave, Pronto Classic and professional, RMDU (partially), and different text based formats; not only from files, but also from the clipboard, from URLs, and from file hierarchies), as well as the Internet IR Databases by Global Caché and by IRDB. Imported signals can be decoded, analyzed, edited, and plotted. A collection of IR signal can thus be assembled and edited, and finally exported in one of the many supported

formats. In addition, the program contains the powerful IrpMaster IR-renderer, which means that almost all IR protocols known to the Internet community can be generated.

There are some support projects, independent in the sense of a separate package. First, there is the subproject HarcHardware, containing a number of classes for hardware access. This is believed to be of interest also for other projects. Secondly, the subproject GuiComponents (no documentation page present!) contains a number of, in principle, recyclable components, often in the form of Java Beans. Although it was written and conceived as a support project for IrScrutinizer, it is not unlikely that they can be used directly in other programs.

As described in the introduction, the main project is presently slightly "stale".

In the context of the LIRC project, there is a large body of recoded infrared signals in "LIRC format". For many users of home automation and remote control, it would be desirable to use this knowledge outside of the LIRC framework. Unfortunately, the LIRC format (`lircd.conf`) is is all but well documented. The program lirc2xml therefore uses LIRC itself to decode and export its signals, here to (somewhat arbitrarily) an XML file, that can be further processed.

Finally, the LIRC patch for CCF signals describes an enhancement to the LIRC server.

1.1.3 Content

- IrScrutinizer, very powerful general purpose IR program.
- IrMaster, a predecessor to IrScrutinizer.
- IrpMaster, the IR renderer.
- Glossary and terms.
- HarcHardware, hardware related functionality.
- Girr, a specification of a general exchange format for IR signals
- Generic IR Server, a specification of a general IR server API
- HarcToolbox, the old main project.
- Transforming XML Export from Ir(p)Master, a tutorial article on generating "interesting stuff" (here, C code) from the XML export. Applies to IrpMaster and IrMaster, not IrScrutinizer, thus somewhat obsolete.
- Jirc, LIRC configuration files in Java.
- Lirc2xml, a program for extracting IR codes from LIRC files. Predecessor to Jirc, thus obsolete.
- Improved lirc driver for the Raspberry Pi.
- Raspberry Pi daughter board for IR and RF transmission and reception.
- LIRC CCF patch, to make the LIRC server more usable.
- Relationship to other, similar projects.
- Downloads.

1.1.4 General background links

- SB-Projects' IR Remote Control Theory. A very good place to start leaning about IR protocols. Covers the basics of infrared signals, and its modulation. Not too long, not too "dummy"-oriented.
- JP1 wikiMany articles, of different ages and qualities, are collected here, in a clearly arranged way.

1.2 NEWS!!

1.2.1 Latest news!

New versions of IrScrutinizer 1.1.2! Accompanying are new versions of Girr, IrpMaster, HarcHardware, GuiComponents.

Sources now hosted on Github. Many thanks to Alec Leamas of Fedora for support and discussions on the project reorganization.

1.2.2 History

2011-01-10
Initial version. Creation of domain www.harctoolbox.org.

2012-05-01
Incorporated IrpMaster and IrMaster as subprojects, just as lirc2xml (moved from my personal home site) and the LIRC CCF patch (published for the first time). General reconstruction of site. Release of version 0.1.2 of IrpMaster, version 0.1.2 of IrMaster, version 0.7.0 of Harctoolbox (unfinished...), as well as the version 0.1.2 of lirc2xml.

2012-06-07
Updated for the release of version 0.2.0 of IrMaster and IrpMaster. The document on transforming XML new.

2012-08-19
Updated for the release of version 0.3.0 of IrMaster and release 0.2.1 of IrpMaster.

2012-11-18
Updated for the release of version 0.3.1 of IrMaster and release 0.2.2 of IrpMaster.

2014-02-02
Whole site reworked. New programs IrScrutinitzer, Jirc. New libraries Girr, HarcHardware. New hardware support Improved Lirc driver for Raspberry Pi. New articles Girs IR server specification, description of a IR/RF daughter board for the Raspberry Pi, and a Glossary. New versions of IrpMaster (1.0.0) and IrMaster (1.0.0).

2014-06-12
New versions of IrScrutinizer, IrpMaster, Girs, Jirc.

2014-09-27
New versions of IrScrutinizer, IrpMaster, Girs, GuiComponents, HarcHardware.

2015-04-16
New versions of IrScrutinizer, IrpMaster, Girs, GuiComponents, HarcHardware.

1.3 Interaction with other projects

1.3.1 Revision history

Date	Description
2011-01-09	Initial version, derived from old `harc.xml`.
2014-02-02	Update.

1.3.2 Introduction

HARCToolbox interacts with other projects within the area. It can be pointed out that in the case of Java projects, Harctoolbox uses unmodified jar-files; in the case of shared libraries (`.so` or `.dll`) these are also used in an unmodified state. In no case, Harctoolbox "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).

Also see the copyright notices in the individual programs/projects, where more third-party software is listed.

1.3.3 LIRC: Linux Infrared Remote 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 Harctoolbox can talk to a network LIRC server (see source in the file `lirc.java`), there is a large number of IR senders that Harctoolbox 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. It is available here.

From its IR data base, Harctoolbox can generate configuration files for LIRC. IrScrutinizer, using Jirc, which is a translation of LIRC to Java, LIRC files can be imported and translated to other formats.

LIRC is licensed under GNU General Public License, version 2 or later. However, Harctoolbox 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.

1.3.4 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 Harctoolbox, as rmdu-exports. Thus, for an IR-controlled device in the Harctoolbox 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. Harctoolbox 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.

IrScrutinizer can "almost" import and export RemoteMaster's device upgrade files. Since this is dependent on some rather intricate protocol dependent transformations between the JP1-specific "EFC"-numbers and the function codes, I consider it as a better idea to extend RemoteMaster to import and export the general Girr format of IrScrutinizer.

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.

1.3.5 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. Harctoolbox neither uses code or links to it, and is not a derived work.

IrScrutinizer can replace IrScope.

1.3.6 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. Harctoolbox and IrScrutinizer use 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 inactive since 2004.

1.3.7 IRDB

IRDB is a web site that describes itself as "one of the largest crowd-sourced, manufacturer-independent databases of infrared remote control codes on the web, and aspiring to become the most comprehensive and most accurate one". IrScrutinizer can directly import a set of IR signals from this data base. Interestingly, the site uses software from this site extensively.

1.3.8 wakeonlan

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

1.3.9 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.

1.3.10 Sunrise/Sunset

Harctoolbox uses the sunrise/sunset Java library (see also Blog) by LuckyCat Labs/Mike Reedell (licensed under the Apache License 2.0). (which is considered as "compatible" with GPL3). This is a Java library for computing sunrise and sunsets. I have implemented some improvements, which I intend to publish shortly.

1.4 Impressum

Responsible for this site is Dr. Bengt Martensson, webmaster@harctoolbox.org. With the exception of my other two sites (www.bengt-martensson.de and www.bengt-martensson-consulting.de), I am not responsible for the content of linked sites, over which I have no control. At the time of the original publishing, no illegal or otherwise questionable content on linked sites was known to me.

1.5 Legal

This site is Copyright (c) 2011, 2012, 2013, 2014 by Bengt Martensson. All rights reserved.

Software or software fragments ("patches") are published under the GPL3 license. This also applies to all XML-files on the site. File formats and their machine readable descriptions (DTD and Schemas, extension `.dtd` amd `.xsd`) are in the public domain. **Other material (text, pictures, and style elements) are not to be reused without permission of the author, but may be linked to (including "deeplinks").**

It is possible that information or downloads from this page can cause damage to your hardware, software, or anything else. In particular, sending undocumented IR commands to your equipment may damage or even destroy it. It can also not be excluded that usage or downloads, or usage of herein described software, will violate applicable laws, or agreements. By using information or downloads from this page, you agree to take the full responsibility yourself, and not hold the author responsible.

2 Current

2.1 Current program and articles

Date	Description
2014-02-02	Initial version.

Table 1: Revision history

2.1.1 Content

This page consists of a table of content of currently maintained programs, libraries, and articles.

- IrScrutinizer, a very advanced IR program.
- IrMaster, a general purpose IR program, now discontinued and superseded by IrScrutinizer.
- IrpMaster, the IR renderer, as a library and command line program.
- HarcHardware, a number of Java classes for hardware related functionallity.
- LIRC CCF patch, to make the LIRC server able to send signals not residing on the server.
- Girr (General InfraRed Remote format), a specification for a general IR signal exchange format. Also a library for its implementation.
- Girs (General InfraRed Server), a specification for a class of IR servers.
- Glossary and terms.
- Jirc, a substantial part of the Lirc project translated to Java, that reads and translates Lirc configuration files.
- Raspberry Pi Improved Lirc driver.

- Raspberry Pi daughterboard of IR and RF sending and receiving.

Downloads.

2.2 IrScrutinizer documentation

> **Note:**
> This is a reference manual. It is written for completeness and correctness, not for accessability. For an easier introduction, see this tutorial.

> **Warning:**
> Sending undocumented IR commands to your equipment may damage or even destroy it. By using this program, you agree to take the responsibility for possible damages yourself, and not to hold the author responsible.

Date	Description
2013-11-12	Initial version.
2013-12-01	Next unfinished version.
2014-01-22	Version for release 1.0.0, still leaving much to be desired.
2014-06-07	Version for release 1.1.0, some fixes, much still to be done.
2014-09-21	Version for release 1.1.1, some more fixes and enhancements.
2015-04-07	Version for release 1.1.2, installation issued reworked. Misc. minor improvements.
2015-08-19	Version for release 1.1.3. Misc. minor improvements.

Table 1: Revision history

2.2.1 Introduction

IrScrutinizer is a powerful program for capturing, generating, analyzing, importing, and exporting of infrared (IR) signals. For capturing and sending IR signals several different hardware sensors and senders are supported. IR Signals can be imported not only by capturing from one of the supported hardware sensors (among others: IrWidget, Global Caché, Command Fusion, and Arduino), but also from a number of different file formats (among others: LIRC, Wave, CML, Pronto Classic and professional, RMDU (partially), and different text based formats; not only from files, but also from the clipboard, from URLs, and from file hierarchies), as well as the Internet IR Databases by Global Caché and by IRDB. Imported signals can be decoded, analyzed, edited, and plotted. A collection of IR signal can thus be assembled and edited, and finally exported

in one of the many supported formats. In addition, the program contains the powerful IrpMaster IR-renderer, which means that almost all IR protocols known to the Internet community can be generated.

Written in Java (with the exception of two native libraries), most of the functionality of the program is available on every Java platform. The native libraries (DecodeIR and RXTX) are presently available for 32- and 64-bit versions of Windows, Linux (x86 and amd-64), and MacOsX, and can with moderate effort be compiled for other platforms.

For someone with knowledge in the problem domain of IR signals and their parameterization, this program is believed to be simple to use. This knowledge is assumed from the reader. Other can acquire that knowledge either from the JP1 Wiki or, e.g., this link.

Note that screen shots are included as illustrations only; they may not depict the current program completely accurately. They come from different versions of the program, using different platforms (Linux and Windows), and using different "look and feels".

Sources are hosted on Github. Bugreports are enhancement requests are welcome (e.g. as issues), as is contributions (code, documentation, use cases, protocols etc.).

The present document is written more for completeness than for easy accessibility. For an easier introduction, see this tutorial.

Here are the current release notes.

2.2.1.1 Background

In 2011 I wrote an IR signal "engine" called IrpMaster. It can also be invoked as a command line program. Then a program called IrMaster was released, which among other things constitutes a user friendly GUI front end to IrpMaster. The present program, IrScrutinizer, is also based on IrpMaster, and adds functionality from IrMaster, in particular the possibility to collect IR signals, a vastly improved import and export facility, and edit collections of IR commands. IrScrutizer almost completely replaces IrMaster. The final version was released in Febrary 2014, slightly ironically called version 1.0.0. No further development is planned.

2.2.1.2 Copyright and License

The program, as well as this document, is copyright by myself. My copyright does not extend to the embedded "components" Analyze, Makehex, DecodeIR, and Jirc. ExchangeIR was written by Graham Dixon and published under GPL3 license. A subset has been translated to Java by Bengt Martensson. DecodeIR was originally written by John S. Fine, with later contributions from others. It is public domain software. IrpMaster is using ANTLR3.4 and depends on the run time functions of ANTLR3, which is free software with BSD license.

The "database file" IrpProtocols.ini is derived from DecodeIR.html, thus I do not claim copyright.

The program uses JCommander by Cédric Beust to parse the command line arguments. It is free software with Apache 2 license.

Icons by Everaldo Coelho from the Crystal project are used; these are released under the LGPL license.

The Windows installer was built with Inno Setup, which is free software by Jordan Russel. To modify the user's path in Windows, the Inno extension modpath by Jared Breland, distributed under the GNU Lesser General Public License (LGPL), version 3.

Serial communication is handled by the RXTX library, licensed under the LGPL v 2.1 license.

JSON handling is implemented using the "fast and minimal JSON parser for Java" by Ralf Sernberg, licensed under the Eclipse Eclipse Public License Version 1.0.

LIRC (Linux Infrared Remote Control) is according to its web site copyright 1999 by Karsten Scheibler and Christoph Bartelmus (with contribution of may others), and is licensed under GPL2. The parts used here have been translated to Java by myself, available here.

Tonto was written by Stewart Allen, and is licensed under the "Clarified Artistic License".

The contained Arduino firmware contains code copyrighted by Michael Dreher (IrWidget, GPL2 or later) and Chris Young (IRLib LGPL).

The program contains icons from Global Caché, Dangerous Prototypes, Arduino, and IrTrans. These are used exclusively in the context of these firms, and only used to illustrate their products. The icons for JP1 and LIRC are also exclusively used to illustrate themselves.

The program and its documentation are licensed under the GNU General Public License version 3, making everyone free to use, study, improve, etc., under certain conditions. File formats, their description in human- or machine-readable form (DTDs, XML Schemas), are placed in the public domain.

2.2.1.3 Privacy note

Some functions (Help -> Project Home page, Help -> IRP Notation Spec, Help -> Protocol Specs, Tools -> Check for updates) access the Internet using standard http calls. This causes the originating machine's IP-address, time and date, the used browser, and possibly other information to be stored on the called server. If this is a concern for you, please do not use this (non-critical) functionality (or block your computer's Internet access).

2.2.2 Overview

Next a high-level view of the different use cases will be given.

Analyze ("Scrutinize") individual IR Signal/Ir Sequence

An IrSignal or IrSequence can be captured from connected hardware, or imported from files in different formats, the clipboard, or from Internet databases. The IrSequence can be broken into a beginning-, repeat-, and ending sequence, and decoded, analyzed, and plotted. It can be exported in different formats, or sent to different transmitting hardware.

Analyze/edit/compose/export collections of IR signals (""remotes")

A collection of commands can be assembled either from individual IR signals (as above), captured several at a time, or imported from files in different formats, the clipboard, or from Internet databases. The collection and the individual commands can be edited as in a spreadsheet. It can be exported in a number of different formats.

Generate IR Signals from known protocols

IR Signals can be generated from the Internet's largest protocol data base, containing over 100 protocol. Necessary protocol parameter values are to be entered. Thus generated signals can be analyzed as single signals, incorporated into remotes, or exported to files — also from e.g. intervals of parameters.

2.2.3 Installation

2.2.3.1 General

IrScrutinizer, and all but two of its third-party additions, are written in Java, which means that it should run on every computer with a modern Java installed; Windows, Linux, Macintosh, etc. Java 1.7 or later is required. The two exception are DecodeIR and the native part of RXTX, which are written in C++ and C respectively, and invoked as shared library (`.dll` in Windows, `.so` in Linux, `.jnilib` or `.dylib` in Mac OS X). If DecodeIR or RXTX are not available on for a particular platform it is not a major problem, as most parts of IrScrutinizer will work fine without it; just the DecodeIR-related functions or the serial hardware access will be unavailable.

In all versions, IrScrutinizer behave civilized, in that the installation directory can be read only after the installation.

As of version 1.1.2, there are five different way of installing the program, described next.

2.2.3.2 Fedora Linux

Fedora rpm packages are available for the previous version 1.1.2.

To install, the command `sudo yum install harctoolbox harctoolbox-doc` should be issued in a terminal window. This will install IrScrutinizer as a desktop program, as well as the command line commands `irscrutinizer` and `irpmaster`. To uninstall, use the command `sudo yum erase harctoolbox harctoolbox-doc`.

Unfortunatelly, the packaging contains a few semi-serious problems. These are reported, but stay unfixed. Fedora users are recommended to use the generic binary package.

2.2.3.3 Windows

Download the Window setup file and double click it. Select any installation directory you like; suggested is `C:\Program Files\IrScrutinizer`, unless you can not install with administrator rights. Unless reason to do so, create the start menu folder, and the desktop icon. Administrator rights are needed only it installing in a directory like `Program Files` is desired. IrScrutinizer can now be started from `Start -> IrScrutinizer -> IrScrutinizer`, or from the desktop icon.

This will also install the command line program IrpMaster, which can be called as `irpmaster` from a so-called DOS box.

To uninstall, select the uninstall option from the Start menu. Very pedantic people may like to delete the properties file too, see properties.

2.2.3.4 Mac OS X app

Download the compressed app. Uncompress it by double clicking. Opening it will show an app, and a few documention files. The app can just be dragged to the desktop, to the tray, to "Applications" or any other location the user prefers. IrScrutinizer now behaves like any other app in Mac OS X.

(The previous version was instead distrubuted as a compressed disk image. This should be uncompressed it by double clicking, and mounted by double clicking the resulting file. A mounted disk image "IrScrutinizer" will appear on the desktop. Then proceed as above.)

To unistall, just move the app to the trash.

The command line program IrpMaster is not supported in this mode. (For this, the generic binary distribution has to be used.)

2.2.3.5 Generic Binary

The generic binary version consists of all the java code packed in one executable jar file, together with supporting files, like all the compiled shared libraries for the different operating systems Linux, Windows, and Mac. It can be used on all those systems. (In other environments, the shared libraries may be compiled with moderate effort.)

The generic binary distribution be used whenever using the rpm/setup.exe/app installation is not possible, not up-to-date, or not desired.

To install, unpack in an empty directory of your choice, suggested is `/usr/local/irscrutinizer`. Inspect the wrapper `irscrutinizer.sh`, and make changes if necessary.

It is recommended to make links from a directory in the path to the wrapper script, e.g.

```
ln -s /usr/local/irscrutinizer/irscrutinizer.sh /usr/local/bin/irscrutinizer
ln -s /usr/local/irscrutinizer/irscrutinizer.sh /usr/local/bin/irpmaster
```

If your system supports the RXTX, you should preferably use that. See the comments in the wrapper `irscrutinizer.sh`.

The JNI libraries `libDecodeIR.so`/`DecodeIR.dll`/`libDecodeIR.jnilib` are contained in the distribution and should be found by the program in the installed location.

The desktop file `irscrutinizer.desktop` should, if desired, be installed in `/usr/local/share/applications` alternatively `~/.local/share/applications`.

The program can now be started either as a desktop program, or by typing `irscrutinizer` on the command line. Also the command line program IrpMaster can be started by the command `irpmaster`. It is also possible to start the program by double clicking on the jar file. In case this brings up the archive manager, the desktop needs to be taught to open executable jar files with the "java" application. For this, select a jar file the file browser, select the properties, and select "java" as the application to "open with". (The details might vary.)

To uninstall, just delete the files. Very pedantic people may like to delete the properties file too.

2.2.3.6 Source distribution

Compiling the sources is covered in the Appendix. This allows to install the different components in a way compliant with e.g. the installation by the Gnu Autotools, normally in subdirectories of `/usr/local`.

2.2.4 Concepts

For anyone familiar with the problem domain, this program is believed to be intuitive and easy to use. Almost all user interface elements have tool-help texts. Different panes have their own pop-up help. In what follows, we will not attempt to explain every detail of the user interface, but rather concentrate on the concepts. Furthermore, it is possible that new elements and functionality has been implemented since the documentation was written.

This program does not disturb the user with a number of annoying, often modal, pop ups, but directs errors, warnings, and status outputs to the *console window*, taking up the lower third of the main window. This window is re-sizeable. There is a context menu for the console, accessible by pressing the right mouse button in it.

In the upper row, there are six pull-down menus, named File, Edit, Actions, Options, Tools, and Help. Their usage is believed to be mainly self explanatory, with some the exceptions.

Options to the program are in general found in the Options menu, or its subordinate menus. Some parameters for particular export formats are found in the sub-panes of the "Export" pane. Also the hardware configuring panes contain user parameters.

The main window is composed of seven sub panes denoted by "Scrutinize signal" (for processing single signal), "Scrutinize remote" (for collecting several signals to one

"remote"), "Generate" (generates IR signal from protocol name and parameters), "Import", "Export", "Capturing Hardware", and "Sending Hardware" respectively. These panels will be discussed in Section GUI Elements walk through

2.2.5 Analyzing a single IR Sequence or IR Signal

The pane "Scrutinize Signal" is devoted to the analysis of one single IR sequence.

To capture IR Sequences from a hardware sensor, first set it up and open it, see Section Capturing Hardware. An IR Sequence is captured by pressing the "Capture" button, and sending an IR signal to the selected hardware. Note that the hardware captures an *IR Sequence*, not an IR Signal. It consists of an (sometimes empty) start sequence, an unknown number of repeat sequences, and sometimes an ending sequence.

2.2.6 Adding new export formats

Only some of the many export formats are defined in the Java code of the program, the rest in the file `exportformats.xml`, located in the root of the install directory. By modifying this file, the user can simply add his/her own export formats according to own needs. An exporformat consists of a number of properties, together with a small "program" written in the transformation language XSLT, for transforming a Girr-XML-tree to the desired text format.

The rest of this section documents the format of the file, and is supposed to be read only when needed. Fundamental knowledge of XML and XSLT transformations are assumed.

2.2.6.1 Format of the file exportformats.xml

The file is an XML file read in without validation. The syntax and semantics are beleived to be essentailly self explaining or clear from the examples aleady in there. An export format is packed in an element of type `exportformat`. It contains the following attributes:

name
Text string used for identifying the format.
extension
Text string denoting preferred file extension (not including period) of generated files.
multiSignal
Boolean (value: true and false). Denotes if several signals can be represented in one export, or only one.
simpleSequence
Boolean, (values `true` of `false`). If true, the export really describes an sequence rather than am signal (with intro-, repeat- , and ending-sequences), therefore the user must explictly state a particular number of repetitions for an export.

The element contains a single child element, namely the XSLT transformation, which is an element of type `xsl:stylesheet`, with attributes as in the examples. It should

transform a Girr XML DOM tree in the "fat" format, with `remotes` as root element, into the desired text format. It may be advisable to use the already present formats as guide.

After editing the file, it can be reloaded either by re-starting the program, or by selecting Options -> Export formats database -> Reload.

2.2.7 Properties

Under Windows, the properties are stored in `%LOCALAPPDATA%\IrScrutinizer\IrScrutinizer.properties.xml` using Windows Vista and later (on my Windows 7 system, this is `%HOME%\AppData\Local\IrScrutinizer\IrScrutinizer.properties.xml`), otherwise in `%APPDATA%\IrScrutinizer\IrScrutinizer.properties.xml`. Using other operating systems, it is stored under `$HOME/.config/IrScrutinizer/properties.xml`. It is not deleted by un-install. (If weird problems appear, for example after an update, try deleting this file. For this, there is a command line option `--nuke-properties` that can be used to conveniently delete the present property file, without knowing its exact name.)

2.2.8 GUI Elements walk through

2.2.8.1 The "Scrutinize signal" pane

This panel is devoted to the analysis of a *single* *IR signal* or IR sequence. A (single) signal is either read from hardware using the "Capture" button (requires that the capturing hardware has been set on the "Capturing Hardware" pane), imported from a file (using the context menu in the data window, or through File -> Import -> Import as single sequence), or pasted from the clipboard. Also, some other panes can transfer data to this pane. For text import, the signal can be in either Pronto CCF format, raw format (indicated by a leading "+"-sign), or in the UEI learned format. The signal is printed in the data window, in the preferred text format, which can be selected from the options menu. The text representation may be edited (assuming sufficient knowledge!), after which the edited signal is analyzed and plotted again by pressing the "Scrutinize" button. The signal may be sent to the sending hardware by pressing the "Transmit" button.

Using context menus, the result can be sent to the clipboard or saved to a file.

The plot can be zoomed by pressing the left mouse button and dragging. Printing and exported as graph is presently not implemented.

The menu entry Actions -> Enter test signal (or its accelerator, the F9 key) enters a test signal.

In rare cases, transforming the signal between different formats may introduce some rounding errors causing decoding to fail.

2.2.8.2 The "Scrutinize remote" pane

This panel is devoted to the capturing/import/editing of a collection of IR signals, called "a remote" in the sequel. The panel contains two sub-panels: for parametric signals and for non-parametric, "raw", signals.

A "parametric" signal is determined by its protocol name, and the values of the protocol's parameters. A "raw" signal is determined by its timing pattern, and its modulation frequency. It may have one or many decodes, or none. Nevertheless, by definition, a raw signal is determined by its timing, not the decodes.

In both cases, the sub panes consists of tables with a number of columns. Every signal takes up a row in the table. The content of the individual cells (with the exception of its number and date) can be individually edited, like in a spreadsheet program.

In both tables, the right mouse button opens a context menu containing a number of ways to manipulate the table, its view, or the data contained therein. By enabling the row selector, the rows can be sorted along any of the present columns.

To capture a number of IR signals, first configure the hardware using the capturing hardware pane. Next press the Capture button. The program will now run the capturing in a separate thread, so the user just have to press the buttons of the remote. The signals will be received, interpreted, decoded, and entered on subsequent lines in the selected table (raw or parameterized). The capture thread will continue until the captured button is pressed again. (Note that this is completely different from the capture button on the "Scrutinize signal" panel.) The user may mix captures with other activities, like entering information (name, comments,...) in the table.

The export button exports the content of the currently selected table (raw or parameterized) according to the currently selected export format.

The menu entry Actions -> Enter test signal (or its accelerator, the F9 key) enters a test signal, either as parametric signal, or as a raw signal.

2.2.8.3 The "Generate" pane

In the upper part of this pane, an IR protocol is selected, identified by name, and the parameters D ("device", in almost all protocols), S ("sub-device", not in all protocols), F ("function", also called command number or OBC, present in almost all protocols), as well as T, "toggle" (in general 0 or 1, only in a few protocols). These number can be entered as decimal numbers, or, by prepending "0x", as hexadecimal numbers.

By pressing "Generate", the signal is computed, and the middle window is filled with a textual representation, in the form selected by Options -> Output Text Format.

The Export button initiates an export to a file format selected by the Export pane. The three lower buttons transfer the signal(s) to the scrutinize signal panel, the raw remote table, or the parameterized panel.

Accessing a number of different parameter values

For the export and the transfer to the "scrutinize remote" tables, not only a single parameter value can be selected, but whole sets. The complete syntax and semantics is given in the IrpMaster documentation, we here just mention that e.g. 12:34 means all numbers between 12 and 34 (inclusive), and * denotes all possible values (as defined by the protocol's IRP notation).

2.2.8.4 The Import pane

The import pane allows for selective import of collection of IR commands. Both Internet data bases and file formats are supported. Import can take place from local files or even file hierarchies, from the clipboard, or from Internet URLs.

There are a number of elements common to most of the sub-panes, so these will be described next.

For file/URL based imports, there is a text field, named File or File/URL. For the latter case, an URL (like `http://lirc.sourceforge.net/remotes/yamaha/RX-V995` can be entered, for subsequent import without downloading to a local disc. By pressing the "..."-Button, a file selector allows the selection of a local file. For files and URLs, the "Edit/Browse" button allows to examine the selected file/URL with the operating system's standard command.

If the option "`Open ZIP files`" (accessible from `Options -> Import`) is selected, zip files can be selected, opened, and unzipped "on the fly", without the need for the user to unzip to intermediate files.

When pressing one of the "Load", "Load File/URL", or "Load from clipboard" button, the selected information is downloaded, and presented in the format of an expandable tree. By placing the mouse cursor over a command, additional information, like decode, is presented. A single command can be selected by mouse click, a sequence of adjacent commands by shift-click, a subset of not necessarily adjacent commands be selected by Ctrl-click, as usual from most GUIs. A single selected command can be transferred to the "Scrutinize signal" pane by pressing "Import signal". The "Import all" ("Import selection") button transfers all commands (the selected commands) to the "Scrutinize remote" pane, sub-pane "Parametric remote" (without overwriting already present commands), while the buttons "Import all/raw" and "Import selected/raw" transfer to the sub-pane "Raw remote".

The key "Transmit selected" transmits the (single) selected signal to the selected sending hardware.

Global Caché Database

Global Caché maintains a data base of IR signals, made available free of charge. "Contains over 100,000 Infrared codes for over 2,000 different remotes from over 500 manufacturers". To use from IrScrutinizer, an API Key has be be retrieved. This can be

done from a Facebook, Google, or Yahoo account. After pressing the "APIKey" button, the API key is entered in the pop-up window. It is subsequently saved to the program's properties. To use, select, in order, a manufacturer, a device type, and a setup code, the latter possibly by trial-and-error.

The IRDB Database

IRDB is "one of the largest crowd-sourced, manufacturer-independent databases of infrared remote control codes on the web, and aspiring to become the most comprehensive and most accurate one."

To use, select, in order, a manufacturer, a device type, and a protocol/parameter combination, the latter possibly by trial-and-error.

Pressing the "Load all" button transfers all present protocol/parameters combinations to the tree.

Girr (the native format of IrScrutinizer)

The Girr format is the native format of IrScrutinizer.

LIRC

The LIRC import feature is based upon Jirc, which is basically LIRC translated into Java. The LIRC importer can even import a file system hierarchy by selecting the top directory as File/URL. (Importing the entire lirc.org database with over 100000 commands takes around 1 minute and 1GB of memory.)

CML

The CML format is the format used by the RTI Integration Designer software. Many CML files are available in Internet, in particular on RemoteCentral. Particularly noteworthy is the "megalist" by Glackowitz. IrScrutinizer can import these files to its import tree, making every remote a nodes in the tree. Note that there is no need to unzip such a file; IrScrutinizer will unzip it on the fly.

Command Fusion

The native format that Command Fusion uses, file extension `.cfir`, can be imported here.

RemoteMaster

The JP1 community has a large data base of parametric IR commands. IrScrutinizer has support for importing RMDU files for RemoteMaster. Unfortunately, the signals are stored as parameters for so-called executors, with sometimes different parameterization ("hex", "efc") than the IRP protocols. Translating these files to one of the known protocol/parameter format is nothing but straightforward. It uses protocol information

contained in protocols.ini. IrScrutinizer reads this file, and can do some computations, for example on NEC1 protocols, but not on all protocols.

For signals without recognized protocol name, importing as raw signals, or to "Scrutinize signal", is not possible. However, they can always be imported as parametric signals, possibly for manual edit.

Pronto Classic (CCF format)

Many Pronto CCF files are available in Internet, in particular by Remote Central. IrScrutinizer can read in these files to its import tree, even preserving the Pronto "devices" as nodes in the tree.

Unfortunately, the Tonto library used produces the warning message `Error loading JComm JNI library`. This message should be ignored.

Pronto Prof. (XCF format)

Pronto Professional XCF files are found for example at Remote Central. IrScrutinizer can read in these files to its import tree, even preserving the Pronto "devices" as nodes in the tree.

ICT IrScope format

The ICT format, introduced by Kevin Timmerman's IrScope, contains the timing pattern, the modulation frequency, and optionally a name ("note") of one or many IR signals.

Text format

In the Internet, there are a lot of information in table-like formats, like Excel, describing the IR commands of certain devices. IrScrutinizer has some possibilities of importing these — after exporting them to a text format, like tab separated values (tsv) or comma separated values.

Raw

The sub-pane allows for the parsing of text files separated by a certain characters, like commas, semicolons, or tabs. The separating characters is selected in the "Field separator" combo box. The column to be used as name is entered in the "Name col." combo box, while the data to be interpreted either as raw data or CCF format, is entered in the "Raw signal col.". If the "... and subsequent columns" is selected, all subsequent columns are added to the data.

Raw, line-based

This pane tries to interpret a line-based file as a number of named IR commands, using heuristics.

Parameterized

The sub-pane allows for the parsing of text files separated by a certain characters, like commas, semicolons, or tabs. The separating characters is selected in the "Field separator" combo box. The column to be used as name is entered in the "Name col." combo box, while protocol name and the parameters D, S, and F are entered in their respective combo boxes. They are parsed in the number base selected.

Wave

This pane imports and analyzes wave files, considered to represent IR signals. The outcome of the analysis (sample frequency, sample size, the number of channels, and in the case of two channels, the number of sample times the left and right channels are in phase or anti-phase) is printed to the console.

2.2.8.5 The Export pane

Using the export pane, export files can be generated, allowing other programs to use the computed results. Single signals (from the "Scrutinize signal" pane), collections of signals (from the "Scrutinize remote" pane), or generated signals can be exported. Exports can be generated in a number of different formats. Some (Girr (=XML) and text) can contain both the Pronto format and the "raw" format (timings in microseconds, positive for pulses, negative for gaps), as well as other formats. These formats, together with Wave, LIRC, and Pronto Classic, are built-in in the program. However, it is possible to define new export formats by extending a configuration file, see Adding new export formats. The formats are, at the time of this writing:

Girr
The program's native format, based on XML. Very flexible and extensible. Can contain information like the raw format, CCF format, UEI learned format, and the Global Caché sendir format. Documentation.

Text
The text format is essentially the Girr format stripped of the XML markup information.

Wave
IR sequences encoded as wave audio files.

LIRC
The LIRC-exports are in lirc.conf-format using the raw LIRC format. They can be concatenated together and used as the LIRC server data base. Can also be used with WinLirc.

Pronto Classic
This format generates a CCF configuration file to be downloaded in a Pronto, or opened by a ProntoEdit program.

IrTrans
This export format generates .rem files for the IrTrans system, using its CCF format.

Lintronic

Simple text protocol for describing a single IrSequence.

Spreadsheet

Simple tab separated value export format for importing in a spreadsheet program.

RM Functions

Variant of the Spreadsheet format, this format is intended to be pasted directly into the "Functions" table of RemoteMaster.

C

Intended mostly as an example of generating C code, cf. this article.

TV-B-Gone

Variant of the C format, this format generates C code for the TV-B-Gone.

Export file names are either user selected from a file selector, or, if "Automatic file names" has been selected, automatically generated.

The export is performed by pressing the "Export" button. The "..."-marked button allows for manually selecting the export directory. It is recommended to create a new, empty directory for the exports. The just-created export file can be immediately inspected by pressing the "Open last file"-button, which will open it in the "standard way" of the used operating system. (Also available on the actions menu.) The "Open" button similarly opens the operating systems standard directory browser (Windows Explorer, Konquistador, Nautilus,...) on the export directory.

Some export formats (presently Wave and Lintronic) export an IR sequence rather than an IR signal (consisting of an intro sequence, an repetition sequence (to be included 0 or more times), and an (most often empty) ending sequence). Using these formats, the number of repetition sequences to include can be selected.

Some export formats have some more parameters, configured in sub panes. These will be discussed next.

The Girr sub-pane

A style sheet can be selected to be linked in into the exported Girr file. The type of style file (presently xslt and css) can also be selected.

"Fat form raw" can be selected; this means that the raw signals are not given as a text string of alternating positive and negative numbers, but the individual flashes and gaps are enclosed into own XML elements. This can be advantageous if generating XML mainly for the purpose of transforming to other formats.

The Wave sub-pane

Parameters for the generated Wave export (except for the number of repeats) can be selected here.

The sendir sub-pane

The Global Caché sendir format requires a module number and a connector number. Also, there is a compressed format, that can be enabled by selecting the compressed check-box.

The Pronto Classic sub-pane

A Pronto Classic export consists of a CCF file with the exported signals associated to dummy buttons. The Pronto (Classic) model for which the export is designed is entered in the combo box. Screen size of the Pronto is normally inferred from the model, but can be changed here. The button size of the generated buttons is also entered here.

2.2.8.6 The "Capturing HW" pane

The sub-panes of this pane allow for the configuration of capturing hardware. Selecting a sub-pane also selects the associated hardware, if possible. The hardware can also be selected from the tool bar, Options -> Capturing hardware.

Unfortunately, by e.g. selecting non-existing hardware or such, there is a possibility to "hang" the program.

After configuring and opening the capturing hardware, the "Test" button can be used for testing the configuration without switching pane.

Selected ports are stored in the properties, thereby remembered between sessions. So, for future sessions, only opening the preferred device is necessary.

IrWidget

Plug the IrWidget it into the computer. Check that the operating system has assigned a port to it, and note which one it is. On Windows: open the device manager, and check that there is one "USB Serial Port" under Ports. Note the port number (e.g. `COM8`). On a Linux system, it likely shows up as a device like `/dev/ttyUSB0`. If the port does not show up, a suitable driver needs to be installed. If the correct port is already visible in the combo box, just press "Open". Otherwise, press "Refresh", which makes the program determine the currently available serial ports. Select the correct one. Press "Open". which should now remain "pressed". The port can be closed again by another press, but there is not much reason to do so, unless another capturing hardware should be used, or the IrWidget should be used from another program.

Global Caché capture

IrScrutinizer automatically detects alive Global Caché units in the local area network, using the AMX Beacon. However, this may take up to 60 seconds, and is not implemented in very old firmware. Using the "Add" button, the IP address/name of older units can be entered manually.

For this to work, port 9131/udp must be open in a used firewall

The "Browse" button points the browser to the selected unit.

The reported type and firmware version serves to verify that the communication is working.

LIRC Mode2

`mode2` is a program from the LIRC distribution, that prints timing information in a simple text format to its standard-out. In theory, any program that prints information in that format can be used. The command line for the program with possible parameters is to be entered as command. With the Start button, a sub-process is started, running the given command line. The "Stop" button stops the sub-process — although sometimes this may not stop the started program.

Has been tested only on Linux, should however work on all systems.

Arduino

To use the Arduino with a non-demodulating receiver for IR capture, the sketch `GirsLite` needs to be loaded.

After connecting the Arduino to a real or virtual serial port, select the port in the combo box, pressing "Refresh" if necessary. "Open" the port thereafter.

At least on Linux, the ACMX port may be finicky. Sometimes disconnecting and reconnecting the device may help.

IrToy

After connecting the IrToy to a USB port, select the the actual (virtual) serial port in the combo box, pressing "Refresh" if necessary. "Open" the port thereafter.

The ending timeout is 1.4 seconds, and cannot be changed. This is not optimal for most IR signal capture use cases.

At least on Linux, the ACMX port may be finicky. Sometimes disconnecting and reconnecting the device may help.

Command Fusion Learner

After connecting the Command Fusion learner to a USB port, select the the actual (virtual) serial port in the combo box, pressing "Refresh" if necessary. "Open" the port thereafter.

The "Sending HW" pane

The sub-panes of this pane allows for the selection and configuration of the deployed IR sending hardware.

The "Global Caché" pane.

IrScrutinizer automatically detects alive Global Caché units in the local area network, using the AMX Beacon. However, this may take up to 60 seconds, and is not implemented in very old firmware. Using the "Add" button, the IP address/name of older units can be entered manually.

The "Browse" button points the browser to the selected unit.

The reported type and firmware version serves to verify that the communication is working.

"Stop IR"-Button allows the interruption of ongoing transmission, possibly initiated from another source.

The user can select one of the thus available Global Caché units, together with IR-module and IR-port (see the Global Caché API specification for the exact meaning of these terms).

The "LIRC" pane

To be fully usable for IrScrutinizer, the LIRC server has to be extended to be able to cope with CCF signal not residing in the local data base, but sent from a client like IrScrutinizer, thus mimicking the function of e.g. a Global Caché. The needed modification ("patch") is in detail described here. However, even without this patch, the configuration page can be used to send the predefined commands (i.e. residing it its data base `lirc.conf`). It can be considered as a GUI version of the `irsend` command.

The LIRC server needs to be started in network listening mode with the `-l` or `--listen` option. Default TCP port is 8765.

After entering IP-Address or name, and port (stay with 8765 unless a reason to do otherwise), press the "Read" button. This will query the LIRC server for its version (to replace the grayed out "<unknown>" of the virgin IrScrutinizer), and its known remotes and their commands. Thus, the "Remote" and "Command" combo boxes should now be selectable. After selecting a remote and one of its command, it can be sent to the LIRC server by pressing the "Send" button. If (and only if) the LIRC server has the above described patch applied, transmitting signals to "LIRC" now works.

Due to LIRC's peculiar form of API stop command, the "Stop IR" command presently does not work. See this thread in the LIRC mailing list for a background.

The "IRTrans" pane

The configuration of IRTrans is similar to LIRC, so it will be described more briefly.

Enter IP name or -address and select an IR port (default "intern"). If the Ethernet IRTrans contains an "IR Database" (which is a slightly misleading term for an internal flash memory, that can be filled by the user), its commands can be sent from this pane. By pressing the "Read" button, the known remotes and commands are loaded, and the version of the IRTrans displayed. The selected command can now be sent by the "Send"

button. (However, this functionality is otherwise not used by IrScrutinizer.) Selecting "IRTrans" on the "Analyze" and "War dialer" pane should now work. The IRTrans module is then accessed using the UDP text mode.

The "IrToy" Pane

Using this pane, the IrToy (version 2) can be used to transmit IR signals. To my knowledge, only the firmware version 2.2.2 works.

The "CommandFusion" Pane

Using this pane, the CommandFusion learner can be used to transmit IR signals.

The "Arduino" Pane

Using this pane, an Arduino equipped with a suitable IR Led can be used to transmit IR signals. First however, the sketch `GirsLite` from the arduino subdirectory has to be downloaded to the Arduino.

The "Audio" Pane

As additional hardware device, IrScrutinizer can generate wave files, that can be used to control IR-LEDs. This technique has been described many times in the Internet the last few years, see for example this page within the LIRC project. The hardware consists of a pair of anti-parallel IR-LEDs, preferably in series with a resistor. Theoretically, this corresponds to a full wave rectification of a sine wave. Taking advantage of the fact that the LEDs are conducting only for a the time when the forward voltage exceeds a certain threshold, it is easy to see that this will generate an on/off signal with the double frequency of the original sine wave. (See the first picture in the LIRC article for a picture.) Thus, a IR carrier of 38kHz (which is fairly typical) can be generated through a 19kHz audio signal, which is (as opposed to 38kHz) within the realm of medium quality sound equipment, for example using mobile devices.

IrScrutinizer can generate these audio signals as wave files, which can be exported from the export pane, or sent to the local computers sound card. There are some settings available: Sample frequency (44100, 48000, 96000, 192000Hz), sample size (8 or 16 bits) can be selected. Also "stereo" files can be generated by selecting the number of channels to be 2. The use of this feature is somewhat limited: it just generates another channel in opposite phase to the first one, for hooking up the IR LEDs to the difference signal between the left and the right channel. This will buy you double amplitude (6 dB) at the cost of doubling the file sizes. If the possibility exists, it is better to turn up the volume instead.

Most of "our" IR sequences ends with a period of silence almost for the half of the total duration. By selecting the "Omit trailing gap"-option, this trailing gap is left out of the generated data – it is just silence anyhow. This is probably a good choice (almost) always.

Note that when listening to music, higher sample rates, wider sample sizes, and more channels sound better (in general). However, generating "audio" for IR-LEDs is a completely different use case. The recommended settings are: 48000kHz, 8bit, 1 channel, omit trailing gap.

2.2.9 Command line arguments

Normal usage is just to double click on the jar-file, or possibly on some wrapper invoking that jar file. However, there are some command line arguments that can be useful either if invoking from the command line, or in writing wrappers, or when configuring custom commands in Windows.

The options `--version` and `--help` work as they are expected to work in the GNU coding standards for command line interfaces. Use the `--help`-command to see the complete list of command line parameters. The `-v`/`--verbose` option set the verbose flag, causing commands like sending to IR hardware printing some messages in the console.

The option `--nuke-properties` makes the program delete the property file, and exit immediately.

For automating tasks, or for integrating in build processes or Makefiles or the like, it is probably a better idea to use IrpMaster instead, which has a reasonably complete command line interface.

The program delivers well defined and sensible exit codes.

2.2.10 Questions and Answers

2.2.10.1 Does IrScrutinizer completely replaces IrMaster?

Almost. Using MakeHex as renderer (or more correctly, its Java version) instead of IrpMaster is not implemented. (The practical usage of this feature is probably *very* limited, and IrMaster is still available, should it ever be needed.) The "war dialer" is also not implemented, but see next question. For the wave export, some rarely used options (the possibility to select big-endian format (for 16-bit samples), the possibility *not* to half the carrier frequency, and the possibility to select sine (instead of square) for modulation) have been removed. Finally, there is some stuff that simply works differently, like the export function.

2.2.10.2 How do I emulate the war dialer in IrScrutinizer?

Use "Scrutinize remote" -> Parametric Remote. Fill in the table with signals to be tested, either using the pop-up button (right mouse in the table) Advanced -> Add missing F's, or from the Generate pane, using suitable parameter intervals (see TODO), and transfer them using the "To parametric remote" button. Then test the candidate signals one at a time by transmit-ting them, using suitable sending hardware. The comment field, or the "verified" check-box, can be used for note taking.

A "war dialer" like in IrMaster may be implemented in a later version.

2.2.10.3 Can I use this program for conveniently controlling my favorite IR controlled device from the sofa?

No, the program is not meant for that. While you definitely can assemble a "remote" on the "scrutinize remote" panel, and transmit the different commands with mouse commands (appropriate hardware assumed), the program is intended for developing codes for other deployment solutions.

2.2.10.4 The pane interface sucks.

Yes. There are several use cases when the user would like to see several "panes" simultaneously. Also, it should be possible to have several windows of the same sort (like the "scrutinize signal") simultaneously. Replacing the top level panes with something "Eclipse-like" (sub-windows that can be rearranged, resized, moved, iconized) is on my wish list.

2.2.10.5 Why can't I have a device opened both for capturing and sending?

This is unfortunately a fundamental design flaw, one that sits deep in the design. A quick fix is therefore not likely. Sorry. (However, it is possible to use two different instances of the hardware, e.g. two IrToys; one for capturing and one for transmission.)

2.2.10.6 What about the "fishy" icon?

It is a Babel fish, as found in The Hitchhiker's Guide to the Galaxy, having the property, that "... if you stick one in your ear, you can instantly understand anything said to you in any form of language". This symbolizes the program's ability to "understand" a large number of different IR formats.

2.2.10.7 Why supporting the old Global Caché data base but not the new ("Control Tower")?

Sad, really, it is a really nice data base. The reason is that API retrieval of codes is not allowed for the free accounts. The data base can still be used with IrScrutinizer however: Log in with the web interface, and have the codes mailed in CSV format (press "Send Code Set"). The mail received can be imported by the import text pane, raw subpane, Name col. = 1, Raw signal col = 2 (or 3), Field separator: comma.

2.2.10.8 I did something funny, and now the program does not startup, with no visible error messages.

Try deleting the properties file. (Note the command line option `--nuke-properties` which will do exactly that, without having to manually find the file or its name.) If that does not help, try starting the program from the command line, which may leave hopefully understandable error message on the console.

2.2.10.9 (Windows) When I double click on the IrScrutinizer symbol, instead of the program starting, WinRar (or some other program) comes up.

The program that comes up has "stolen" the file association of files with the extension `.jar`. Restore it. (Allegedly, WinRar can gracefully "unsteal" file associations.)

2.2.10.10 (Linux) I get error messages that lock files cannot be created, and then the Arduino and IrToy hardware do not work.

When starting IrScrutinizer, or by pressing the "Refresh" button, error messages occur like

`check_group_uucp(): error testing lock file creation Error details:Permission deniedcheck_lock_status: No permission to create lock file.`

(and a number of them...) The problem is that the library `rxtx` (like some other program accessing a serial interface) wants to create a lock file in a particular directory. This is/was traditionally `/var/lock`, which is often a symbolic link to `/var/run/lock`. This directory is normally writable by members of the group `lock`. So your user-account should probably be a member of that group. (How to perform this is different in different Linux distributions.) The rxtx library delivered with IrScrutinizer expects the lock directory to be `/var/lock`. However, recently some Linux-distributions (e.g. Fedora 20), instead are using `/var/lock/lockdev` as its lock directory (while `/var/lock` still is a link to `/var/run/lock`. To solve this, I recommend, if possible, installing rxtx provided by the Linux distribution used, i.e. not using the one with IrScrutinizer. For example, on Fedora, the command is

```
sudo yum install rxtx
```

which installs the library in `/usr/lib/rxtx` or `/usr/lib64/rxtx`, depending on the operating system. (Other distributions uses other commands, for example `apt-get` on Debian-like systems.) Finally, the correct installation directory of the library (`librxtxSerial-2.2pre1.so`) has to be given to the JVM running IrScrutinizer. For this, see the wrapper `irscrutinizer.sh` and the comments therein, and make the necessary adaptations.

2.2.10.11 What is on the splash screen?

From left to right, a Global Caché iTach Flex, an IrToy, and a low-cost clone of an Arduino Nano, the latter equipped with a non-demodulating IR detector (TSMP4138) for capturing and an IR diode (SFH415) for sending. These are all hardware which work well with IrScrutinizer, both for sending and capturing.

2.2.10.12 Do you solicit or accept donations?

No.

2.2.11 Appendix. Building from sources

"IrScrutinizer" is one subproject (corresponding to a Java package) within "harctoolbox.org". It depends on several other subprojects within harctoolbox. The project "harctoolboxbundle" consists of these subproject bundled together, with some dependent third-party components added.

The released versions are found on the download page. The development sources are maintained on my GitHub repository. Forking and pull requests are welcome!

2.2.11.1 Dependencies

As any program of substantial size, IrScrutinizer uses a number of third-party components. All of these are also free software, carrying compatible licenses. The dependent packages need to be installed also in maven in order for the build to work. With the dependencies available, the script `tools/install-deps.sh` can be used to install them in the local maven repository before building.

Presently, Fedora RPMs are available for all dependencies. Installing the `harctoolbox` package will also install all dependencies, in which case the information in this chapter is not needed.

The Crystal icons

A subset of the Crystal icons are included, and will be included in a built jar. Most Linux distributions contain these too, so a Linux packaging may like to use the system icons instead.

RXTX

The serial communication packate RXTX is also included in the source package. This builds a shared library and a jar file. If there is a system supported RXTX (librxtxSerial really), it should be preferred. The distribution constains pre-compiled binaries for Linux, Windows, and Mac OS X, both in 32- and 64-bit versions. To compile the C sources, see the sub-directory `rxtx-pre2h` and the instructions therein.

Note that the system supplied RXTX jar on many system (e.g. Fedora 21) has some issues (version number incompatible with the shared library, does not recognize the `/dev/ttyACM*`-ports required by IrToy and many Arduinos, unflexible library loading), so using our RXTX jar together with the system supplied shared library can be sensible.

DecodeIR

If the system supports DecodeIR, use the system version. On recent Fedora, this can be installed with the command `sudo yum install DecodeIR`. This will install both the shared library `libDecodeIR` as well as the jar file `DecodeIrCaller.jar`. To download and compile the sources, see (or execute) the script `tools/build-decodeir.sh`.

ExchangeIR

It the system supports ExchangeIR (java), use the system version. (On recent Fedora, use `sudo yum install ExchangeIR`.) Otherwise, it can be downloaded and installed by the script `tools/build-exchangeir.sh`.

minimal-json

It the system supports minimal-json, use the system version. (On recent Fedora, use `sudo yum install minimal-json`.) Otherwise, it can be downloaded and installed by the script `tools/build-minimal-json.sh`.

jcommander

It the system supports jcommander, use the system version. (On recent Fedora, use `sudo yum install beust-jcommander`.) Otherwise, it can be downloaded and installed by the script `tools/build-jcommander.sh`.

Tonto

It the system support Tonto, use the system version. (On recent Fedora, use `sudo yum install tonto`.) Otherwise, it can be downloaded and installed by the script `tools/build-tonto.sh`.

Note that the shared library `libjnijcomm`, which is required by the program Tonto for communicating with a Pronto remote through a serial interface, is not required for use with IrScrutinizer, and can therefore be left out.

2.2.11.2 Building

As of version 1.1.2, the Maven "software project management and comprehension tool" is used as building system. Modern IDEs like Netbeans and Eclips integrate Maven, so build etc can be initiated from the IDE. Of course, the shell command `mvn install` can also be used. It creates some artifacts which can be used to run IrScrutinizer in the `IrScrutinizer/target` directory.

It also creates a `package/dist` directory containing jars (without dependencies), docs, and configurations files. This is intened to support packaging.

To prepare the Windows version, some shell tools are needed. These are:

- The `unix2dos` and `dos2unix` utilities, typically in the `dos2unix` package.
- The `icotool` utility, typically in the `icoutils` package

2.2.11.3 Windows setup.exe creation

For building the Windows setup.exe, the Inno Installer version 5 is needed. To build the Windows setup.exe file, preferrably the work area should be mounted on a Windows computer. Then, on the Windows computer, open the generated file `IrScrutinizer/`

target/IrScrutinizer_inno.iss with the Inno installer, and start the compile. This will generate the desired file IrScutinizer-*version*.exe.

Alternatively, the "compatibility layer capable of running Windows applications" software application Wine (included in most Linux distributions) can run the ISCC compiler of Inno. The Maven file IrScrutinizer/pom.xml contains an experimental invocation along these lines.

2.2.11.4 Mac OS X app creation

The Maven build creates a file IrScrutinizer-*version*-app.zip. This file can be directly distributed to the users, to be installed according to these instructions.

The icon file IrScrutinizer.icns was produced from the Crystal-Clear icon babelfish.png in 128x128 resolution, using the procedure described here.

2.2.11.5 Test invocation

For testing purposes, the programs can be invoked from their different target directories. IrScrutinizer can be invoked as

```
$ cd IrScrutinizer
$ java -jar target/IrScrutinizer-jar-with-dependencies.jar
```

and IrpMaster as

```
$ cd IrpMaster
$ java -jar target/IrpMaster-jar-with-dependencies.jar [arguments...]
```

IrScrutinizer can also be started by double clicking the mentioned jar file, provided that the desktop has been configured to start executable jar with "java".

2.2.11.6 Installation

For reasons unclear to me, Maven does not support something like make install for installing a recently build program on the local host. Instead, the supplied script tools/install-irscrutinizer.sh installs the program to normal Linux/autotools locations. (Actually, invoking Maven with the "deploy" target will invoke this script too.) Alternativelly, the just created generic-binary package (IrScrutinizer/target/IrScrutinizer-bin.zip) can be installed using these instuctions.

2.2.12 References

1. IrpMaster. Also a GPL3-project by myself. Much harder to read than the present document :-). See also this discussion thread in the JP1 forum.
2. IpMaster. Also a GPL3-project by myself. The precessor of the this program. See also this discussion thread in the JP1 forum.
3. The Harctoolbox project, also a GPL3-project by myself.

4. DecodeIR. This shared library tries to identify protocol name and parameters of an IR signal in raw form. Thus, it is in a sense, it implements the "inverse mapping" of IrpMaster.
5. GlobalCaché, a manufacturer of Ethernet connected IR hardware. Note that I have only tried with the GC-100 series, but the IR sending models of theiTach family are believed to work too. (Feel free to send me one :-).)
6. IRTrans, another manufacturer of Ethernet connected IR-hardware. The "IRTrans Ethernet" module, preferably with "IRDB Option" (internal flash memory), is directly supported by the current software.
7. LIRC, Linux InfraRed Control This project contain drivers for almost everything IR-related. The present project is able to use a modified LIRC-server for transmitting IR signals.

2.3 IrpMaster: a program and API for the generation of IR signals from IRP notation

> **Note:**
> It may not be necessary to read this document. If your are looking for a user friendly GUI program for generating IR signals etc, please try the program IrScrutinizer (or its predecessor IrMaster), and get back here if (and only if) you want to know the detail on IR signal generation.

2.3.1 Revision history

Date	Description
2011-08-15	Initial version.
2012-04-24	Converted to the document format of Apache Forrest. The program documentation is now generated from that file. Many minor fixes and updates.
2012-06-03	Minor updates for upcoming release 0.2.0.
2012-08-19	Minor updates for upcoming release 0.2.1.
2012-11-18	Minor updates for upcoming release 0.2.2.
2014-01-27	Minor updates for upcoming release 1.0.0.
2014-05-30	Minor updates for upcoming release 1.0.1.

2.3.2 Revision notes

Release notes for the current version

2.3.3 Introduction

The "IRP notation" is a domain specific language for describing IR protocols, i.e. ways of mapping a number of parameters to infrared signals. It is a very powerful, slightly cryptic, way of describing IR protocols. In early 2010, Graham Dixon (mathdon in the JP1-Forum) wrote a specification. Up until this program was released, there has not been a usable implementation of the IRP-notation in the sense of a program that takes an IRP Protocol together with parameter values, and produces an IR signal. (The MakeHex program operates on a previous, more restricted version of the IRP notation. The MakeLearned program has severe restrictions, most importantly, its sources are not available.) The present work is a Java program/library that is hoped to fill that gap. It is written in Java 1.6, may or may not run with Java 1.5, but definitely not with earlier Java versions. It, optionally, calls the shared library DecodeIR on Windows, Linux, or Macintosh, but has no other "impurities" in the sense of Java. It can be used as a command line program, or it can be used through its API. For parsing the IRP-Notation, the tool ANTLR is used, generating the parser automatically from the grammar.

This project does not contain a graphical user interface (GUI). See Main principles for a background. Instead, the accompanying program IrScrutinizer (and its predecessor IrMaster) provides a GUI for the present program, among many other things.

For understanding this document, and the program, a basic understanding of IR protocol is assumed. However, the program can be successfully used just by understanding that an "IRP protocol" is a "program" in a particular "domain specific language" for turning a number of parameters into an IR signal, and the present program is a compiler/interpreter of that language. Some parts of this document requires more IRP knowledge, however.

2.3.3.1 Spelling, pronunciation

The mandatory section... :-; Preferred spelling is "IrpMaster", with "I" and "M" capitalized (just as the Java class). Pronounce it any way you like.

2.3.3.2 Synergies within other projects

I hope that this program/library should be useful to other projects involved in IR signals. It makes the vast knowledge of the JP1 project available to other programs. It can be used off-line, to manually or automatically produce e.g. configuration files containing rendered IR signal in some popular format, like the Pronto format. More exciting is to implement a real time "IR engine", that can generate and transmit IR signals in any of the known formats.

2.3.3.3 Copyright and License

The program, as well as this document, is copyright by myself. Of course, it is based upon the IRP documentation, but is to be considered original work. The "database file" IrpProtocols.ini is derived from DecodeIR.html, thus I do not claim copyright.

The program uses, or interfaces with (the different is slightly blurred), other projects. ExchangeIR was written by Graham Dixon and published under GPL3 license. Its Analyze-function has been translated to Java by myself, and is used in by the present program. DecodeIR was originally written by John S. Fine, with later contributions from others. It is free software with undetermined license. IrpMaster depends on the runtime functions of ANTLR3, which is free software with BSD type license.

The program and its documentation are licensed under the GNU General Public License version 3, making everyone free to use, study, improve, etc., under certain conditions.

2.3.4 Main principles

2.3.4.1 Design principles

It is my opinion that it is better to get the functionality and the API right, before you do a graphical user interface (GUI). It is much easier and logical to put a GUI on top of a sane API, then to try to extract API functionality from a program that was never designed sanely but built around the GUI. (Look at WinZip for a good example of the latter. Or almost any Windows program, commercial or freeware...)

I have tried to follow the IRP document as closely as possible, in particular with respect to the grammar and the syntax. However, the Execution model of Chapter 14, turned out not to be usable.

Performance consideration were given minimal priorities. As it stands, rendering a single IR signal typically takes less than 1 ms, so this seems justified. Some debugging statements are covered by functionally superfluous if-statements in order not to have to evaluate arguments (`to String()` etc) not needed anyhow.

Everything that is a "time" is internally represented as a double precision number. Most output formats however, are in some integer format. I here use the principle of sound numerics, do all computations with "high precision" (i.e. double precision) as long as ever possible, then transform to the lower precision form (i.e. integer formats) only in the final step.

All "integer quantities" like expressions, are done in Java long format, 64 bits long, amounting to 63 bits plus sign. Already the `metanec`-example would not work with Java int's. The performance penalty over using `int` (32 bits) is believed to be neglectable.

Differently put, all parameters are limited to Java's long, and can thus be no larger than 2^63-1 = 9223372036854775807. A real-life protocol where this limit is exceeded is not known to me.

Versions prior to 0.2.1 also limited the length of bitfields, also the concatenation of bitfields, to 64. In version 0.2.1 this restriction has been removed, and arbitrary length (concatenation of) bitfields are allowed, as long as the parameters are less than 2^63-1. (Example: The concatenation of bitfields `A:50,B:50` produces a concatenated bitfield of length 100, which is accepted by IrpMaster 0.2.1, but rejected by prior versions.) (Thanx to 3FG for pointing this out to me.)

I do not have the slightest interest in internationalization of the project in its present form — it does not contain a user friendly interface anyhow.

2.3.4.2 Repetitions

Possibly the major difficulty in turning the IRP Specification into programming code was how to make sense of the repetition concept. Most treatises on IR signals (for example the Pronto format) considers an IR signal as an introduction sequence (corresponding to pressing a button on a remote control once), followed by a repeating signal, corresponding to holding down a repeating button. Any, but not both of these may be empty. In a few relatively rare cases, there is also an ending sequence, send after a repeating button has been released. Probably 99% of all IR signals fit into the intro/ repetition scheme, allowing ending sequence in addition should leave very few practically used IR signals left. In "abstract" IRP notation, these are of the form A,(B)+,C with A, B, and C being "bare irstreams".

In contrast, the IRP notation in this concept reminds they syntax and semantics of regular expressions: There may be any numbers, and they can even be hierarchical. There certainly does not appear to be a consensus on how this very,... general ... notation should be practically thought of as a generator of IR signals. The following, "finite-automaton interpretation" may make sense: An IRP with several repetitions, say, A(B)+C(D)+E, can be thought of as a remote control reacting on single and double presses. Pressing the key and holding it down produces first A, then B's as long as the button is pressed. An action such as shortly releasing the key and immediately pressing it again then sends one C, and keeps sending D's as long as button is kept pressed. When released, E is sent. Similarly, hierarchical repetitions (repetitions containing other repetitions) may be interpreted with some secondary "key" being pressed and/or released while a "primary button" is being held down — possibly like a shift/meta/control modifier key on a keyboard or a sustain/ wah-wah-pedal on a musical instrument?

The present program does not implement hierarchical repetitions. However, an unlimited number of non-hierarchical repetitions are allowed, although not in the form if the class IrSignal — it is restricted to having three parts (intro, repeat, ending). Also, the repetition pattern (...)* is rejected, because it does not make sense as an IR signal.

The command line interpreter contains an "interactive mode", entered by the argument `--interactive`. This way the intrinsic finite state machine (see above) inherent in an IRP with repetitions can be interactively traversed, probably in the context of debugging.

2.3.5 Command line usage

2.3.5.1 Installing binaries

There is no separate binary distributions of IrpMaster. The user who do not want to compile the sources should therefore install the binary distribution of IrScrutinizer, which contains everything needed to run IrpMaster from the command line. Installing that

package, either the Windows installer or the ZIP file, will install a wrapper, which is the preferred way to invoke IrpMaster.

2.3.5.2 Usage of the program from the command line

I will next describe how to invoke the program from the command line. Elementary knowledge of command line usage is assumed.

There is a lot of functionality crammed in the command line interface. The usage message of the program gives an extremely brief summary:

```
Usage: one of
        IrpMaster --help
        IrpMaster [--decodeir] [--analyze] [-c|--config <configfilename>] --version
        IrpMaster [OPTIONS] -n|--name <protocolname> [?]
        IrpMaster [OPTIONS] --dump <dumpfilename> [-n|--name <protocolname>]
        IrpMaster [OPTIONS] [--ccf] <CCF-SIGNAL>|<RAW-SEQUENCE>
        IrpMaster [OPTIONS] [--ccf] "<INTRO-SEQUENCE>" ["<REPEAT-SEQUENCE>" ["<ENDING-
SQUENCE>"]]
        IrpMaster [OPTIONS] [-n|--name] <protocolname> [PARAMETERASSIGNMENT]
        IrpMaster [OPTIONS] [-i|--irp] <IRP-Protocol> [PARAMETERASSIGNMENT]

where OPTIONS=--stringtree <filename>,--dot <dotfilename>,--xmlprotocol
 <xmlprotocolfilename>,
-c|--config <configfile>,-d|--debug <debugcode>|?,-s|--seed <seed>,-q|--quiet,
-P|--pass <intro|repeat|ending|all>,--interactive,--decodeir,--analyze,--lirc
 <lircfilename>,
-o|--outfile <outputfilename>, -x|--xml, -I|--ict, -r|--raw, -p|--pronto, -u|--uei,
--disregard-repeat-mins, -#|--repetitions <number_repetitions>.

Any filename can be given as `-', meaning stdin or stdout.
PARAMETERASSIGNMENT is one or more expressions like `name=value' (without spaces!).
One value without name defaults to `F`, two values defaults to `D` and `F`,
three values defaults to `D`, `S`, and `F`, four values to `D`, `S`, `F', and `T`, in
 the order given.

All integer values are nonnegative and can be given in base 10, 16 (prefix `0x'),
8 (leading 0), or 2 (prefix `0b' or `%'). They must be less or equal to 2^63-1 =
 9223372036854775807.

All parameter assignment, both with explicit name and without, can be given as
 intervals,
like `0..255' or '0:255', causing the program to generate all signals within the
 interval.
Also * can be used for parameter intervals, in which case min and max are taken from
the parameterspecs in the (extended) IRP notation.
```

Note that if using the wrapper previously described, it has already added the option `--config` *`standard_confile`* to the command line.

We will next explain this very brief description somewhat more verbosely:

- The first version simply produces the help message, as per above.
- The second version will will print the versions of the program, and optionally, the version of the configuration file and the DecodeIR dynamic library. The `--version`

argument should normally be given last, since it is executed immediately when the command line is parsed.

- The third version prints the IRP string of the protocol with the given name to the terminal.
- In the forth version, a CCF string (or, alternatively, a string in raw format (with leading "+"), or in UEI learned format) is read in, and, depending on the to the other options invoked, translated to another format, or sent to DecodeIR and/or AnalyzeIR.
- The fifth version differs form the forth version in that an intro-, and optionally a repeat-, and an ending sequence are explicitly given in raw format, each as a separate argument. In most shells, this means that they have to be enclose within quotes.
- The sixth version dumps either the whole IRP data base, or just the protocol given as argument, to the file name used as the argument to the `--dump` option (use `-` for standard output).
- The sixth version uses the name of an IRP protocol (using the `-n` or `--name` option), to be found in the data base/configuration file specified by the `-c` or `--config` option, that protocol is used to render an IR signal or sequence using the supplied parameters (more on that later).
- Finally, the last version allows the user to enter an explicit IRP string using the `-i` or `--irp`-option, to be used to render the signal according to the parameters given.

In the simplest and most general form, parameter assignments are made on the command line in one argument of the type *`name=value`*. On both sides of the "="-signs, there should not be any spaces. (More precisely, it is required that all assignments are made within a single "argument" to the program, which is determined by the command line interpreter. Thus writing the arguments within single or double quotes, extra spaces can be parsed.) After named parameters are given (possibly none), up to four "standard" parameters can be given. These are, in order D, S, F, and T (which per convention in the JP1 community stands for "Device", "Subdevice", "Function" (also called OBC or command number), and "Toggle"). If using -1 as the value, that parameter is considered as not being assigned. One value without name defaults to `F', two values defaults to `D' and `F', three values defaults to `D', `S', and `F', and four to `D', `S', `F', and `T', in the order given. For example,

```
E=12 34 -1 56 1
```

assigns the value 12 to E, the value 34 to D, the value of 56 to F, and 1 to T, while S is not assigned anything at all. Parameters can be given not only in decimal notation, but also as hexadecimal (using prefix 0x) binary (using prefix 0b or %), or octal (using prefix 0).

If the command line cannot be parsed the usage message will be printed. If you are unsure of exactly what is wrong, consider issuing "`-d 1`" (the debug option with argument 1) as the first argument on the command line, which may produce more verbose error messages.

Using the `-r` or `--raw` option, the output is given in "raw form" (in JP1-Forum jargon, this is a sequence of positive numbers (indicating "flashes", or on-times in micro seconds) and negative numbers (indicating "gaps" or off-times, where the absolute value indicates the duration in micro seconds. Carrier frequency is specified separately). Alternatively, or additionally, using the `-p` or `--pronto` option, output is produced in the so-called Pronto format, see e.g. this document. This format is popular in several IR using Internet communities, like Promixis (known for their (commercial) products Girder and NetRemote), as well as EventGhost. Optionally, these can be wrapped into an XML skeleton, offering an ideal platform for translating to every other IR format this planet has encountered. If desired, the output of the program is directed to a particular named file using the `-o` *`filename`* or `--output` *`filename`* option. (There is also a possibility (using the `--ict` or `-I` option) to generate output files in IRScope's ict-format, but I am not sure this was as wise design decision: it may be a better idea to generate additional formats by post-processing the XML file.)

Preventing intro sequence in repeat sequence

Motivated by this thread in the JP1 forum, I have been thinking over the "correct" way to render signals of this type `... (...)+`. This is a real issue, to determine the correct behavior when e.g. a program is sent the instruction "send the signal one time", and not an academic question like "keypress shorter than 6ms" or de-bouncing circuitry.

The Pronto notation is normally described as "intro part exactly once, repetition part if and as long as the button is held down". I.e., zero or more times. Therfore, IMHO, the IRP `I (R)+` should properly be rendered as having intro sequence `I R`, which is what IrpMaster normally does. However, in a sense, this can be considered as ugly, awkward, and redundant. If I recall properly, there is a flag in the LIRC configuration called something like "send_repeat_least_once", which should be exactly what we need.

The option called `--disregard-repeat-mins` will make IrpMaster render the intro sequence without repetition part, also in the `... (...)+` case.

2.3.5.3 Iterating over input parameter ranges

Either for generating configuration files for other programs, or for testing, there is a very advanced feature for looping over input parameter sets. For all of the parameters to a protocol, instead of a single value, a set can be given. The program then computes all IR signals/sequences belonging to the Cartesian product of the input parameter sets. There are five types of parameter sets:

1. Of course, there is the singleton set, just consisting of one value
2. There is also a possibility to give some arbitrary values, separated by commas. Actually, the commas even separate sets, in the sense of the current paragraph.
3. An interval, optionally with a stride different from 1, can be given, either as `min..max++increment` or `min:max++increment`, or alternatively, simply as *, which will get the min and max values from the parameter's parameter specs.

4. Also, a set can be given as `a:b<<c`, which has the following semantics: starting with `a`, this is shifted to the left by `c` bits, until `b` has been exceeded (reminding of the left-shift operator << found in languages such as C).
5. Finally, `a:b#c` generates c pseudo random numbers between a and b (inclusive). The "pseudo random" numbers are completely deterministically determined from the seed, optionally given with the `--seed` option. As of version 0.2.2 `a` and `b>` are optional. If left out, the values are taken as from the protocol parameters min and max respectively, just as with the `*` form.

See the test file `test.sh` (include in the distributions) for some examples. Of course, using the command line, some of the involved characters, most notably the *, has a meaning to the command line interpreter and may need "escaping" by a backslash character, or double or single quotes.

There is also an option, denoted -# or `--repetitions` taking an integer argument, that will compute that many "copies" of the IR signal or sequence. This may be of interest for signals that are non-constant (toggles being the simplest example) or for profiling the program.

2.3.5.4 Debugging possibilities

There are a number of different debug parameters available. Use `-d` or `--debug` with "?" as argument for a listing:

```
$ java -jar IrpMaster.jar --debug ?
Debug options: Main=1, Configfile=2, IrpParser=4, ASTParser=8, NameEngine=16,
 BitFields=32, Parameters=64, Expressions=128,
IrSignals=256, IrStreamItems=512, BitSpec=1024, DecodeIR=2048, IrStreams=4096,
BitStream=8192, Evaluate=16384
```

For every debug option, there is an integer of the form 2^n associated with it. Just add the desired numbers together and use as argument for the `-d` or `--debug` command. There are also commands for debugging the parsed version of the IRP: Notably the `--stringtree` *`filename`* option (produces a LISP-like parsed representation of the so-called AST (abstract syntax tree)). `--dot`*`filename`* produces a dot-file, that can be translated by the open-source program `dot` contained in the [Graphviz project](), producing a nice picture (e.g. in any common bitmap format) of the current IRP protocol, and `--xmlprotocol` *`filename`* producing an XML representation. It may be possible in the future to use any of these representations to e.g., write a C code generator for a particular protocol.

Some of the classes contain their own main methods (for those not familiar with the Java jargon: these can be called as programs on their own) allowing for both debugging and pedagogical exploration, together possibly with other possibilities. In particular, this goes for the Expression class, One day I am going to document this...

```
java -classpath IrpMaster.jar org.harctoolbox.IrpMaster.Expression -d  'a + b
 *c**#d' {a=12,b=34,c=56,d=4}
(+ a (* b (** c (BITCOUNT d))))
1916
```

2.3.5.5 Third-party Java archives (jars)

For the DecodeIR-integration, IrpMaster requires a small support package, DecodeIR.jar, which is distributed together with IrpMaster. It consists of the compiled `DecodeIRCaller.java` from DecodeIR (full name `com.hifiremote.decodeir.DecodeIRCaller.class`), and `com.hifiremote.LibraryLoader.class` from RemoteMaster, which is also free software. To get rid of some (in this context) annoying messages, it was necessary to create a (very) lightly modified version, which can be found on the download page. IrpMaster also requires the runtime libraries of the parser generator ANTLR, which is also free software but licensed under a BSD-License. I distribute the whole (binary) package `antlr-3.4-complete.jar`. (No usable "runtime version" is known to me.)

2.3.6 Extensions to, and deviation from, IRP semantic and syntax

2.3.6.1 Parameter Specifications

In the first, now obsolete, version of the IRP notation the parameters of a protocol had to be declared with the allowed max- and min-value. This is not present in the current specification. I have reinvented this, using the name parameter_spec. For example, the well known NEC1 protocol, the Parameter Spec reads: `[D:0..255,S:0..255=255-D,F:0..255]`. (D, S, and F have the semantics of device, sub-device, and function or command number.) This defines the three variables D, S, and F, having the allowed domain the integers between 0 and 255. D and F must be given, however, S has a default value that is used if the user does not supply a value. The software requires that the values without default values are actually given, and within the stated limits. If, and only if, the parameter specs is incomplete, there may occur run-time errors concerning not assigned values. It is the duty of the IRP author to ensure that all variables that are referenced within the main body of the IRP are defined either within the parameter specs, defined with "definitions" (Chapter 10 of the specification), or assigned in assignments before usage, otherwise a run-time error will occur (technically an `UnassignedException` will be thrown).

The preferred ordering of the parameters is: D, S (if present), F, T (if present), then the rest in alphabetical order,

The formal syntax is as follows, where the meaning of the '@' will be explained in the following section:

```

parameter_specs:
 '[' parameter_spec (',' parameter_spec )* ']' | '['  ']'

parameter_spec:
```

```
name ':' number '.' '.' h=number ('=' i=bare_expression)?
| name '@' ':' number '.' '.' number '=' bare_expression
```

2.3.6.2 The GeneralSpec

For the implementation, I allow the four parts (three in the original specification) to be given in any order, if at all, but I do not disallow multiple occurrences — it is quite hard to implement cleanly and simply not worth it. (For example, ANTLR does not implement exclusions. The only language/grammar I know with that property is SGML, which is probably one of the reasons why it was considered so difficult (in comparison to XML) to write a complete parser.)

Persistency of variables

Graham, in the specification and in following forum contributions, appears to consider all variables in a IRP description as intrinsically persistent: They do not need explicit initialization, if they are not, they are initialized to an undefined, random value. This may be a reasonable model for a particular physical remote control, however, from a scientific standpoint it is less attractive. I have a way of denoting a variable, typically a toggle of some sort, as persistent by appending an "@" to its name in the parameter specs. An initial value (with syntax as default value) is here mandatory. It is set to its initial value by the constructor of the Protocol class. Calling the renderIrSignal(...) function or such of the Protocol instance typically updates the value (as given in an assignment, a 0-1 toggle goes like T=1-T). As opposed to variables that has not been declared as persistent, it (normally) retains its value between the invocations of renderIrSignal(...). A toggle is typically declared as `[T@:0..1=0]` in the parameter specs.

Comments and line breaks

Comments in the C syntax (starting with `/*` and ended by `*/`) are allowed and ignored. Line breaks can be embedded within an IRP string by "escaping" the line break by a backslash

Data types

The IRP documentation clearly states that the carrier frequency is a real number, while everything else is integers. Unfortunately, users of the IRP notation, for example in the DecodeIR.html document, has freely used decimal, non-integer numbers. I have implemented the following convention: Everything that has a unit (second or Hz), durations and frequency, are real numbers (in the code double precision numbers).

Extents

The specification writes ``*An extent has a scope which consists of a consecutive range of items that immediately precede the extent in the order of transmission in the signal. ... The precise scope of an extent has to be defined in the context in which it is used.*", and, to

my best knowledge, nothing more. I consider it as specification hole. I have, starting with IrpMaster 0.2.2, implemented the following: Every extend encountered resets the duration count.

Multiple definitions allowed

It turned out that the preprocessing/inheritance concept necessitated allowing several definition objects. These are simply evaluated in the order they are encountered, possibly overwriting previous content.

Names

Previous programs (makehex, makelearned) have only allowed one-letter names. However, in DecodeIR.html there are some multi-letter names. The IRP documentation allows multi-letter names, using only capital letters. I have, admittedly somewhat arbitrarily, extended it to the C-name syntax: Letters (both upper and lower cases) and digits allowed, starting with letter. Underscore "_" counts as letter. Case is significant.

Also there are a few predefined, read-only variables, mainly for debugging, although a practical use is not excluded. To distinguish from the normal, and not to cause name collision, they start by a dollar sign. Presently, these are: `$count` (numbers the call to a `render*-()`-function, after the constructor has been called), `$pass`(Requested pass in a `--pass`-argument, (or from API call), not to be confused with the following), `$state` (current state (intro=0, repeat=1, ending=2,...) of parsing of an IRP), `$final_state` (undefined until the final state has been reached, then the number of the final state). For example, the OrtekMCE example `{...}<...>([P=0][P=1][P=2],4,-1,D:5,P:2,F:6,C:4,-48m)+[...]` could be written with `$state` as `(4,-1,D:5,$state:2,F:6,C:4,-48m)+` (disregarding last frame).

GeneralSpecs, duty cycle

Without any very good reason, I allow a duty cycle in percent to be given within the GeneralSpec, for example as `{37k,123,msb,33%}`. It is currently not used for anything, but preserved through the processing and can be retrieved using API-functions. If some, possibly future, hardware needs it, it is there.

Namespaces

There is a difference in between the IRP documentation and the implementation of the Makehex program, in that the former has one name space for both *assignments* and *definitions*, while the latter has two different name spaces. IrpMaster has one name space, as in the documentation. (This is implemented with the `NameEngine` class.)

Shift operators (not currently implemented)

It has sometimes been suggested (see this thread) to introduce the shift operators "<<" and ">>" with syntax and semantics as in C. This far, I have not done so, but I estimate that

it would be a very simple addition. (The reader might like to have a look at my example, which possibly would have been more naturally expressed with left shifts than with multiplication with powers of two.)

Logical operators (also not implemented)

In particular in the light of current discussion on the F12 protocol, in my opinion more useful would be the logical operators `&&`, `||`, and `?:`, having their short circuiting semantics, like in languages such as C, Perl,..., but unless, e.g. Pascal. Recall, the expression `A && B` is evaluated as follows: First `A` is checked for being 0 or not. If 0, then 0 is returned, without even evaluating `B`. If however, `A` is nonzero, `B` is evaluated, possibly to a "funny" type and is returned. The F12 protocol (cf. the latest version 2.43 of DecodeIR.html) could then probably be written like `<...>(introsequence, (H && repetitionsequence*))` or `<...>(H ? longsequence+ : shortsequence)`.

BitCount Function

Generally, I think you should be very reluctant to add "nice features" to something like IRP. However, in the applications in DecodeIR.html, the phrase "number of ones", often modulo 2 ("parity"), occurs frequently in the more complicated protocols. This is awkward and error prone to implement using expressions, for example: `F:1 + F:1:1 + F:1:2 + F:1:3 + F:1:4 + F:1:5 + F:1:6 + F:1:7`. Instead, I have introduced the BitCount function, denoted by "#". Thus, odd parity of `F` will be `#F%1`, even parity `1-#F%2`. It is implemented by translating to the Java Long.bitCount-function.

2.3.6.3 Preprocessing and inheritance

Reading through the protocols in DecodeIR.html, the reader is struck by the observation that there are a few general abstract "families", and many concrete protocol are "special cases". For example all the variants of the NEC* protocols, the Kaseikyo-protocols, or the rc6-families. Would it not be elegant, theoretically as well as practically, to be able to express this, for example as a kind of inheritance, or sub-classing?

For a problem like this, it is easily suggested to invoke a general purpose macro preprocessor, like the C preprocessor or m4. I have successfully resisted that temptation, and am instead offering the following solution: If the IRP notation does not start with "{" (as they all have to do to confirm with the specification), the string up until the first "{" is taken as an "ancestor protocol", that has hopefully been defined at some other place in the configuration file. Its name is replaced by its IRP string, with a possible parameter spec removed — parameter specs are not sensible to inherit. The process is then repeated up until, currently, 5 times.

The preprocessing takes place in the class IrpMaster, in its role as data base manager for IRP protocols.

Example

This shows excepts from a virtual configuration file. Let us define the "abstract" protocol `metanec` by

```
[protocol]
name=metanec
irp={38.4k,564}<1,-1|1,-3>(16,-8,A:32,1,-78,(16,-4,1,-173)*)[A:0..4294967295]
```

having an unspecified 32 bit payload, to be subdivided by its "inherited protocols". Now we can define, for example, the NEC1 protocol as

```
[protocol]
name=NEC1
irp=metanec{A = D | 2**8*S | 2**16*F | 2**24*(~F:8)}[D:0..255,S:0..255=255-D,F:0..255]
```

As can be seen, this definition does nothing else than to stuff the unstructured payload with D, S, and F, and to supply a corresponding parameter spec. The IrpMaster class replaces "`metanec`" by `{38.4k,564}<1,-1|1,-3>(16,-8,A:32,1,-78,(16,-4,1,-173)*)`" (note that the parameter spec was stripped), resulting in an IRP string corresponding to the familiar NEC1 protocol. Also, the "Apple protocol" can now be formulated as

```
[protocol]
name=Apple
irp=metanec{A=D | 2**8*S | 2**16*C:1 | 2**17*F | 2**24*PairID} \
{C=1-(#F+#PairID)%2,S=135} \
[D:0..255=238,F:0..127,PairID:0..255]
```

The design is not cast in iron, and I am open to suggestions for improvements. For example, it seems reasonable that protocols that only differ in carrier frequency should be possible to express in a concise manner.

2.3.6.4 The Configuration file/IRP protocol database

There is presently not a "official" IRP database. MakeHex comes with a number of protocol files with the `.irp`-extension, but that is another, obsolete and much less powerful format. MakeLeaned also comes with a number of "irp-files", in the new format, but incomplete. The DecodeIR.html-file presently comes closest: it has a number (upper two-digit) of IRPs, however, often not even syntactically confirming to the specification, and often with the description of the protocol at least partially in prose ("C is the number of ..."), parseable only by humans, not by programs.

Possibly as an intermediate solution, I invented the `IrpProtocols.ini` file. This file has a format similar to ini-files under Windows. For every protocol, it contains name and an IRP-string, possibly also a documentation string. The latter can, in principle, contain HTML elements, i.e. it can be an HTML fragment.

2.3.6.5 Syntax and semantics of the IrpProtocols.ini file

Every protocol is described in a section starting with the key `[protocol]`. Then there are a few keywords describing different properties:

- `name` The name of the protocol. This is folded to lowercase for searches and comparisons.
- `irp` The IRP string representation. This may continue over several lines if the line feeds are escaped by a backslash ("\"), i.e. having the backspace as last character on the line.

Other keywords are allowed, but ignored. Then, optionally, there may be a section `[documentation]`, that, in principle, could contain e.g. an HTML-fragment. The documentation section continues until the next `[protocol]` is encountered.

2.3.6.6 Requirements for an IRP data base

I have created the present `IrpProtocols.ini` by hand editing the DecodeIR.html-file. I would welcome if the community can settle for one endorsed format for such a data base. It can be one file, or many files: One file per protocol is easier for the developer, in particular if several developers are using a version management system (with or without file locking), but less convenient for the user.

It would be highly desirable in the future to be able just to maintain one file (or set of files). Some possibilities for this are:

1. Have one master file, for example in XML format, that after preprocessing *generates* both DecodeIR.html, and a protocol description file. There is also the possibility of having a program like IrpMaster parsing the master file directly.
2. Extend `protocol.ini`("belonging to RemoteMaster") with the IRP information. Leaves the problem of duplicated "documentation" between DecodeIR.html and protocols.ini.
3. Formalizing the IRP-Strings within DecodeIR.html, e.g. by using div or span elements with class-attributes, (and formatting with, for example, better CSS style sheets) so that the IRP information can be unambiguously read out.

2.3.6.7 Integration with DecodeIR

Optionally (when installed and selected with the `--decodeir` option) the computed IR signal is sent to DecodeIR, to check DecodeIR's opinion on the nature of the signal. This gives a magnificent possibility for automated tests, not only of the present program, but also of DecodeIR. Note in particular that there are very advance possibilities for testing not only a single signal, but for testing whole ranges of signals, a list of signals, "random" inputs, equidistant inputs, or inputs achieved by shifting, see the section on parameter iterating.

The shared library is sought first in architecture dependent sub-directories, like in RemoteMaster, `.\windows` on Windows, `./Linux-amd64` and `./Linux-i386` on

64- and 32-bit Linux respectively, etc, then in system libraries, for example given on the command line to the Java VM, using the `-Djava.library.path=` option.

There is some fairly hairy programming in `DecodeIR.java` for identifying some different cases.

The enclosed script `test.sh` runs under a Unix/Linux shell such as bash or sh. It should also run within Cygwin on Windows. It does not run with the standard Windows command line interpreter. Note that it might need some adjustment of file paths etc.

Possibly because I did not find any more logical way to dispose it, the current distribution contains a class (with main()-method) named EvaluateLog that can be used to evaluate the output of the above script. Use like

```
java -classpath IrpMaster.jar IrpMaster/EvaluateLog protocols.log
```

2.3.7 The API

The Java programmer can access the functionality through a number of API functions.

The class `IrpMaster` is the data base manager. The class is immutable, constructed from a file name (or an InputStream), and can deliver assorted pieces of information from the data base. Most interesting is the `newProtocol()`-function that generates a Protocol-object from parsing the IRP-string associated with the requested protocol name. It contains a very elaborate `main()`-function for command line use -- strictly speaking this is "the program" that is described herein. Actually, that `main()`-function does not necessarily belong to the `IrpMaster` class, but could be located somewhere else.

Instances of the Protocol class are constructed (essentially) from a String, containing the IRP representation to be parsed. Once constructed (and IRP-String parsed), the Protocol instances can render IrSignals and IrSequences for many different parameter values. This is done with the `render(...)` and `renderIrSignal(...)` functions, producing `IrSequences` and `IrSignals` respectively:

An `IrSequence` is a sequence of pulse pairs. It does not know weather it is supposed to repeat or not. In contrast, an `IrSignal` has one introductory `IrSequence`, one repetition `IrSequence` (either, but not both, of these can be empty), and an (in most cases empty) ending `IrSequence`.

The API is documented in standard Javadoc style, which can be installed from the source package, just like any other Java package. For the convenience of the reader, the Javadoc API documentation is also available here.

2.3.7.1 Example of API usage

The task is to write a command line program, taking, in order, the configuration file name, a protocol name, a device number and a function/command/obc number, and send the corresponding IR signal to a GlobalCaché GC-100-06 networked IR-transmitter, having IP address 192.168.1.70, using its IR Port 1. For this, we use the GlobalCaché

functionality of the Harctoolbox, which is also GPL-software written by myself. This task is solved with essentially just a few lines of code.

2.3.8 References

1. IrScrutinizer. A program, also by myself, than, among other things, provides a user friendly GUI for IrpMaster.
2. IrMaster. A program, also by myself, than, among other things, provides a user friendly GUI for IrpMaster.
3. Specification of IRP Notation, Graham Dixon. Also in PDF version for download. A very thorough specification.
4. Discussion thread on the IRP documentation
5. DecodeIR.html. (The link points to a slightly nicer formatted wiki page, though). Contained within the current distribution of DecodeIR.
6. Makehex. Source, binary. A functional predecessor of the present program. Operates on a predecessor of the current version of the IRP. Written in C++, also available as DLL (within the first link). Java translation by myself.
7. MakeLeaned. Windows, binary only, source unavailable. GUI only, no API. Not maintained since 2005. Almost certainly incomplete with respect to current IRP specification. Discussion thread in JP1-Forum.

2.4 The Girr format for universal IR Commands and remotes.

Date	Description
2014-01-28	Initial version.

Table 1: Revision history

2.4.1 Background and Introduction

There are several Internet sites in whole or in part dedicated to infrared control of customer electronics. Very soon the question on exchange of IR signals, individual or as a set of commands from one remote or one device, comes up. For individual IR signals, the CCF format, also called Hex or Pronto Hex, is the one most used. This describes *one* signal, without a name or any other attributes. To use, the user will most likely have to copy-paste the information from a downloaded file, or a forum contribution, into his/her application program. For several signals, this unsystematic procedure is both tedious and error prone.

Some manufacturers publish the IR commands for their products, often as tables as Excel list or as PDF documents. There are also some quite impressive user contributed collections around in Excel format, e.g. for Sony and Yamaha equipment. Often, these lists contain not only the CCF form, but even a protocol/parameter form. These lists definitely mark a step in the right direction. With sufficient skills with the involved tools it is often possible to transfer a whole set of commands, possibly even preserving

names, with a few clicks. However, this is still a manual process, that is not suited for automation.

On the other hand, there are a few file formats around, describing a complete setup of a programmable remote control, like the Philips Pronto CCF file format or the XCF format of the Pronto Professional. These describe a complete setup, including layout of buttons and pages, font selection and other items not of interest for the exchange of IR signals. The "device updates" (rmdu-files) of RemoteMaster also falls into this category: They do contain the IR Signals, either as a raw representation or in a protocol/parameter format, but also a number of key bindings, more-or-less specific to a particular JP1 remote.

The Lirc project however has a data base file format, containing named commands, grouped into named "remotes". Unfortunately, the Lirc format was never intended as an exchange format, and, as a general rule, only Lirc program can read Lirc files. Also, Lirc has a flawed concept of repeat sequences.

This leads to our mission:

2.4.1.1 Mission

To define a universal format for the exchange of IR signals, encompassing both for protocol/parameter form, and different textual formats, like CCF. The format should describe the IR signals with their names, (not their semantics). The commands should be bundled within "remotes". It should be readable and writable by both humans and programs. The format should be free/open for everyone to implement, in open or proprietary contexts. It should use open technology, so that tools can be implemented using currently spread free software.

Everyone is invited to implement this format in other programs, or tools for the format.

2.4.2 Program support

Jirc, which is a translation of substantial parts of the Lirc project to Java, generates Girr files from Lirc files.

IrScrutinizer uses Girr as its preferred format for import and export of IR signals. It can import and export from many different file formats and data bases.

My earlier programs IrpMaster and IrMaster generate XML format output, that is similar, but unfortunately not compatible with Girr. (It can be considered as a predecessor to Girr.) However, the distribution contains an XSLT stylesheet `irpmaster2girr.xsl` that can be used to translate to Girr format.

I plan to extend the RemoteMaster program (for programming and configuring Jp1 remotes) with the possibility to export and import Girr files.

I also hope to be able to integrate Girr import into the OpenRemote project, namely its database ("Beehive").

2.4.3 Copyright

The rights to the described format, as well as the describing file are in the public domain. That also goes for the present document. Note that this is in contrast to other documents on www.harctoolbox.org for which no copying or re-distribution rights are granted, or the therein contained software, which is licensed under the Gnu General Public License, version 3.

2.4.4 The name of the game

Pronounce "Girr" as one word (not G.I.R.R.), but otherwise any way you want. It should be used as a proper noun, capitalized (not uppercase). Preferred file extension is `girr`, but this is not necessary. Also, e.g. `xml` is possible.

2.4.5 Requirements on a universal IR command/remote format

It should be an XML file determined by an XML Schema. It should, however, be usable without validating parsers etc.

The formal rules (enforced by Schema) should be as non-intrusive as possible, possibly prohibiting "silliness", but otherwise requiring at most a minimum of formal syntactic sugar. For this reason, it was decided not to use XML namespace, since it seem to infer another level of difficulty, not only for defining the language, but also for handling instances and writing tools.

A remote is in principle nothing else than an abstract remote, having a number of commands. In particular, it should not determine the semantics of the commands, nor does it describe how to control a device that can be commanded by the said remote. Names for commands can be "random", in any language, using any printable characters including white space. A well defined semantic of command names is not granted. However, in some cases uniqueness in the purely semantic sense is required, for example ensuring that all commands within a particular remote have unique names.

It can be assumed that all signals consists of an intro-, an repeat-sequence (any of which, but not both, may be empty), and an optional ending sequence.

It should be possible to describe signals either as parameterized protocols, in raw form, or in CCF form. If several forms are present, it should be clear which one is the primitive form, from which the others are derived.

It should be suitable both for human authoring (with a minimum of redundancy), as well as machine generation (where simple structure may be more important than minimum redundancy).

It should be a container format, namely extensible with respect to textual representation of IR Signals and -sequences.

2.4.6 Demarcation

- The present work aims at a description for remotes, not devices (e.g. in the sense of this). Thus, command names are free form strings, with no semantics inferred.
- Only unidirectional "commands" are considered, not data communication.
- It is only attempted to define IR signals and "sufficiently similar" signals. One such signal/sequence consists of a sequence of durations, namely alternating on- and off-times. Except for the "normal" IR signals, this includes RF signals of frequencies 433, 318, 868 MHz etc. used e.g. for controlling power switches.

2.4.7 Informal overview of the Girr format

There are four different high-level element in the format: `remotes`, `remote`, `commandSets`, and `commands`. All can be the root element of a conforming Girr document, although all software may not handle all of them. Basically, the element `remotes` contains one or more `remotes`, each containing one or more `commandSets`, each containing either other `commandSets` and/or `commands`.

2.4.7.1 command

This element models a command, consisting essentially of a name and an IR signal, in one or several different representations. Names can consist of any printable characters including white space, and carries a priori no semantics.

Consider the following example:

```
<command name="play" displayName="Play |&gt;" comment="" master="parameters">
    <parameters protocol="nec1">
        <parameter name="D" value="0"/>
        <parameter name="F" value="0"/>
    </parameters>
    <raw frequency="38400" dutyCycle="0.50">
        <intro>+9024 -4512 +564 -564 +564 -564 +564 -564 +564 -564 +564
               -564 +564 -564 +564 -564 +564 -564 +564 -1692 +564 -1692 +564 -1692 +564
 -1692
               +564 -1692 +564 -1692 +564 -1692 +564 -1692 +564 -564 +564 -564 +564
 -564 +564
               -564 +564 -564 +564 -564 +564 -564 +564 -564 +564 -1692 +564 -1692 +564
 -1692
               +564 -1692 +564 -1692 +564 -1692 +564 -1692 +564 -1692 +564 -39756
         </intro>
         <repeat>+9024 -2256 +564 -96156</repeat>
    </raw>
    <ccf>0000 006C 0022 0002 015B 00AD 0016 0016 0016 0016 0016
          0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016
          0016 0041 0016 0041 0016 0041 0016 0041 0016 0041 0016
          0041 0016 0041 0016 0041 0016 0016 0016 0016 0016 0016
          0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016
          0041 0016 0041 0016 0041 0016 0041 0016 0041 0016 0041
          0016 0041 0016 0041 0016 05F7 015B 0057 0016 0E6C
    </ccf>
    <format name="uei-learned">00 00 2F 00 D0 06 11 A0 08 D0 01 1A
           01 1A 01 1A 03 4E 01 1A 4D A6 11 A0 04 68 01 1A BB CE 22
```

```
            01 11 11 11 12 22 22 22 21 11 11 11 12 22 22 22 23 82 45
    </format>
</command>
```

(Details on syntax and semantics are given in the next section.)

In the `parameters` element, parameters and protocol can be given. They can be completely given, or they may be inherited from parent element of type `commandSet`.

The raw and the CCF form may be given next, as above. Finally, one or may auxiliary formats of the signal can be given, here `uei-learned`.

Fat Format

For the ease of further processing of the result, the sequences within the `<raw>` element can alternativelly be given in the "fat" format, where each flash (on-period) and each gap (off-period) are enclosed in their own element, like in the following example:

```
<command name="play" displayName="Play |&gt;" comment="" master="parameters">
    <parameters protocol="nec1">
        <parameter name="D" value="0"/>
        <parameter name="F" value="0"/>
    </parameters>
    <raw frequency="38400" dutyCycle="0.50">
        <intro>
           <flash>9024</flash>
           <gap>4512<gap>
           <flash>564</flash>
           <gap>564</gap>
           <flash>564</flash>
           <gap>564</gap>
```

2.4.7.2 commandSet

`commandSets` bundles "related" commands together. They may contain `parameters` elements, in which case the values are inherited to children `commandSets` and their contained `commands`.

Although a `remote` cannot contain `commands` directly — it must contain a commandSet — the use of `commandSets` is somewhat arbitrary. They can be used e.g. to structure a remote containing a few different protocols, or one protocol and a few different device numbers nicely, in particular if hand writing the Girr file. However, protocol and their parameters can also be given as `parameters` within the `command` element.

2.4.7.3 remote

A `remote` is an abstract "clicker", containing a number of `commands`. The name of the contained commands must be unique, even across different `commandSets`.

2.4.7.4 remotes

`remotes`, as the name suggests, is a collection of `remotes`, identified by a unique name.

2.4.8 Detailed description of syntax and semantics of the Girr format

This article describes the Girr format version 1.0, identified by the attribute `girrVersion`, expected in the root element of an instance. (Not to be confused with the version of the support library.)

The Scheme file contains internal documentation of the semantics of the different elements.

2.4.8.1 Automatically generated documentation

Browsable schema documentation (thanks to Gerald Manger).

2.4.9 Stylesheets

A Girr file can be viewed in the browser, provided that it is associated with a style sheet. This is either a cascading style sheet (css), which essentially tells the browser how different elements are to be rendered, or an XSLT style sheet, which internally translates the XML document to a HTML document, normally with embedded style information. A description of these techniques is outside of the scope of the current document (see this document as an introduction); an example is given as simplehtml.xsl.

To use, add a line like

```
<?xml-stylesheet type="text/xsl" href="simplehtml.xsl"?>
```

to the Girr file. (Some programs, like IrScrutinizer, can do this automatically.) Note that some browsers, like Firefox, for security reasons limits the usage of style sheets to the current directors.

XSLT style-sheets can however be used for other purposes than the name suggests. Included in the distribution is a "stylesheet" `irpmaster2girr.xsl` that translates the XML code from IrpMaster and IrMaster to Girr format.

2.4.10 Supporting Java library

For importing and exporting Girr files to Java programs, a Java library is provided. It is documented by its Javadoc documentation. As opposed to the specification as such, it is licensed under the Gnu General Public License, version 3.

Presently, only import of documents having remotes as root element is supported.

At the time of writing, the library carries the version number 1.0.0.

The library requires the IrpMaster classes, i.e., the `irpmaster.jar` file.

Sources download.

2.4.11 Appendix. Parameterized IrSignals

The purpose of this section is to make the article more self-contained. Information herein are described in greater detail elsewhere.

The Internet community has classified a large number of IR Protocols, see e.g. this listing. These protocols consist of a name of the protocol, a number of parameters and their allowed domains, and a recipe on how to turn the parameters into one, two, or three IR sequences, making up an IR signal. This recipe is often expressed in the IRP Notation, which is a compact formal representation of the computations involved. For particular values of the parameters, a rendering engine computes the resulting IR signal, often in CCF, also called Pronto Hex format, or in raw format.

2.5 General InfraRed Server Command Language

Date	Description
2014-01-06	Initial version.
2014-09-16	Added the communication section.

Table 1: Revision history

2.5.1 Requirements

- Demarcation: This deals only with sending, receiving (including decoding), storing etc of IR signals. Not: serial and other text base communication, nor the acting on received signals.
- However, RF signals for remote control are included, since they only differs from IR signals by using another carrier signal.
- Modularized, named modules containing commands (like Java interfaces). Written in capitalized CamelCase.
- Inheritance within modules, multiple inheritance
- Extensible: Developers can define new modules
- Only the Basic module mandatory, containing the commands `version` and `modules`.
- Very low-weight, should be implementable on e.g. Arduino.
- As "dumb" as possible.
- Basic version: text socket/serial interface. Versions using json, xml/soap, http/rest possible.
- Authentication as optional module, several submodules for different sort of authentication.
- Command structure: `command [subcommand] [options] [arguments]`

- Response structure: TBD.
- Names for IR commands, hardware: arbitrary strings using the English language, case sensitive matched using charsets.
- Our command names: C-syntax; lowercase only, underscore discouraged. “get” and “set” left out unless necessary for uniqueness or understandability (like `getcommand`).

2.5.2 Specification

Typography: module names are in **bold**, command names `monopitch`.

2.5.2.1 Introduction

This list is not an unrealistic Christmas wish list, but a list of modules, only the first one mandatory. Through the module concept, a conforming GIRS server can be anything from an Arduino with just an IR sender LED and a sketch a few pages long, and a fat server with several input- and output-devices, (each) having several transmitters, combined with a full blown data base, with user administration and authentication.

Note that there is a number of properties for e.g. LIRC that has been rejected here, in particular the ability to execute commands. (These should be handled by another program.)

A capable server should probably also implement some sort of discovery beacon, for example AMX style.

2.5.3 Modules

2.5.3.1 Base

This is the only mandatory module.

`version`

- returns manufacturer, manufacturer's version number, or another useful version string.

`modules`

- returns list of implemented modules, separated by whitespace.

2.5.3.2 NamedRemotes

Support of remotes identified by name, like LIRC.

`remotes`

- argument named/uei: What type of remotes to report.
- returns: list of remotes, either names or manufacturer/device-type/setupid

`commands`

- argument: remote in a supported format (mandatory)
- returns: tab(?) separated list of command names, in currently selected char set.

`database` (module database)
argument: data base name. Required.

2.5.3.3 UeiRemotes

Support of remotes identified by manufacturer, device type (both arbitrary strings), and a setup number (most commercial data bases)

`manufacturers`

- returns: tab separated list of manufacturers.

`devicetypes`

- argument: manufacturer
- returns: tab separated list of device types

`setupids`

- arguments: manufacturer, device
- return list of setup ids.

`database (module database)`

- argument: data base name. Required.

`database-password` (module database)

- argument:password.

2.5.3.4 OutputDevices

Allows for accessing several devices; several instances of the same type: Names like “Greg's GlobalCaché”. (Configuration of these over this API is not intended.) Each has their own set of transmitters.

`outputdevices`

- returns:list of known devices

`outputdevice` Set default output-device

- argument: device name

`outputdevicecapacities`

- argument device name (optional, defaulted)
- result: list of capacites. Possible values (extensible): fmax, fmin, zero_frequency_tolerant. Inherit to transmitters.

2.5.3.5 InputDevices

Allows for accessing several devices; several instances of the same type: Names like “Greg's GlobalCaché”. Configuration of these over this API is not planned. An input device does not possess transmitters.

`inputdevices`

- returns: list of known devices

`inputdevice` Set default input device

- argument: device name

2.5.3.6 Transmitters

Same commands as OutputDevices. (????)

`transmitters (module transmitters)`

- argument: output-device

`output-device`

- returns: list of transmitters, max-number-transmitters-enabled

`settransmitters` Selects default transmitter for the output device selected. ***(TBD: Alternative: ditch the default transmitter and this command, thus transmitter argument mandatory.)***

`transmittercapacities`

- arguments:
  - output-device (optional, use default if not given)
  - `transmitter` transmitter (only one!)
- result: list of capacites.Possible values (extensible): ir (connected to IR LED). Rf (connected to RF modulator) hard-carrier=frequency (in particular for RF, 433M, 868M (Hz or suffix M,k)). Inherits from outputdevicecapacities.

2.5.3.7 Transmit

Access may be restricted through user rights. There is always a default output device; if the **OutputDevices** module is implemented, there may be more.

`transmit` (semantic for repeats may be implementation dependent)

- subcommands (at least module (??) has to be implemented):
  - `ccf` (module ccf). Parameter: CCF string
  - `raw` (module raw). Parameter: frequency, duty cycle, intro, reps, ending.
  - `Irp` (module irp). Parameters: protocol name OR irp-protocol, parameters.
  - `Name` (module named-command). Parameters: remote (one of the suppoted formats), command name
- options:
  - transmitters (module transmitter) (optional (or not?))
  - output-device (optional, otherwise use default)
  - transmit-id (module transmit-id) (optional)
  - # sends (default 1)
  - wait (wait for command completion)
- returns: (after completion) confirmation command, with transmitter and transmit-id

`stop` (module `Stop`)

- Argument: output-device, transmitter, transmit-id (optional)

2.5.3.8 Capture

for capturing ("learning") of new remotes. Dumb command, intelligence should sit in the calling program.

`analyze`

- Arguments: (all having sensible defaults.)
  - input-device
  - start-timeout
  - capture-timeout
  - ending-timeout
- Returns: frequency, raw ir-sequence, optionally duty cycle.

2.5.3.9 Receive

for receiving commands, possibly for deployment solutions. Dumb command, intelligence should sit in the caller. Identifying start separately (like for volume control) not supported.

`receive`

- Arguments:
  - return format(TBD)
  - input-device
  - timeout
  - filter, syntax, (syntax, semantics TBD)
- subcommand named (module named-command)
  - Return value: received command name (+ remote)
- subcommand decode (module decoder)
  - Return value: protocol name, parameters

`relay` (module relay), to send events to other servers

- Arguments:
  - return format (TBD)
  - `protocol` (http/tcp/udp/shell?)
  - portnumber
  - ipaddress
  - filter (TBD)

2.5.3.10 Store

allows for uploading new commands to the server. May be restricted through authentication and user rights.

`store`

- arguments: data base (optional), name, remote (in a supported format), signal in a form dependent on the subcommands.
- subcommands
  - ccf (module ccf)
  - raw (module raw)
  - irp (module irp(?))

`commit` Stores the recently downloaded commands persistently.

- Argument: data base name (optional)

2.5.3.11 Command

allows for downloading commands from the server. Inverse of store. May be restricted through authentication and user rights.

`getCommand`

- Argument:
  - data base (optional)
  - output format: ccf, raw, irp,... (also other can be supported)
- Return: command in desired format.

2.5.3.12 Authentication

Several different models for access control are possible, and can be implemented through different modules. The first just requires a password to all the services. The second allows user based restrictions: Some commands/subcommands/arguments might be restricted to some users. Of course, sending passwords unencrypted over the net is not to be considered very secure, so preferably ssh or similar, or a challenge-response system should be used.

`login (module password), for password protected services`

- argument: password

`login` (module UserPassword), for user/password protected services, possibly with different rights for different users.

- argument: user, password

`sshlogin` TBD (module ssh)

`logout`

2.5.3.13 Charset

Determines charset used for input and output.

`charset`

- argument: charset name.

2.5.4 Communication

Communication is typically taking place over a bidirectional ASCII stream, like serial, "terminal", connection or a through a TCP socket. The commands sent to the GIRS server should be of the form `command [subcommand] [options] [arguments]`, where `command` can be abreviated as much as unabiguity allows (typically to the initial character). The form of the responses should be a "natural" ASCII response in the form of one line (typically); the tokens separated by whitespace.

Interacting with a GIRS server through static or dynamic linking can also be possible, either by decoding a command line, or with a number of API functions.

2.6 Architecture Concept

Date	Description
2014-09-20	Initial version.

Table 1: Revision history

2.6.1 Preliminaries

2.6.1.1 Sending/Receiving

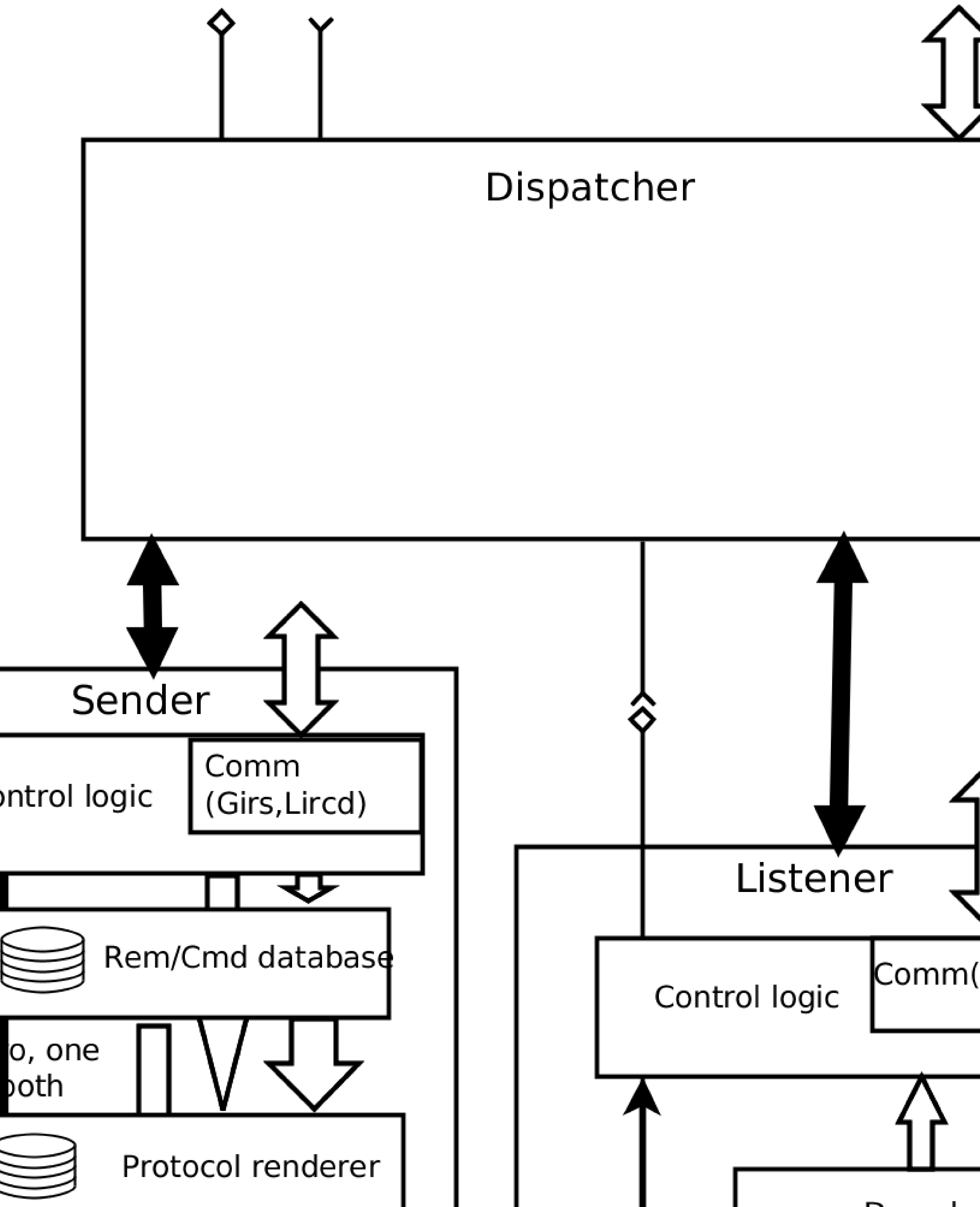
Traditionally, IR hardware and software combines sending and receiving capacities in one unit — unless of course only one of these functinallities are supported. Instead, we argue that sending and receiving of IR signals are fundamentally different activities which preferrably are threaded separately. Of course, this does not prohibit a particular implementation to implement both of these aspects, just like a program may contain both a wordprocessor and a music player.

2.6.1.2 Learning

Capturing (often called "learning") should be taken out of the requirements. Capturing and analyzing unknown signals is a completely different use case from (deployment) sending and receiving of IT signals. There are other software, like the IrScrutinizer that are optimized for this use case. Capturing of the signals of a present remote is done during installation time, the result saved to a data base used by the sender and/or receiver. (Alternatively, in very many cases, the IR signals for a particular device/remote can be found in public or propritary data bases found on the Internet.).

2.6.2 Main Concept

A three-component model consisting of a *sender*, a *receiver*, and a *dispatcher*, as shown in the figure below, is proposed. These should be designed as exchangable components.


Dispatcher
Sender
ontrol logic
Comm
(Girs,Lircd)
Rem/Cmd database
o, one
oth
Protocol renderer
Listener
Control logic
Comm(

2.6.2.1 The Sender

Top level Interface

The sender encapsulates sending of IR signals. It should be considered as a server, communicating with its clients using the Girs command language. Basically, it is sent commands to send particular IR signals through the IR hardware it commands. This is specified in the Transmit module of Girs. With moderate effort, it would is also be possible to implement the somewhat more limited protocol of the Lirc daemon Lircd.

The specification of the to-be sent IR signals can take place in three different ways:

- Raw IR sequence/signal; basically equivalent is the CCF, also called Pronto Hex, format. This is simply a list of durations in microseconds of interleaved pulse and gap durations.
- Protocol/Parameter format.
- Remote/Command format, this is simply the name of a remote or device, together with the name of its commands.

It is possible that a particular implementation is not implementing all three forms, but just one or two. However, leaving out the raw format is probably not productive.

Internal behavior

It is assumed that suitable hardware is connected to the computer running the sender. Note that it is possible and sensible that a driver implemented on separate hardware (e.g. an Arduino) also communicates using Girs.

A "low level send driver" accepts commands of sending a certain, raw, IR Sequences, using a particular transmitter. A "high level driver" adds the capability to send an IR signal consisting of an intro sequence, a repeat sequence, and (in some cases) an ending sequence a certain number of times, as well as the possibility to interrup an ongoing transmission of a repeating signal.

Timing issues are the sole responsibility of the driver layer. I.e., the upper layers deliver sending requests to the driver, and the driver executes the requests. After a request has been made, a stop request may however be issued (if the software implements it).

In the case of a raw transmission requested, the control logic directs the IR sequence or the IR signal to the send driver.

If protocol/parameter form is implemented, the sender contains a protocol renderer and possibly a data base of protocols, computing the raw signal corresponding to the requested protocol/parameter combination.

If the remote/command format is implemented, the sender has to contain a data base, like the LIRC configuration file(s). If the data base contains the already rendered signals, these can be send directly to the driver. If a protocol renderer is implemented, it is also possible for the remote/command data base to contain protocol/parmeter data, which are sent to the protocol renderer, which in its turn forwards it to the driver.

2.6.2.2 The Listener

Top level interface

The listener encapsulates the receiving and decoding of IR signals, and its translation into an event or command response.

Often, a received and correctly identified IR signals is expected to generate an event of some kind, for example a Linux input event. Alternatively/additionally, the listener can implement a Girs server, and respond to Receive commands.

Internal Behavior

It is assumed that suitable hardware is connected to the computer running the listener. Note that it is possible and sensible that a driver implemented on separate hardware (e.g. an Arduino) also communicates using Girs.

The driver delivers its received signals as sequencies of pulses and gaps. This data is delivered to a *decoder*, which decodes the sequences either as a protocol/parameters set, or a remote/command set, analogusly with the sender, which was described above.

2.6.2.3 The Dispatcher

An IR system like Lirc not only sends IR signals and reacts on received IR signals, it also invokes other actions, like starting programs etc., in the response to received IR signals. Lirc even can act on other input events, as if they where received IR signals (see the section below. Instead, we here advocate the separation of generating/receiving IR signals and acting on them. The component for handling the events we will call the *dispatcher*. Since the demarcation to general home/computer automation and remote control is unclear, we will not go into any details.

Top level interface

The dispatcher can receive events and messages, not only from the IR sender and listener here, but in the general case, also from other sources, like other sensors or input events. From this input, it can generate other events, invoke other programs, send messages over the network, etc.

2.6.3 Implementation notes

2.6.3.1 Sender

Drivers

There exists a number of "drivers" (or "plugins") for IR hardware for different programs. Unfortunately, these are not always possible to use in another context. For example, the Lirc dynamically loaded drivers are not meant to be used outside of Lirc, for example, they do not implement a simple send of an IR sequence (instead there needs to be a

`remote`, and a `command`). To use these drivers (for the reason given, "plugin" is really the better term), it will be necessary to create a "mini-Lirc" to support them. These are by definition for Linux, or at least a Unix-like operating system. Also WinLirc and Eventghost should be examined for its possiblity to "donate drivers".

Our package HarcHardware contains some Java drivers for some IR hardware, that can be used more or less directly.

Protocol Renderer

It is fairly straight-forward to write a simple renderer for a particular protocol, like NEC1 or RC5. A very advanced general and extensible renderer is IrpMaster, which is a GPL3 licensed Java program.

Remote/Command data base

A format is needed for importing (and possibly exporting) of IR signals. Another format is needed for internally persistently storing the signals, for example to a disk file. These formats may or may not coincide. As external import format, the Girr format is suggested, or possibly a restriction thereof — for example a implementation without a protocol renderer should require that all signals are present in either raw or Pronto Hex (CCF) format.

For migration of Lirc configuration files, IrScrutinizer can be used.

2.6.3.2 Listener

Communication logic

It would be possible to implement a Lirc compatibility mode by writing on the Lirc socket, typically `/var/run/lirc/lircd`. That way, Lirc "client programs" like `irexec`, can be (re-)used.

Drivers

Many (most?) drivers for IR receiving hardware are not usable to receive general IR signals (not even with "normal" and known modulating frequency). Instead, they try to decode the signal itself, the react only on their "own" protocol, and in the case of a match (and only then!), they deliver a decode, typically as an integer. In Lirc, these drivers are called LIRCCODE drivers.

Typically, the hardware is not intended to be a "generic" component, but may be e.g. a TV card with an IR receiver, just intended to react to a bundled hardware remote.

This type of driver does not fit into the model here. Instead, it may be possible to turn such a driver into a "listener" in its own right, sending events ("received command 42 from the TV card") to a dispatcher.

Otherwise, the comments in the sender sections apply here too.

Decoding

Decoding can take place either protocol-oriented or command-oriented (trying to determine which one of the know commands that fits). It is believed that the first one is the more systematic, and normally the better approach, so we will only consider it here.

It is fairly straight-forward to write a decoder for a particular standard protocol, like NEC1 and RC5. A very versatile decoder is DecodeIr, knowing over 100 different protocols. It is widely used and tested. Unfortunately, partially due to its chaotic code base, it is effectively not maintainable nor extendable, and its API also has some problems.

2.6.3.3 Dispatcher

There are a number of possiblities to implement a dispatcher. The Lirc program `irexec` is a simple such. I have also written a (presently not published) simple dispatcher in Java, presently reacting on IR signals received from an Arduino, generating net events etc. as configured from an XML file. The OpenRemote project contains a rule engine based on Drools giving very interesting possibilities for elaborate "dispatching". For Windows users, invoking Eventghost as dispatcher is also an interesting option. This program allows the programming of e.g. if-then-else rules with simple graphic programming. (When will this clever — Python! — program be ported to non-Windows?)

2.6.4 Comparision with Lirc (Lircd)

The daemon Lircd takes the role of all of the components sender, listener, and (to some extent) dispatcher.

Lirc listens for sending requests either on a Unix domain socket (typically `/var/run/lirc/lircd`), or a TCP socket (default 8765). Sending request can be generated by a Lirc client like the command line program `irsend`. (Another Lirc client, implemented in Java, is found in our HarcHardware package, in the class LircClient. This is integerated in IrScrutinizer.) A sending request contains a remote/command combination, together with a number of repetitions. It will use the data base ("configuration file") to render the IR signal.

When a (reading) client opens the Lirc socket (the Unix domain socket or the TCP socket), Lircd starts listening to IR signals. If a signal arrives, it is tried to decode it to any of the known commands in its data base ("configuration file"). If decoding is successful, the name of the identfied remote/command is written to the communication socket. Lircd may also send events to another daemon (`irexec`) that can invoke other actions, like starting certain programs or invoking other events, like X Window system events. It is even possible to have Lircd injecting events into the Linux input layer.

Using the `devinput` driver and a `/dev/input/event`*N* input device, any Linux input device can be cloaked as an IR receiver. This may be an IR receiver with kernel (-module) support (like the IguanaIR or a MCE receiver), but may also be a completely

different kind of animal. In this use case, The Linux kernel takes the role of our listener, in some cases even explicitly decoding protocols such as NEC1 and RC5, while the Lircd daemon is nothing else than a dispatcher.

2.7 Glossary and terms

Date	Description
2013-12-01	Initial version.
2013-12-20	More stuff added.
2014-02-02	Even more stuff added.
2014-09-18	Yet more stuff added.

Table 1: Revision history

2.7.1 Glossary

Here we explain and define some of the used terms. In most cases (but not all!), this correspond to established usage in the Internet, e.g. in the JP1 forum. It should also be pointed out that in some cases, in particular when comparing programs by others to my own, the assessment should be considered as subjective.

Substantial program/program packages are written captalized, (not uppercase), in some case in CamelCase, and in Roman typeface, like a proper noun. "Small" programs are written as `code`. For example, we write "Lirc", not "LIRC", or `lirc`; and `irsend`.

AMX Beacon
A daemon program implemented in some networked components. It is used for periodically announce their existence, and some of their properties.

Analyzer
The part of the ExchangeIR library that analyzes a raw IR Signal. If successful, it returns an IRP form, together with parameter values. The returned IRP form may or may not correspond to a known protocol.

Arduino
A single-board microcontroller, based upon a 8-bit Atmel AVR microcontroller, or a 32-bit Atmel ARM. Official site. Can be used for implementing sending and/or capturing of infrared signals with additional hardware attached to its IO-pins.

Capturing (sometimes "learning")
The process of recording an unknown physical IR sequence as a sequence of on- and off-durations, preferably together with a measure of its modulation frequency. Cf. receiving IR signals.

CCF (text signal format)
IR signal format. Not to be confused with the ccf file format of the Pronto Classic remotes! Often called "hex", "Pronto", or "Pronto hex". Consist of a sequence of

four-digit hexadecimal numbers. For the interpretation, see the Appendix. It is a very popular format, e.g. for textual the exchange in Internet forums.

ccf file format

Not to be confused with the CCF text signal format! Configuration file format for the Pronto Classic. File extension is `ccf`. Can be edited by the ProntoEdit program. Has been completely reverse engineered, and the open-source program Tonto is able both to interactively edit them, as well as non-interactively through an API. IrScrutinizer can import and export ccf files, using the said API.

Cleansed signal

Given a captured IR sequence, using the repeat finder, all occurrences of the repeat sequence but one are removed, and numerically "close" duration values are lumped into one single value.

Command

Here, an IR signal with a name, like "Play".

Consumer IR (CIR)

Consumer IR deals with IR control of various devices. Wikipedia article. Not to be confused with IRDA. Typically uses wave lengths of 930-950nm.

CVS (comma separated values)

Primitive data base format, one record consisting of one line, the entries separated from one another by a comma (,) (or sometimes another character). One possible file extension `csv`. Can be read directly by spreadsheet programs.

Decode (noun)

Given an IrSequence or an IrSignal, a *decode* is a protocol, together with parameter values, that are compatible with the given IrSequence/IrSignal, i.e. could have generated the original signal. Note that the determination is governed by numerical uncertainties, so that small deviations from the perfect signal are accepted. Furthermore, one signal/sequence may have none, one, or more valid decodes.

DecodeIR

Library for the decoding of IrSequences. Originally written by John Fine, extended by others; used by many widely spread programs as a shared library, often using JNI. The current version is 2.44. Binaries for Windows, Linux, and Mac, source code.

Demodulating IR Receiver

An integrated circuit that receives a modulated IR signal and recovers the original signal with the modulation removed. The modulation frequency of the signal must "match" the frequency of the demodulator. Data sheet for a typical product. Not suited for capturing of unknown signals, since it removes the modulation frequency without identifying it.

Device Number

See protocol parameters. Denoted by D in IRP protocols.

Device Type

Class of components, like TV, VCR, Satellite receiver, etc.

Duty Cycle

The percentage of the time the the modulation pulse is on. Typically slightly less than 50%.

Ending sequence

See IrSignal.

Eventghost

"EventGhost is an advanced, easy to use and extensible automation tool for MS Windows. It can use different input devices like infrared or wireless remote controls to trigger macros, that on their part control a computer and its attached hardware." Licensed under GPL2.Home page for Eventghost.

ExchangeIR

Library for IR signal analysis and exchange by Graham Dixon. Licensed under the GPL3 license. For the present project, the interesting parts are the Analyzer, the repeat finder, and functions for the UEI learned format. These parts has been translated to Java by myself: API documentation, source.

Flash

Period of time when the IR light is "on", or flashed with the selected modulation frequency. See IrSequence.

Function Number

See protocol parameters. In IRP protocols, denoted by F. A synonym is OBC.

Gap (or "Pause")

Period of time when the IR light is off. See IrSequence.

Generating, sometimes called rendering

The process of evaluating an IrProtocol for a particular parameter value, rendering an IrSignal. Commonly used rendering programs/engines are the older MakeHex, and the more modern and capable IrpMaster.

Girr (Generic IR Remote)

A general XML-based exchange formats for IR Signals. Really a container format that can contain any of the CCF, raw format, protocol/parameter format, as well as other text formats like Global Caché sendir and UEI learned format. For a full description, see the full documentation.

Global Caché

http://www.globalcache.com/

GPL3 license

The current version of the GNU General Public License. Used by my software projects (with some exceptions), and many so-called open-source software projects. The basic idea is the licensee is allowed to use, enhance etc. the software (also in a commerical product and context), but is not allowed to turn it, or a derived product, into non-free software.

IR (Infrared light)

According to Wikipedia, infrared light are light (electromagnetic radiation) of wavelength between 700nm and 1mm. For control of consumer electronics (CIR), according to Wikipedia, wavelengths around 870 nm and 930-950 nm (latter

preferred), in comparison to IrDA (850-900nm) are used. Almost always generated by an IR LED.

IrDA

IrDA is a method for data exchange between PCs and portable devices. It is no longer to be considered as state-or-the-art, and has been almost completely replaced by Bluetooth and WiFi. Many devices with IrDA hardware is still around, but it is, with few exceptions, unsuitable for consumer IR.

IR LED (light emitting diode)

Semiconductor component capable of sending light with the desired IR wavelength. A typical representative is the Osram SFH 4512.

(IR) Protocol

An algorithm for turning a number of parameters into an IR signal. It defines the necessary parameters and their allowed values. In almost all protocols, the most frequently changing parameter is called "F" (function number). Almost all protocols have a "device number" called "D". Many protocols have a "sub-device" number, called "S". A few protocols have a toggle parameter, in general called "T", and being persistent. A protocol may also have other parameters, with "arbitrary" names.

IrMaster

A program for generating, analyzing, importing, and exporting of infrared signals. Now discontinued, replaced by IrScrutinizer.

IrpMaster

A program and API library for rendering IRP protocols version 2. See its documentation. Comes with a powerful (but slightly hard to use) command line interface. For GUI usage, see IrMaster and IrScrutinizer. Note that the word "IrpMaster" sometimes refers to the command line program, sometimes to the rendering engine contained in IrMaster and IrScrutinizer.

IRP Notation

Compact, slightly cryptical, notation for defining an IrProtocol. Specification.

IrScope

Program that accompanies the IrWidget, also by Kevin Timmerman. Originally a support to the IrWidget, was further developed (in particular through Graham Dixon) to a fairly general and capable IR analyzing program, supporting also DecodeIR and ExchangeIR. The program was a major inspiration source for IrScrutinizer.

IrScrutinizer

IrScrutinizer is a powerful program for capturing, generating, analyzing, importing, and exporting of infrared signals.

IR Sequence

Sequence of time durations, in general in expressed microseconds, together with a modulation frequency. The even numbered entries normally denote times when the IR light is on (modulated), called "flashes", the other denote off-periods "gaps".

IR Signal

Consists of three IR sequences, called

1. *start sequence* (or "intro", or "beginning sequence"), sent exactly once at the beginning of the transmission of the IR signal,
2. *repeat sequence*, sent "while the button is held down", i.e. zero or more times during the transmission of the IR signal (although some protocols may require at least one copy to be transmitted),
3. *ending sequence*, sent exactly once at the end of the transmission of the IR signal, "when the button has been released". Only present in a few protocols.

Any of these can be empty, but not both the intro and the repeat.

IrToy
An interesting "open hardware" project by Dangerous Prototypes, see the product page. Consists of a microprocessor PIC18F2550, a demodulating IR-receiver, a non-demodulating IR-receiver, an IR-LED, and a USB-connector. Thus, is usable both for learning, including frequency measurements, receiving demodulated R-signals, and sending IR signals. Supported by IrScrutinizer.

IrTrans
A series of IR products from the firm with the same name. IrScrutinizer and IrMaster supports the Ethernet models (preferrably with the "IR data base"), for sending only.

IrWidget
A very interesting "open hardware" project by Kevin Timmerman. Project page. That page presents many different versions, but the most spread version (until recently made commercially available by Tommy Tyler) consists of a micro processor PIC12F629, a non-demodulating sensor, and a USB serial FTDI interface. Supported by Kevin's IrScope, as well as IrScrutinizer.

Java

Java Native Interface (JNI)
A technique for having a Java program calling a native shared library (DLL in Windows, "Shared object" (`.so`) in other operating systems). See the Wikipedia article.

JP1

JP1 Remote

JSON
A standard for using human readable text to transfer structured data, as an alternative to XML. See the Wikipedia article.

Lirc
An open source project for sending and receiving IR signals from Linux. Official web site. First release in May 1996, current version is 0.9.1a, released in August 2014. IrScrutinizer supports sending through a LIRC server, patched with the CCF patch.

Lircd
The main daemon of the Lirc project. Accepts commands on a Unix domain socket, as well as on a TCP socket, by default 8765. Accepts commands to send IR signals in Remote/Command format only. For a complete list or commands implemented, see the man page of `irsend`, a progam that implements a simple client for the Lircd server daemon.

MakeHex
A predecessor to IrpMaster. Adheres to an earlier version ("Version 1") of the IRP Notation. For the original C++ program by John Fine, neither a GUI nor a command line interface are present; the parameters are given to the program by editing the data base files. A Java translation (by myself) exists, which has a command line interface, available here.
Mode2 (LIRC) format/program
Modulation frequency
During the "on" periods, the IR light is not just constantly on, but "flashed" on and off at a frequency called the modulation frequency, typically between 36kHz to 40kHz, in some cases higher (up to 56kHz), or much higher (455kHz, Bang & Olufsen equipment). This reduces noise sensitivity and power consumption, and also allows higher currents through the IR LED (that thus does not have to be able to survive the high current continuously). Also see Duty cycle.
(Non-demodulating) IR receiver
IR receiver that outputs the received IR signal essentially as received, i.e. without removing a modulation.
Original Button Code
Synonym for function number.
Parametric IR signal
An Ir Signal given as a protocol and a parameter assignment to its parameters. Of course, a renderer may compute the numerical IR Sequences, but these are *considered* secondary, it is *defined* by its protocol and parameters values.
PCF
IR signal format, not to be confused with the pcf file format of the Pronto NG remotes! This is a proprietary and encrypted form of IR signals. As far as I am aware, it is presently not known how to decode this representation.
pcf file format
Like the xcf format, this is a ZIP file containing an XML file with the real payload, and a number of icon files. Unfortunately, the enclosed IR signals are in the PCF format, thus possible to decrypt only by the ProntoEditNG program.
Persistent variable
A *persistent variable* in an IR protocol may, but need not, be given a value before generating. If not, it retains its value from previous invocation, or, for the first invocation, has a default value.
Pronto Classic
Family of advanced touch-screen remote controls. Manufactured by Philips 19xx-19yy. TODO. Consists of the models TS1000, TSU2000, TSU6000, RC5000, RC5000i, RC5200, RC9200, RU890, RU940, RU970, USR5, RAV2K, RAV2KZ1. Configurable/programmable by a GUI program "ProntoEdit", as well as the open-source program Tonto.
ProntoEdit

Windows program for programming the Pronto remotes. Exists in different versions for different Pronto series. From its owner Philips now discontinued, but available for download at RemoteCentral.

Pronto frequency code

The second number in the CCF representation. For *f* in Hertz, this is the four-digit hexadecimal number given as 1000000/(*f**0.241246). It can be conveniently computed by the Time/Frequency Calculator in IrScrutinizer, available under the Tools menu.

Pronto NG (New Generaton)

Later generation of Pronto touch screen programmable remotes. Uses the pcf format as their configurations. Can be read by ProntoEditNG.

Pronto Professional (*.xcf configuration files)

Later generation of Pronto touch screen programmable remotes. Uses the xcf format as their configurations. Consists of the models TSU9800, TSU9600, TSW9500, TSU9400, TSU9300, TSU9200, TSU9500 (Philips) and RC9001 (Marantz).

Protocol Parameters

See IR Protocol.

protocols.ini

properties (of an interactive program)

The part of the program's state saved between sessions for each user; saved to disk.

Raw IR sequence/signal

A raw Ir Sequence is a sequence of (in general) measured on-off durations. It may or may not have one or many decodes, but these are *considered* to be secondary; its is *defined* by its numeric time durations. Often written with signs: a "+" indicates a flash, a "-" indicates a gap.

Receiving IR signals (deployment)

The use case of receiveing an a priori partially known (typically through its protocol, in particular, the modulation frequency) signal, identifying it completely (typically its parameters protocol parameters), and possibly initiating an action. Cf. the other use case capturing.

Repeat finder

Class in ExchangeIr (function in the original version) that from an IR sequence numerically tries to identify the intro-, repeat-, and ending sequence that has generated the original sequence.

Repeat sequence

See IrSignal.

Remote

A collection of commands with unique names.

Remote/Command format

Given appropriate data base entries, the name of a remote (or device) together with the name of a command, can identify a command uniquely.

RemoteMaster

rmdu file

scrutinize
"To examine in detail with careful or critical attention."
sendir (Global Caché) format
Start sequence
See IrSignal.
Sub device Number
See protocol parameters. In IRP protocols, denoted by S.
TVS (tab separated values)
Like CSV but using a tab character (ASCII character 9). File extension `.tsv`, or other.
Toggle
Persistent variable in an IrProtocol, in general alternating between 0 and 1, between different invocations. I.e., if the first invocation has the toggle value 0, all even invocations will have the value 1 of the toggle, all even the value 0, independent of the number of repeat sequences. Also see protocol parameters.
Tonto
An open source re-implementation of ProntoEdit for the Pronto Classic, as well as an Java API library for reading and manipulating CCF files. Author is Stuart Allen. Home site.
Transmitter
Some IR senders have more than one sending channel, called transmitter, allowing for example to control different equipment independently, even if they are using the same commands. These are called transmitters. Note that by definition, every IR sender has at least one transmitter, but only in the case of multiple transmitters, a selection is meaningful.
UEI learned format
wave file format
An IR sequence rendered with halfed modulation frequency, as a sequences of equdistant samples (in general with sample frequency 44.1kHz or 48kHz) considered as an audio signal. It is supposed to be "playbacked" through an audio system connected to a pair of IR LEDs connected in anti-parallel, which will again double the carrier frequency. IrScrutinizer supports both the generation of wave files, as well as its import and analysis.
WinLirc
According to its its web site, it "...is the Windows equivalent of LIRC". It is not a port of LIRC, nor does it share any code with LIRC. Statements on Lirc in these pages are not necessarily true for WinLirc.
xcf configuration file
Configuration file format for the Pronto Professional line of remotes. Consists of a ZIP file containing one configuration file in XML-format, as well as a number of supplementary icon images. The XML file is very easy to understand (for programmers!), and can contain IR signals in different formats, like CCF format (usable!) and PCF format (encrypted, thus not usable).

XML

XML Schema

XML Schema (also called XSD, for "Xml Schema Language") is an XML language for describing the syntax of XML documents. See the Wikipedia article.

XSLT (Extensible Stylesheet Language Transformations)

XSLT is an XML language for transforming XML documents into other XML documents, HTML-pages, or plain text. See the Wikipedia article. The programs here use only XSLT version 1.0.

2.7.2 Appendix. Semantics of the Pronto CCF format.

An IR signal in Pronto CCF form consists of a number of 4-digit hexadecimal numbers. For example:

```
0000 006C 0022 0002 015B 00AD 0016 0041 0016 0016 0016 0016
0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016
0016 0041 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016
0016 0016 0016 0041 0016 0041 0016 0016 0016 0016 0016 0016
0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0041
0016 0041 0016 0041 0016 0041 0016 0041 0016 0041 0016 06FB
015B 0057 0016 0E6C
```

The first number, here 0000, denotes the type of the signal. 0000 denotes a raw IR signal with modulation, while 0100 denotes a non-modulated raw IR signal. There are also a small number of other allowed values, denoting signals in protocol/parameter form, notably 5000 for RC5-protocols, 6000 for RC6-protocols, and 900A for NEC1-protocols.

The second number, here 006C, denotes a frequency code. For the frequency f in Hertz, this is the number 1000000/(f*0.241246) expressed as a four-digit hexadecimal number. In the example, 006C corresponds to 1000000/(0x006c * 0.241246) = 38381 Hertz. (It can be conveniently computed by the Time/Frequency Calculator in IrScrutinizer, available under the Tools menu.)

The third and the forth number denote the number of *pairs* (= twice the number of durations) in the start- and the repeat sequence respectively. In the example, there are 0x0022 = 34 starting pairs, and 2 repeat pairs.

Next the start- and the repeat-sequences follow; their length being given by the third and the forth number, as per above. The numbers therein are all time durations, the ones with odd numbers on-periods, the other ones off-periods. These are all expressed as multiples of the period time; the inverse value of the frequency given through the second number. For this reason, "frequency" must be a sensible number also for the non-modulated case, denoted by the first number being 0100. In the example, the fifth number 0x015B denotes an on-period of 0x015B**periodtime* = 347/f = 347/38381 = 0.009041 seconds.

In particular, all sequences start with an on-period and end with an off-period.

In the Pronto representation, there is no way to express an ending sequence.

2.8 HarcHardware

Warning:
Sending undocumented IR commands to your equipment may damage or even destroy it. By using this program, you agree to take the responsibility for possible damages yourself, and not to hold the author responsible.

Date	Description
2014-02-01	Initial version, for version 0.9.0.

Table 1: Revision history

2.8.1 Introduction

This is a collection of classes for accessing hardware for home automation and remote, in particular infrared, control.

It is intended as a support library for application programs, not as a user friendly program aimed at end users. It is currently used by programs like IrScrutinizer and IrMaster.

There are classes for infrared sending and receiving by communicating with IR sending/ receiving hardware, for serial communication as well as ethernet communication with sockets or HTTP, classes for implementing LAN beacons and beacon listeners, among others.

It is largely designed upon interfaces, so that application programs can access different hardware, e.g. for IR sending, in a uniform way. See the class `Main` for an example.

2.8.2 Copyright and License

The program, as well as this document, is copyright by myself. My copyright does not extend to the embedded "components" ExchangeIR, DecodeIR. ExchangeIR was written by Graham Dixon and published under GPL3 license. Its Analyze-function has been translated to Java by Bengt Martensson. DecodeIR was originally written by John S. Fine, with later contributions from others. It is free software with undetermined license. IrpMaster is using ANTLR3.4 and depends on the run time functions of ANTLR3, which is free software with BSD license.

The program uses JCommander by Cédric Beust to parse the command line arguments. It is free software with Apache 2 license.

Serial communication is handled by the RXTX library, licensed under the LGPL v 2.1 license.

JSON handling is implemented using the "fast and minimal JSON parser for Java" by Ralf Sernberg, licensed under the Eclipse Eclipse Public License Version 1.0.

The program and its documentation are licensed under the GNU General Public License version 3, making everyone free to use, study, improve, etc., under certain conditions.

2.8.3 API documentation

API documentation, generated by javadoc.

2.9 Jirc documentation

> **Note:**
> If looking for a user friendly way of transforming a Lirc file to something else, *do not read this article*. Use the IrScrutinizer instead, in particular the Lirc import feature.

Date	Description
2014-01-12	Initial version.
2014-05-06	Updated for version 0.3.0, improving the handling of repeats, and fixing some bugs.

Table 1: Revision history

2.9.1 Introduction

There is a large body of captured IR signals for different remotes/devices in the Internet, in particular at lirc.sourceforge.net/remotes/. Sometime, it is desirable to transfer this information to more universally known formats for IR signals, such as the CCF format. Although attempts have been made to document the format of the Lirc format, writing a complete and with Lirc coinciding parser/renderer for these files is probably a very difficult undertaking. From this insight, "only Lirc understands Lirc files", the program lirc2xml was created, as a special driver for the `lircd` daemon, writing an XML file instead of accessing hardware, available as a patch to the C sources of Lirc.

For portability reason, a pure Java solution was desired. So I decided to translate the appropriate Lirc files to Java, and the present project was born. The base of the translation was Lirc version 0.9.0.

The initial version, 0.2.0, did not implement the handling of repeats. The current version 0.3.0 however implements repetition and the handling of multiple command numbers, and fixes several small problems, e.g. the handling of multiple remotes with the same name.

2.9.1.1 Copyright

The original work is copyrighted under the GNU General Public License, version 2. Being a derived work, this goes for the present work too.

The program uses JCommander by Cédric Beust to parse the command line arguments. It is free software with Apache 2 license.

2.9.1.2 Program overview

The main goal was, with a moderate effort, to create a (pure) Java version of Lirc2XML, not to make a faithful translation, nor to create maintainable Java code of the highest quality.

The program basically consists of a parser of the configuration files (class `ConfigFile`), some helper classes, and the class `Transmit`. The parser has been written from scratch using ANTLR version 3.4. The classes `Hardware`, `IrNCode`, `IrCodeNode`, and `IrRemote` are essentially translations of Lirc's C structs with similar names. The class `Transmit`, essentially a translation of the C file `translate.c`, carries out the rendering of the IR signals. The main routine resides in the class `Lirc2Xml`, which is written from scratch. The output file is generated in the general Girr format, using its supporting Java library.

All references to serial commands have been deleted; it does not belong to the Lirc domain anyhow (IMHO).

2.9.2 Installation

The program is distributed in source format only. For full functionality, also the DecodeIR library has to be installed. For this, giving the directory path to the Java VM using the `-Djava.library.path` option may be necessary. The supplied wrapper `lirc2xml.sh` can be used as an example for this.

2.9.3 Usage

2.9.3.1 Command line arguments

```
Usage: Lirc2Xml [options] [configfile]
  Options:
    -c, --ccf
       Generate the CCF ("Hex", "Pronto") form of the signals
       Default: false
    -d, --debug
       Debug
       Default: 0
    -e, --encoding
       Character encoding of the generated XML file
       Default: UTF-8
    -f, --fatraw
       Use the fat format for raw signals
       Default: false
    -h, --help, -?
       Display help message
       Default: false
    -o, --outfile
       Output filename
    -p, --parameters
       Generate the protocol name and parameters (if possible) for the signals
       Default: false
    -R, --raw
       Generate the raw form of the signals
```

```
      Default: false
    -r, --remote
      Name of the remote to include in the export (will export all if left
      empty)
    -s, --schemalocation
      Create schema location attribute
      Default: false
    -v, --version
      Display version information
      Default: false
    -x, --xslt
      Link to XSLT stylesheet
```

The program has a traditional "Unix" usage. If no file argument is given, standard input is read. As argument, also a directory can be given, in which case an entire file hierarchy is traversed for Lirc files, handling unparsable files gracefully. Also URLs, e.g. http://lirc.sourceforge.net/remotes/yamaha/RX-V995 can be given as arguments.

On my computer, a local copy of the complete Lirc remote collection (slightly over 3000 files) is transformed in 51 seconds, allocating around 900MB of memory.

2.9.3.2 API documentation

Javadoc available here.

2.9.3.3 Restrictions

The parser presently requires the command numbers following the command names to be hexadecimal only. Command names and remote names containing spaces are rejected. Commands named "end" are presently not accepted. Names are assumed to consist of 8-bit characters between "!" and "~", or between hexadecimal 0xA1 to 0xFF (e.g. using the ASCII or the ISO-8859-1 character sets). The input files are being read asuming the encoding ISO-8859-1.

2.9.4 Download

- Jirc-src-0.3.0.zip. Also needed is
- Girr-src-1.0.0.zip, and
- IrpMaster-src-1.0.1.zip.

2.10 Improved LIRC driver for the Raspberry Pi

Date	Description
2014-01-12	Initial version.

Table 1: Revision history

2.10.1 Introduction

Using the Raspberry Pi, with its flexible GPIO pins, for LIRC IR control and reception is a natural wish. Sending IR diodes and IR receivers can easily be attached to the GPIO pins. Aron Szabo wrote a LIRC driver for the Raspberry, as a development of the LIRC serial driver. This article describes a further development of Aron's driver.

The present article assumes fundamental knowledge about LIRC and the Raspberry Pi. It is not a tutorial.

2.10.1.1 Copyright

The original work is copyrighted under the GNU General Public License, version 2 "or later". Being a derived work, this goes for my version too.

2.10.2 Features

Next the features introduced will be described.

2.10.2.1 Multiple transmitter support

The RPi has at least 17 usable GPIO pins, so why should a LIRC driver only support using one? Furthermore, the LIRC driver API has `LIRC_SET_TRANSMITTER_MASK` ioctl. For the user, the `irsend` command the directive `SET-TRANSMITTERS` (taking as argument a list of numbers), so also LIRC can handle several transmitters (although very few drivers have implemented it). I, somewhat randomly, selected to support up to 8 transmitters. Should more be needed, only the compilation constant `LIRC_RPI_MAX_TRANSMITTER`. As a consequence, the module parameter `tx_mask`, which states the initial state of the transmitters selected, has been introduced.

I could not find any recommendation on the numbering of the transmitters in the LIRC documentation. Since the only driver implementing multiple transmitters (CommandIR) numbered the transmitters starting from 1, I have decided to do likewise.

2.10.2.2 Support for "modulation frequency" 0

Almost all modern IR protocols use a modulation frequency, however, some do not (Revox, Barco, Archer). Possibly more interesting, sending RF devices such as the RX433 are not designed to be fed with modulated signals. Many LIRC drivers, like Aron's, have a boolean parameter `softcarrier`, that govern the generation of the modulation. If turned off, no modulation will be generated. However, being a module parameter, it cannot be changed in a running module, and it can also not have a different value for different transmitters.

The modification presented here simply allows the frequency parameter in the ioctr call LIRC_SET_SEND_CARRIER to have the value 0, in which case no carrier will be generated.

To my knowledge, other LIRC drivers no not support non-modulated IR signals in the sense of frequency = 0.

2.10.2.3 Inversion of outputs

The module parameter `invert`, which inverts the on/off-status of the outputs (for all transmitters) is new.

2.10.3 Installation

This module is no different from other LIRC modules when it comes to compiling and installation. There are a number of LIRC pages on the Internet.

Just copying the compiled binary module `lirc_rpi.ko` to the target system may work.

2.10.4 Module arguments

The module arguments are described in this section. These are parameters that are passed to the module on startup.

`gpio_out_pins` (array of integers)
GPIO output/transmitter pins of the BCM processor as array. The first is called transmitter #1 (not 0). Valid pin numbers are: 0, 1, 4, 8, 7, 9, 10, 11, 14, 15, 17, 18, 21, 22, 23, 24, 25. Default is none.

`gpio_in_pin` (integer)
GPIO input pin number of the BCM processor. Valid pin numbers are: 0, 1, 4, 8, 7, 9, 10, 11, 14, 15, 17, 18, 21, 22, 23, 24, 25. Default is none.

`sense` (bool)
Override auto detection of IR receiver circuit (0 = active high, 1 = active low).

`softcarrier` (bool)
Software carrier (0 = off, 1 = on, default on)

`invert` (bool)
Invert output (0 = off, 1 = on, default off)

`tx_mask` (integer)
Transmitter mask (default: 0x01)

`debug` (bool)
Enable debugging messages.

The name `gpio_out_pins` is incompatible with Aron's version, using the name `gpio_out_pin`. This also goes for the default values: Aron uses 18 and 17 for `gpio_in_pin` and `gpio_out_pin`, my version has both undefined as default. To minimize confusion, I have decided just to publish one version of the driver.

2.10.5 Further work

I would be very interested in getting the driver to work with a non-demodulation sensor, like the QSE159 (see this article), to arrive at an estimate of the modulation frequency.

Except for this, I do not plan to continue the development, or to "maintain" the driver.

2.10.6 Downloads

- lirc_rpi.c. The source.
- lirc_rpi.ko. Compiled kernel module. Sometimes it suffices just to copy this to the RPi.

2.10.7 Resources

- Aron's site.
- Sort-of announcement on LIRC mailing list.
- Diskussion thread on the driver. This driver is discussed on page 6.
- Another guide on LIRC on the RPi.

2.11 Raspberry Pi daughter board for IR and RF transmission and reception

Date	Description
2014-01-11	Initial version.

Table 1: Revision history

2.11.1 Description

This article shows a daughter board for fitting to the GPIO Pins of a Raspberry Pi. It is mainly intended for use with LIRC. It necessitated the development of an an improved LIRC driver.

It is built upon a "TriPad" board. It contains of a 2 x 13 board connector and a number of standard components. There is an IR transmitting LED (SFH415) with an NPN driver transistor (BC547) driver transistor. It is designed to be able to withstand 100mA of *continuous* current through the IR LED. There is also an IR-demodulator TSOP4838. For the LIRC world uncommon is the non-demodulating IR sensor QSE159, which makes it possible to determine the modulation frequency of a captured signal. This works with 5V supply voltage, that could cause a problem for the 3.3V operating GPIO pins. Selecting an open collector model like QSE159 solves this problem however. Unfortunately, at the time of this writing, supporting drivers have not been written yet.

Also contained are an RF 433MHz sender (TX433) and Receiver (RX433). The wire coils are antennas for these. Since the photos were made, an 100nF capacitator has been added between power supply and ground of the RF receiver, to reduce disturbances. From the LIRC drivers, the receiver is addressed exactly like a demodulating IR-receiver, while

the transmitter should not be feed with a modulated signal, thus posing another software problem for LIRC. This will be discussed further here.

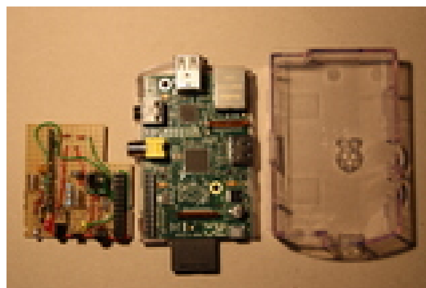
There are also a number of "Debug-LEDs", one of them being a "dummy-IR", to demonstrate the multi-transmitter facility of the improved LIRC driver.

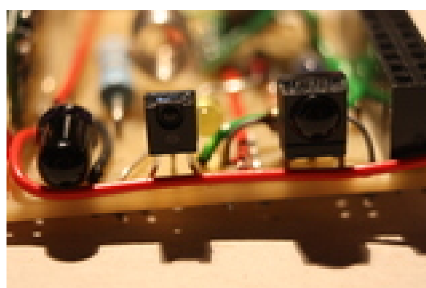
Hooking up these components to the GPIO pins of the Raspberry has been described in many places on the Internet. To be mentioned is in particular Aron Szabo's page. Therefore, we do not describe the circuit in detail.

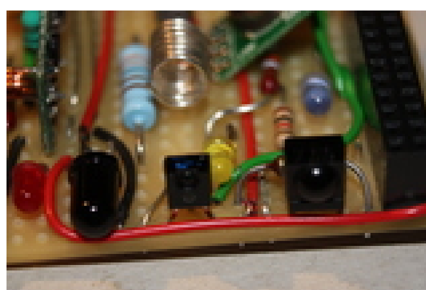
As can be seen on the pictures, I use a case from www.oneninedesign.co.uk. The daughter board fits quite nicely therein. Since it is transparent, it does not interfere with the IR functionality. At least I think/hope that it is transparent also for infrared wavelengths...

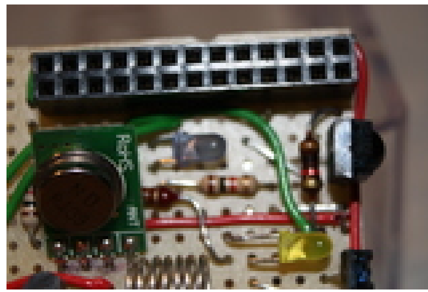
Using the hardware herein to send RF commands for controlling RF switches will be discussed in a future article.

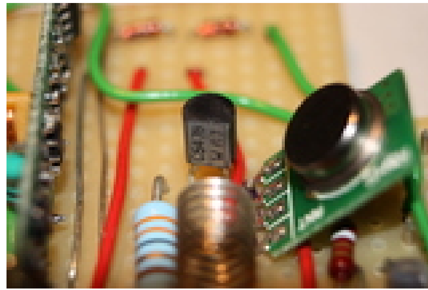
2.11.2 Photos

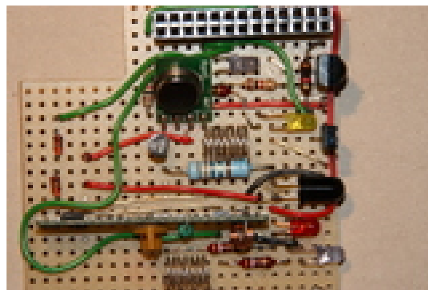


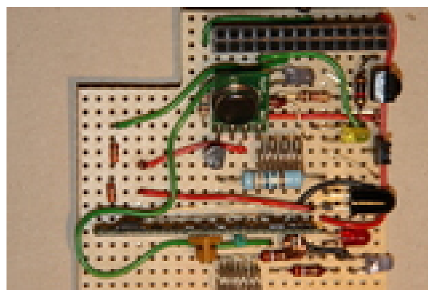


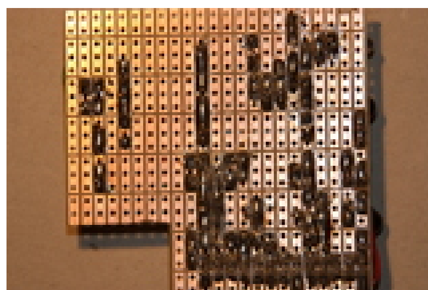


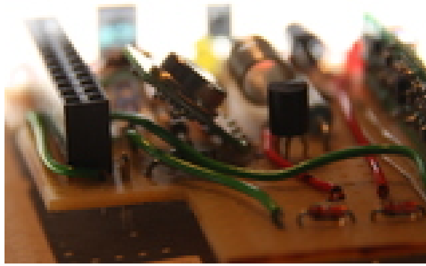


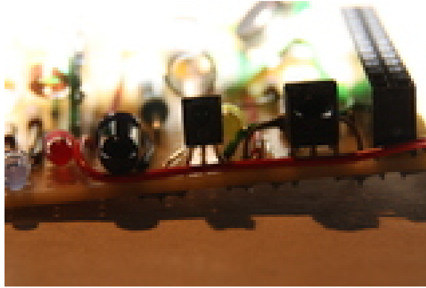


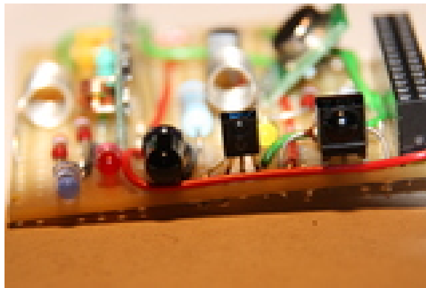


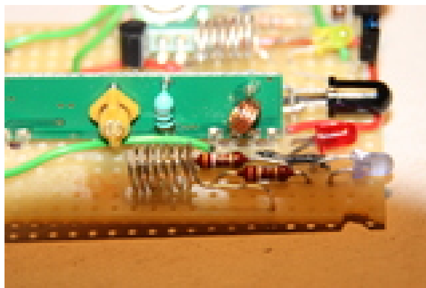


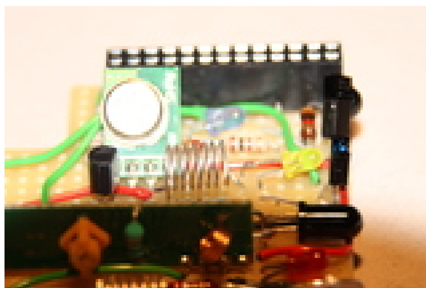












2.12 Enabling LIRC to send CCF signals not residing on the server

Date	Description
2012-04-30	Initial version.
2015-09-10	Updated for the current version.

Table 1: Revision history

2.12.1 Introduction

Some networked devices used for sending IR codes do not store the IR signals locally, but have them sent on every invocation. This goes for e.g. the GlobalCaché IR senders. Other devices, like the LIRC server (`lircd`) keep its own data base of known signal. It can (if started with the --listen option) respond to commands of the type "send *command* from the remote *remote number* of times. It can only send those commands in its data base, typically in the file `/etc/lirc/lircd.conf` (previously `/etc/lircd.conf`). Some devices, like the IRTrans Ethernet with IR Database can both accept commands of the type remote/command and IR signals in raw form.

It is hard to say that one method is universally better than the other. "Keeping the data base on the server, not on the clients" certainly sounds like a good idea in general, however, "the server" is these days not necessarily a big, "fat" machine, and the clients are not necessarily small and dumb. A *server* is by definition something that offers services to clients, but may be small and weak in terms of resources. Furthermore, there may be several such, possibly running on comparatively weak and primitive embedded devices, with limited memory, allowing only a "small" number of flash memory updates, etc. When updating, the database needs to be distributed, possibly a protocol like ssh needs to be implemented for this, etc. Furthermore, as should be clear from my program IrScrutinizer, sometimes the use case is to experimentally try out unknown IR signals. To generate a database for this test, download it to the "server" just in order to perform the tests does not seems like a good idea.

I have submitted a previous version of the patch to the then-maintainer Christoph Bartelmus, who was vehemently against it, meaning that it is cleaner if data bases reside on the server. The issue has been discussed on the LIRC mailing list also when another person submitted a similar patch.

The here presented patch enhances the LIRC server `lircd` to be able to accept commands in the listening socket to send an arbitrary IR signal in Pronto CCF format. Mostly for illustration, the fix also patches the simple client program `irsend` to be able to send these signals.

Implementing the same functionality on WinLirc is non-trivial, since it has completely different code.

The idea of patching LIRC to allow this functionality was originally suggested to me in 2006 by Bitmonster, the creator of the original version of Eventghost.

2.12.2 Installation

The following instruction is intended for users with some experience with installation programs under Linux.

The current version is found as a fork in the Lirc sources on Sourceforge. Clone that fork using a command like `git clone ssh://bengtmartensson@git.code.sf.net/u/bengtmartensson/lirc u-bengtmartensson-lirc`. Then build using the normal instructions. (Note that these have changed somewhat the last few years.)

2.12.3 Extensions to the irsend command

The `irsend` command is contained in the LIRC distribution and (in its unpatched form) is documented here. This patch enhances it with the two *DIRECTIVES* `SEND_CCF_ONCE` and `SEND_CCF_START`. Example:

```
irsend SEND_CCF_ONCE 0000 006C 0022 0002 015B 00AD 0016 0016 0016 0041 0016 0016 0016
 0041 0016 0041 0016 0041 0016 0041 0016 0016 0016 0041 0016 0016 0016 0041 0016 0016
```

```
 0016 0016 0016 0016 0016 0016 0016 0041 0016 0016 0016 0041 0016 0016 0016 0041 0016
 0041 0016 0016 0016 0016 0016 0016 0016 0041 0016 0016 0016 0041 0016 0016 0016 0016
 0016 0041 0016 0041 0016 0041 0016 0699 015B 0057 0016 0EA3
```

sends a pre-rendered NEC1, Device=122, Command=29-command to the LIRC connected device. This turns out to be the power-on command for Yamaha receivers like the RX-V 1400. Another example, assuming that a suitable Yamaha receiver is connected, the command

```
irsend SEND_CCF_START 0000 006C 0022 0002 015B 00AD 0016 0016 0016 0041 0016 0016 0016
 0041 0016 0041 0016 0041 0016 0041 0016 0016 0016 0041 0016 0016 0016 0041 0016 0016
 0016 0016 0016 0016 0016 0016 0016 0041 0016 0016 0016 0041 0016 0016 0016 0041 0016
 0041 0016 0016 0016 0016 0016 0016 0016 0041 0016 0016 0016 0041 0016 0016 0016 0016
 0016 0041 0016 0041 0016 0041 0016 0699 015B 0057 0016 0EA3
```

sends the Volume+-command (NEC1 Device=122, Command=26) "forever" (until it hits `--repeat-max`). Warning: this may have "interesting" side effects on your loudspeakers, your amplifier, your ears, or your neighbors... :-). Do not try this at home...

2.12.4 Known bugs/limitations

The handling of repetitions is not correct. This is due to a limitation in LIRC, that cannot represent a complex intro signal *and* a complex repeat signal.

It would have been a good idea if the patch also modified the LIRC version string, e.g. by appending "`_ccfpatch`".

2.12.5 Downloads

- The old version of the patch for Lirc 0.9.0. *Of historical interest only. Do not use with the current version.*

3 Old programs and docs

3.1 Discontinued projects and old articles

Date	Description
2014-02-02	Initial version.

Table 1: Revision history

3.1.1 Content

In the near future, it is planned to move IrMaster to this section, since it is no more developed, being superseded by IrScrutinizer.

- HarcToolbox, the old main project, no longer developed. Note that the name "HarcToolbox" denotes both this project, and the web site. Sorry for the confusion.

- Transforming XML Export from Ir(p)Master, a tutorial article on generating "interesting stuff" (here, C code) from the XML export of IrpMaster and IrMaster. It is obsolete, since IrScrutinizer uses a different format (Girr) that is not quite compatible with the very simplistic XML format of IrMaster and IrpMaster. Actually, the XML export of these programs became obsolete before I had the time to document them...
- Lirc2xml, a program for extracting IR codes from LIRC files. This is a patch to Lirc 0.9.0, and produces a command line C program. It is superseded by Jirc, written entirely in pure Java, thus attractive to integrate into other programs, like IrScrutinizer, offering a nice GUI.

Downloads.

3.2 IrMaster documentation

> Note:
> Development of this program has been discontinued. It has been superseeded by IrScrutinizer, offering much more functionality.

> Warning:
> Sending undocumented IR commands to your equipment may damage or even destroy it. By using this program, you agree to take the responsibility for possible damages yourself, and not to hold the author responsible.

3.2.1 Revision history

Date	Description
2011-10-23	Initial version.
2012-04-14	Many minor fixes and updates for the upcoming 0.1.2. (version not published)
2012-04-24	Converted to the document format of Apache Forrest. The program documentation is now generated from that file.
2012-06-06	Updated for upcoming version 0.2.0. Many minor impovements. Plotting and Audio new.
2012-08-19	(Not) updated for upcoming version 0.3.0.
2012-11-18	Updated for upcoming version 0.3.1.
2014-01-27	Updated for upcoming version 1.0.0.

3.2.2 Introduction

This is what the program can do: From a data base of known IR signals "recepies" (known as the IRP-notation, essentially corresponding to the collected know-how of the community in machine readable form), IR signals corresponding to certain parameter values can be computed. Export files in different formats can be generated, for usage of other programs. For this, two alternative renders (my own IrpMaster as well as the older Makehex) are available. By using the clipboard, IR signals in Pronto format (for example from Internet articles) can be directly sent to the analyzers AnalyzeIR and DecodeIR. An entered or computed signal can be sent proper hardware, or plotted. For investigating possible non-documented IR signals of owned equipment, a "war dialer" can send whole parameter regions of IR signals. For the latter possibilities, hardware support in the form of GlobalCaché, IRTrans, a LIRC server, or an IR-audio setup, is required. A simple calculator intended for simple computations on IR signal timings is provided.

This program is not a "program with a GUI", nor is it a GUI for a particular program. Rather, it is a "Meta program", offering a GUI for a number of IR related programs, presently IrpMaster (advanced IR rendering program by myself), Makehex (older IR rendering program), DecodeIR (tries to identify an IR signal), and AnalyzeIR (which is my name of the Analyze-Function of the ExchangeIR-Library), and the PtPlot library used for plotting of IR signals. Future extensions to other, possibly not yet written, programs are possible.

Note that I have written two different programs with quite similar names: IrMaster, the present one, a GUI, and Ir**p**Master, a library and a command line program, an IR signal render, but without a GUI. Please do not confuse.

In the sequel, the word "the program" will denote either the "shell" IrMaster, or the GUI together with its "clients" IrpMaster, Makehex, AnalyzeIR, and DecodeIR, as is hopefully clear from the context.

For someone with knowledge in the problem domain of IR signals and their parameterization, this program is believed to be very simple to use. This knowledge is assumed from the reader. Other can acquire that knowledge either from the JP1 Wiki or, e.g., this link.

Note that the screen shots are included as illustrations only; they may not depict the current program completely accurately. They come from different versions of the program, using different platforms (Linux and Windows), and using different "look and feels".

The present document is written more for completeness than for easy accessability. Possibly, in the future, there will be a user's manual as well as a reference manual.

Release notes for the current version.

3.2.2.1 Copyright and License

The program, as well as this document, is copyright by myself. My copyright does not extend to the embedded "components" Analyze, Makehex, DecodeIR, and PtPlot. Makehex was written by John S. Fine (see LICENSE_makehex.txt), and has been translated to Java by Bengt Martensson. ExchangeIR was written by Graham Dixon and published under GPL3 license. Its Analyze-function has been translated to Java by Bengt Martensson. DecodeIR was originally written by John S. Fine, with later contributions from others. It is free software with undetermined license. PtPlot is a part of the Ptolemy Project at the EECS department at UC Berkeley, licensed under the UC Berkeley copyright. IrpMaster is using ANTLR3.4 and depends on the run time functions of ANTLR3, which is free software with BSD license.

The "database file" IrpProtocols.ini is derived from DecodeIR.html, thus I do not claim copyright.

The `main()` method of the Wave class uses JCommander by Cédric Beust to parse the command line argument. (Likely, I will use it much more in the future.) It is free software with Apache 2 license.

Icons by Everaldo Coelho from the Crystal project are used; these are released under the LGPL license.

The Windows installer was built with Inno Setup, which is free software by Jordan Russel. To modify the user's path in Windows, the Inno extension modpath by Jared Breland, distributed under the GNU Lesser General Public License (LGPL), version 3.

The program and its documentation are licensed under the GNU General Public License version 3, making everyone free to use, study, improve, etc., under certain conditions.

3.2.2.2 Privacy note

Some functions (Help -> Project Homepage, Help -> IRP Notation Spec, Help -> Protocol Specs, Tools -> Check for updates) access the Internet using standard http calls. This causes the orginating machine's IP-address, time and date, the used browser, and possibly other information to be stored on the called server. If this is a concern for you, please do not use this (non-critical) functionallity (or block your computer's internet access).

3.2.3 Installation

3.2.3.1 General

IrMaster, and all but one of its third-party additions, are written in Java, which means that it should run on every computer with a modern Java installed; Windows, Linux, Macintosh, etc. Java 1.6 or later is required. The one exception is DecodeIR, which is written in C++, and invoked as a shared library (`.dll` in Windows, `.so` in Linux, etc).

If DecodeIR is not available on your platform it is not a major problem, as IrMaster will work fine without it; just the DecodeIR-related functions will be unavailable.

There is unfortunately no good `make install` or such in the source distribution, so also source code distribution users are recommended to install the binary distribution. Also, all necessary third-party components are included in the binary distribution.

Both under Windows as well as under other operating systems, IrMaster (and IrpMaster) behave civilized, in that they do not write in the installation directory after the initial installation. In both cases (in contrast to the source distribution), the distribution contains everything needed including third party libraries like DecodeIR, AnalyzeIR, MakeHex (Java version) and its irp-files.

Under Windows, the properties are stored in `%LOCALAPPDATA%\IrMaster\IrMaster.properties.xml` using Windows Vista and later (on my Windows 7 system, this is `%HOME%\AppData\Local\IrMaster\IrMaster.properties.xml`), otherwise in `%APPDATA%\IrMaster\IrMaster.properties.xml`. Using other operating systems, it is stored under `$HOME/.irmaster.properties.xml`. It is not deleted by uninstall. (If weird problems appear when updating, try deleting this file.)

3.2.3.2 Windows

Download the Window setup file, save, and double click. Accept the license. Select any installation directory you like; suggested is `C:\Program Files\IrMaster`. Unless reason to do so, create the start menu folder, the desktop icon, and allow the program to add the application directory to your path (for IrpMaster as command line program). Administrator rights are probably needed, at least if you are installing in a directory like `Program Files`. (The *should* not be needed otherwise, but Windows Vista and later are always good for a surprise...) IrMaster can now be started from `Start -> IrMaster -> IrMaster`, or from the desktop icon.

To uninstall, select the uninstall option from the Start menu. Very pedantic people may like to delete the properties file too, see above.

3.2.3.3 MacOSX

Download and double click the binary distribution. Unpack it to a directory of your choice, e.g. on the desktop. Just double clicking the file IrMaster.jar should now start the program. Otherwise, try the "Other systems" instructions and adapt the wrapper irmaster.sh. This also includes invoking as command line program "IrpMaster".

3.2.3.4 Other systems (Linux etc)

For some reason, double clicking an executable jar file in my Gnome installation does not start the program, but starts a browser for the jar file (which is really a form of Zip-Archieve). Instead:

Create an installation directory (suggestion; `/usr/local/irmaster`), and unpack [the current binary distribution] therein. Examine the wrapper `irmaster.sh`, and, if desired, make desired changes to it with your favorite text editor. Then make two symbolic links from a directory in the path (suggestion; `/usr/local/bin` to the newly installed `irmaster.sh`, using the names `irmaster` and `irpmaster`. Example (using the suggested directories)

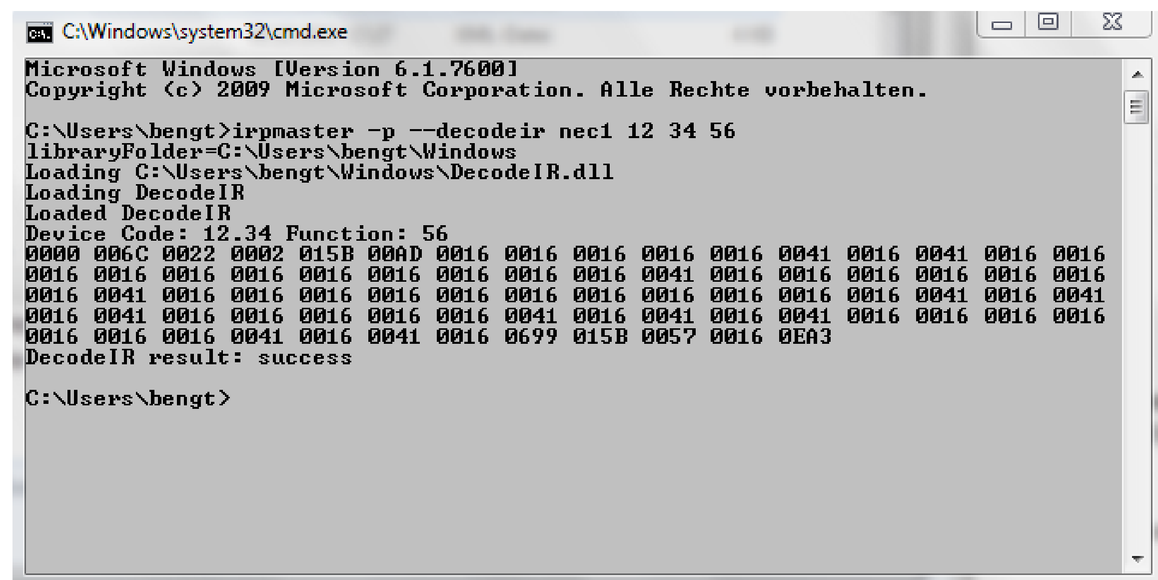
```
cd /usr/local/bin
ln -s ../irmaster/irmaster.sh irmaster
ln -s ../irmaster/irmaster.sh irpmaster
```

(`su` (or `sudo`) may be necessary to install in the desired locations.)

To uninstall, just delete the files. Very pedantic people may like to delete the properties file too, see above.

3.2.3.5 Wrapper for IRPMaster

Both under Windows and Unix-like systems, a wrapper for the command line program IrpMaster is installed. The user can simply open a command window (called anything like "xterm", "Terminal", "Shell", "DOS-Box", "cmd",...) and in that command window can call the program IrpMaster by simply typing `irpmaster`, followed by its arguments. See the screen shot below, that shows the generation of the Pronto form of a NEC1 signal for device 12, subdevice 34, command number 56, together with subsequent invocation of DecodeIR on the computed result. (The output on non-windows system is entirely similar.)



3.2.4 Usage

As stated previously, for anyone familiar with the problem domain, this program is believed to be easy to use. Almost all user interface elements have toolhelp texts. In what follows, we will not attempt to explain every detail of the user interface, but rather concentrate on the concepts. Furthermore, it is possible that new elements and functionality has been implemented since the documentation was written.

This program does not disturb the user with a number of annoying, often modal, pop ups, but directs errors, warnings, and status outputs to the *console window*, taking up the lower third of the main window. Starting with version 0.2.0, this window is resizeable. There is a context menu for the console, accessible by pressing the right mouse button in it.

In the upper row, there are four pull-down menus, named File, Edit, Options, and Help. Their usage is believed to be self explanatory, with the exception of the entries in the Options menu. The latter mimics the Options subpane, and are explained later.

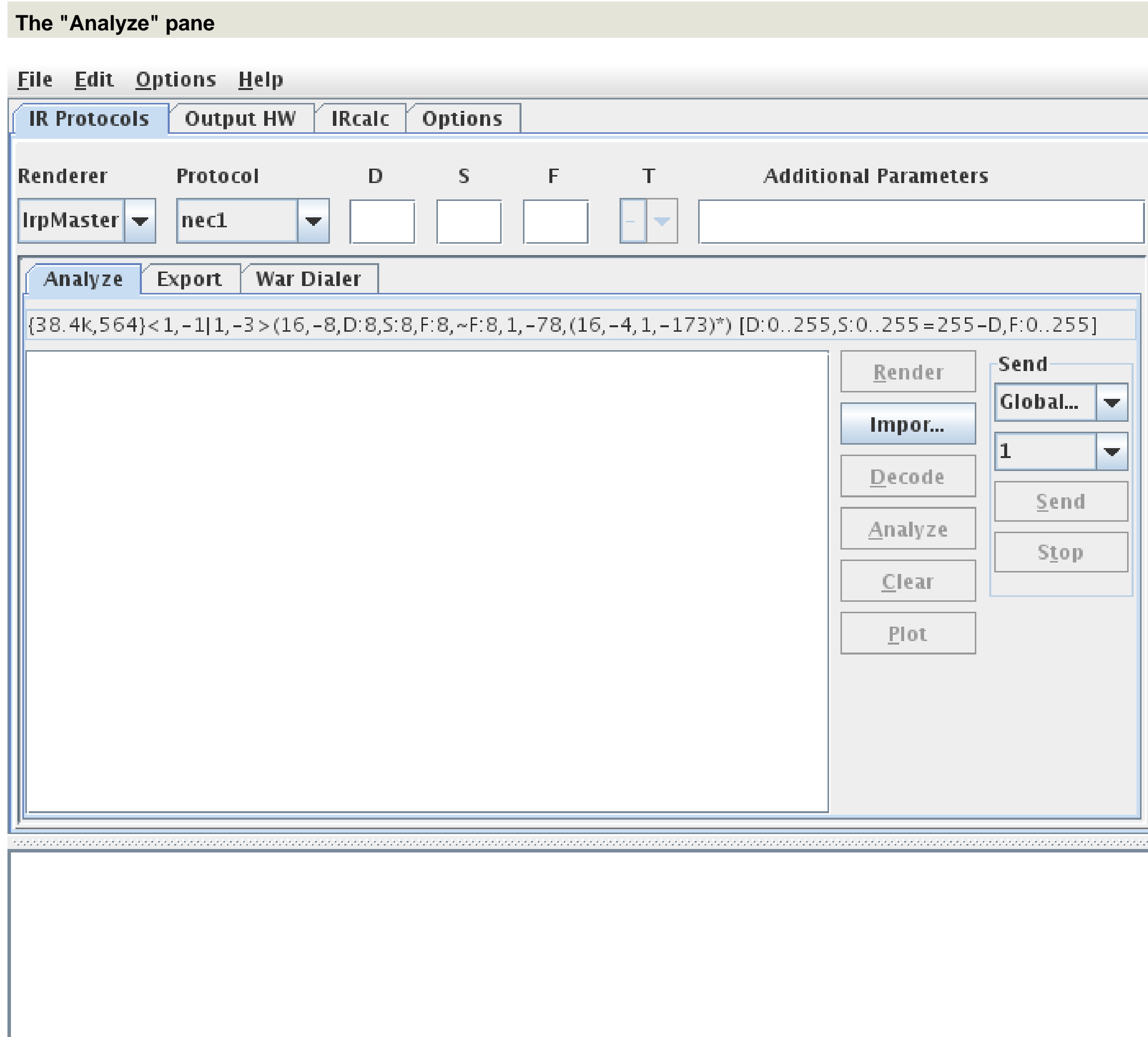
The main window is composed of presently two sub panes denoted by "IR Protocols" and "Hardware" respectively. These will be discussed now.

3.2.4.1 The "IR Protocols" pane

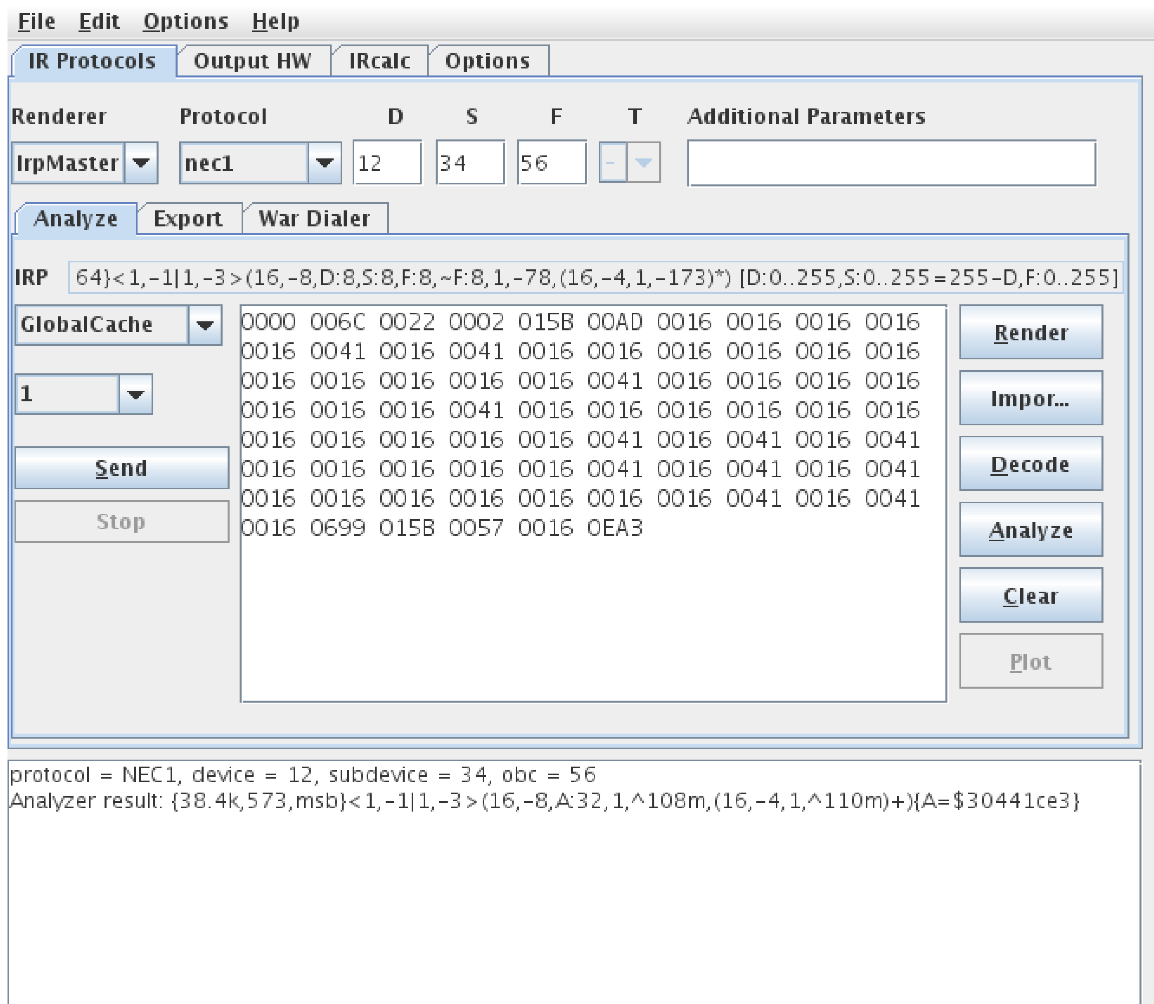
In the upper third of this pane, a render program (IrpMaster or Makehex) can be selected, together with the IR protocol identified by name, and the parameters D ("device", in almost all protocols), S ("sub-device", not in all protocols), F ("function", also called command number or obc, present in almost all protocols), as well as "T", toggle (in general 0 or 1, only in a few protocols). These number can be entered as decimal numbers, or, by prepending "0x", as hexadecimal numbers. Note that the supported protocols are different between the different rendering engines. Not all possibilities of IrMaster are available when using the simpler render Makehex.

The lower two thirds of the window is occupied by another pane setup, consisting of the sub-panes "Analyze", "Export", and "War dialer".

The "Analyze" pane

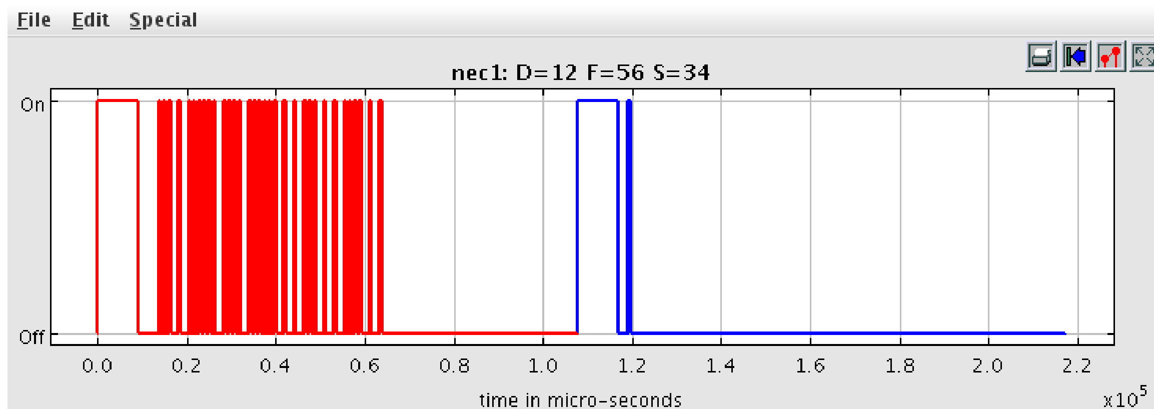


By pressing "Render", the signal is computed, and the middle window is filled with the Pronto representation of it. Pressing "Decode" sends the Pronto representation to DecodeIR. Pressing "Analyze" sends it to AnalyzeIR. In both cases, the programs will send their output to the console window, as can be seen below.



Using context menus, the result can be sent to the clipboard or saved to a file, after invoking a file selector. It is also possible to fill the code window by pasting from the clipboard. Pressing the "Import" button allows to import to import a wave file (see this for a background) or a file in ict-format, for example from the IRScope-Program. The imported IR sequence can be subsequently Decode-d, Plot-ted, and Analyze-d, etc. Note that an ICT file produce by IRScope may contain several IR signals, which cannot be easily separated. They will be imported as one giant signals, with all the gaps and spaces concatenated together. This is a flaw in the IRScope program.

By pressing the "Plot" button, either an IR signal in the CCF window, or a newly rendered signal corresponding to the selected parameters, is shown in a popup plot window, using the PtPlot library.



Using its own controls, the plot can be zoomed (press left button and drag), printed, or exported to encapsulated PostScript. Once created, it cannot be changed (other than zooming, resizing etc), just closed. However, an "unlimited" number of such popup windows are possible.

The right part of the middle window allows for sending the code in the code window to hardware, presently GlobalCaché, IRTrans, or LIRC (requires a patch), or an audio-IR-setup, any number of times the user desires. These run in their own threads, and can be stopped anytime.

In all cases, if the CCF window is non-empty and starts with "0000", DecodeIR/Analyze operates on the actual Pronto code, which may even be manually manipulated by the user. If it start with a "+"-character, it is attempted to interpret it as a "raw" signal, consisting of a number of gaps in pulses, given in microseconds. If it consists of a number of hexadecimal two-digit numbers, it is attempted to interpret the signal as a UEI learned signal.

If the window is empty, new signal is rendered according to the parameters, and subsequently used for sending or plotting. There is thus no need to "render" before plotting or sending.

In rare cases, transforming the signal to the Pronto format may introduce some rounding errors causing DecodeIR to fail to indicate some IR signals it would otherwise identify.

The Export pane

Using the export pane, export files can be generated. These allow e.g. other programs to use the computed results. Using IrpMaster as rendering engine, exports can be generated in text files, XML files, LIRC-format, Lintronic-format, or as Wave-files, the first two both in Pronto format and in "raw" format (timings in microseconds, positive for pulses, negative for gaps). The XML export is intended as a starting point for generate yet other formats, by invoking easily written transformations on it. In this article I demonstrate how to generate C code from it using XSLT transformations. The LIRC-exports are in lirc.conf-format using the raw format. They can be concatenated together and used as the LIRC server data base, typically `/etc/lirc/lircd.conf`. Of courses, they can also be used with WinLirc. For the wave-export, parameters are "inherited" from the Output HW/Audio pane. Options for the wave-export, as well as some of its usage, is explained there. Optionally, for protocols with toggles, both toggle pairs may optionally be included in the export file by selecting the "Generate toggle pairs"-option. Export file names are

either user selected from a file selector, or, if "Automatic file names" has been selected, automatically generated.

The export is performed by pressing the "Export" button. The "..."-marked button allows for manually selecting the export directory. It is recommended to create a new, empty directory for the exports. The just-created export file can be immediately inspected by pressing the "View Export"-button, which will open it in the "standard way" of the used operating system. The "Open" button similarly opens the operating systems standard directory browser (Windows Explorer, Konquistador, Nautilus,...) on the export directory.

The export formats Wave and Lintronic export an IR sequence rather than an IR signal (consisting of an intro sequence, an repetition sequence (to be included 0 or more times), and an (most often empty) ending sequence). Therefore, using the Wave and Lintronic formats, the number of repetition sequences to include can be selected.

The "War Dialer" pane

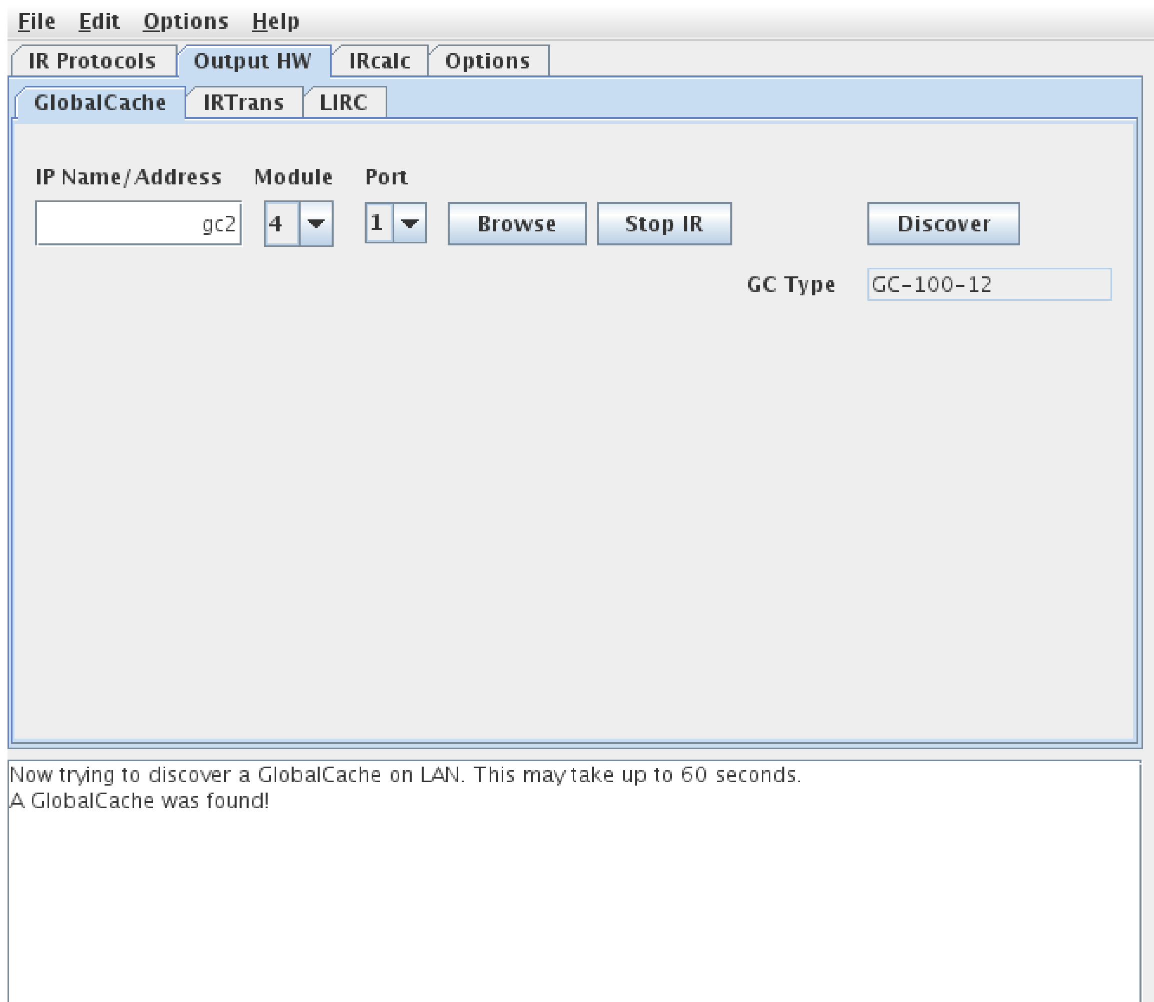
This functionality is intended for the search for undocumented IR commands for customer equipment. It allows for sending a whole interval of IR signals to the equipment, and taking notes when something reacts on the sent signal. The "Start" and "Stop" functions are probably self explaining; the "Pause" button allows for interrupting and later resuming, but is presently not implemented. A note-taking function is planned but presently not implemented: when "Edit" is pressed, a "Document" pops up with current IR signal and time, allowing the user to write a note on that signal, which can later be saved by "Save".

3.2.4.2 The "Hardware" pane

The sub-panes of this pane allows for the selection and configuration of the employed IR hardware.

The "GlobalCache" pane.

This allows for configuring GlobalCaché support. The Discover-button attempts to find the identifying beacon of GlobalCaché modules (only present on recent firmware). If successful, will fill in the IP-Box, its model, the default IR-module and IR-port (see the GlobalCaché API specification for the exact meaning of these terms). In any case, the IP Name/address window, the module and port selection can be changed manually. The Browse-button directs the selected browser to the built-in WWW-server of the module, while the "Stop IR"-Button allows the interruption of ongoing transmission, possibly initiated from another source.



The "LIRC" pane

To be fully usable for IrMaster, the LIRC server has to be extended to be able to cope with CCF signal not residing in the local data base, but sent from a client like IrMaster, thus mimicing the function of e.g. a GlobalCaché. The needed modification ("patch") is in detail described here. However, even without this patch, the configuration page can be used to send the predefined commands (i.e. residing it its data base `lirc.conf`). It can be considered as a GUI version of the `irsend` command.

The LIRC server needs to be started in network listening mode with the `-l` or `--listen` option. Default TCP port is 8765.

After entering IP-Address or name, and port (stay with 8765 unless a reason to do otherwise), press the "Read" button. This will query the LIRC server for its version (to replace the grayed out "<unknown>" of the virgin IrMaster), and its known remotes and their commands. Thus, the "Remote" and "Command" combo boxes should now be selectable. After selecting a remote and one of its command, it can be sent to the LIRC

server by pressing the "Send" button. If (and only if) the LIRC server has the above described patch applied, selecting "LIRC" on the "Analyze" and "War Dialer" panes now works.

Due to LIRC's pecular form of API stop command, the "Stop IR" command presently does not work. See this thread in the LIRC mailing list for a background.

The "IRTrans" pane

The configuration of IRTrans is similar to LIRC, so it will be described more briefly.

Enter IP name or -address and select an IR port (default "intern"). If the Ethernet IRTrans contains an "IR Database" (which is a slightly misleading term for an internal flash memory, that can be filled by the user), its commands can be sent from this pane. By pressing the "Read" button, the known remotes and commands are loaded, and the version of the IRTrans displayed. The selected command can now be sent by the "Send" button. (However, this functionality is otherwise not used by IrMaster.) Selecting

"IRTrans" on the "Analyze" and "War dialer" pane should now work. The IRTrans module is then accessed using the UDP text mode.

The "Audio" Pane

As additional hardware device, IrMaster can generate wave files, that can be used to control IR-LEDs. This technique has been described many times in the internet the last few years, see for example this page within the LIRC project. The hardware consists of a pair of anti-paralell IR-LEDs, preferably in series with a resistor. Theoretically, this corresponds to a full wave rectification of a sine wave. Taking advantage of the fact that the LEDs are conducting only for a the time when the forward voltage exceeds a certain threshold, it is easy to see that this will generate an on/off signal with the double frequency of the original sine wave. (See the first picture in the LIRC article for a picture.) Thus, a IR carrier of 38kHz (which is fairly typical) can be generated through a 19kHz audio signal, which is (as opposed to 38kHz) within the realm of medium quality sound equipment, for example using mobile devices.

IrMaster can generate these audio signals as wave files, which can be exported from the export pane, or sent to the local computers sound card. There are some settings available: Sample frequency (42100, 48000, 96000, 192000Hz), sample size (8 or 16 bits) can be selected. Also "stereo" files can be generated by selecting the number of channels to be 2. The use of this feature is somewhat limited: it just generates another channel in opposite phase to the first one, for hooking up the IR LEDs to the difference signal between the left and the right channel. This will buy you double amplitude (6 dB) at the cost of doubling the file sizes. If the possibility exists, it is better to turn up the volume instead.

Data can be generated in little-endian (default) or big-endian format. This applies only to 16-bit sample sizes.

As an experimental option, the carrier frequency division as described above can be turned off (the "Divide carrier" checkbox). This is only meaningful for sample frequencies of 96kHz and higher, and for "audio equipment" able to reproduce frequencies like 36kHz and above.

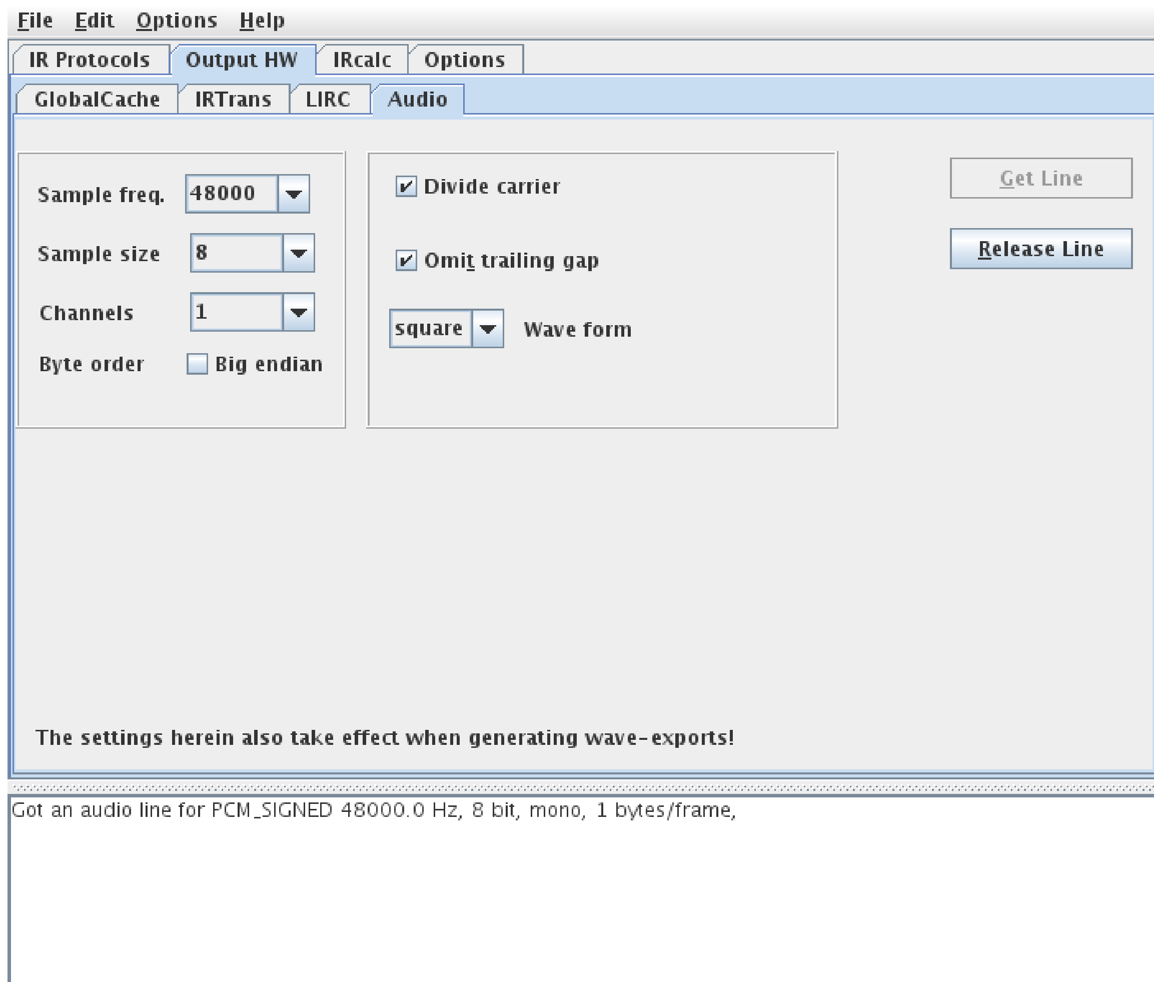
Most of "our" IR sequences ends with a period of silence almost for the half of the total duration. By selecting the "Omit trailing gap"-option, this trailing gap is left out of the generated data -- it is just silence anyhow. This is probably a good choice (almost) always.

Finally, the wave form on the modulation signal can be selected to either sine or square wave. For practical usage, my experiments shown no real performance difference.

Note that when listening to music, higher sample rates, wider sample sizes, and more channels sound better (in general). However, generating "audio" for IR-LEDs is a completely different use case. The recommended settings are: 48000kHz, 8bit, 1 channel, divide carrier, omit trailing gap, and square wave form.

Note that the settings on this pane also take effect when exporting wave files from the export pane.

By pressing "Get Line" a "line" to the audio system on the local computer is allocated. This is actually superflous, since the send-functions make this automatically anyhow. It will possibly be removed in future versions.



3.2.4.3 Command line arguments

Normal usage is just to double click on the jar-file, or possibly on some wrapper invoking that jar file. However, there are some command line arguments that can be useful either if invoking from the command line, or in writing wrappers, or when configuring custom commands in Windows.

```
Usage:
        irmaster [-v|--verbose] [-d|--debug debugcode] [-p|--properties propertyfile]
 [--version|--help]
or
        irmaster IrpMaster <IrpMaster-options-and-arguments>
```

The options `--version` and `--help` work as they are expected to work in the GNU coding standards for command line interfaces: The `-v`/`--verbose` option set the verbose flag, causing commands like sending to IR hardware printing some messages in the console. The debug option `-d`/`--debug` takes an argument, and the result is stuffed into the debug parameter in the program. This is passed to invoked programs that are free

to interpret it any way it likes. For example, here is how IrpMaster interprets the debug variable. This option is likely of interest only to developers, who have to look in the code to see what is sensible.

The second form invokes IrpMaster as a the command line program on the rest of the arguments. It is a convenience feature for just having one user entry point in the distributed jar file.

For automating tasks, or for integrating in build processes or Makefiles or the like, it is probably a better idea to use IrpMaster instead, which has a reasonably complete command line interface.

The program delivers well defined and sensible exit codes.

3.2.5 References

1. IrpMaster. Also a GPL3-project by myself. Much harder to read than the present document :-). See also this discussion thread in the JP1 forum.
2. The Harctoolbox project, also a GPL3-project by myself. It is used for the interface to GlobalCaché and IrTrans, as well as some minor support routines, that I did not feel for duplicating.
3. DecodeIR. This shared library tries to identify protocol name and parameters of an IR signal in raw form. Thus, it is in a sense, it implements the "inverse mapping" of IrpMaster.
4. GlobalCaché, a manufacturer of Ethernet connected IR hardware. Note that I have only tried with the GC-100 series, but the IR sending models of theiTach family are believed to work too. (Feel free to send me one :-).)
5. IRTrans, another manufacturer of Ethernet connected IR-hardware. The "IRTrans Ethernet" module, preferably with "IRDB Option" (internal flash memory), is directly supported by the current software.
6. LIRC, Linux InfraRed Control This project contain drivers for almost everything IR-related. The present project is able to use a modified LIRC-server for transmitting IR signals.

3.3 Using and Transforming XML export

Date	Description
2012-06-06	Initial version.
2013-12-20	Added note.

Table 1: Revision history

> **Note:**
> This document is semi-obsolete! It describes transformations on the XML export of IrpMaster and IrMaster. The XML exports of these programs has been superseeded by the newer

IrScrutinizer and the much more developed Girr format, making a Girr-version of the current document desirable.

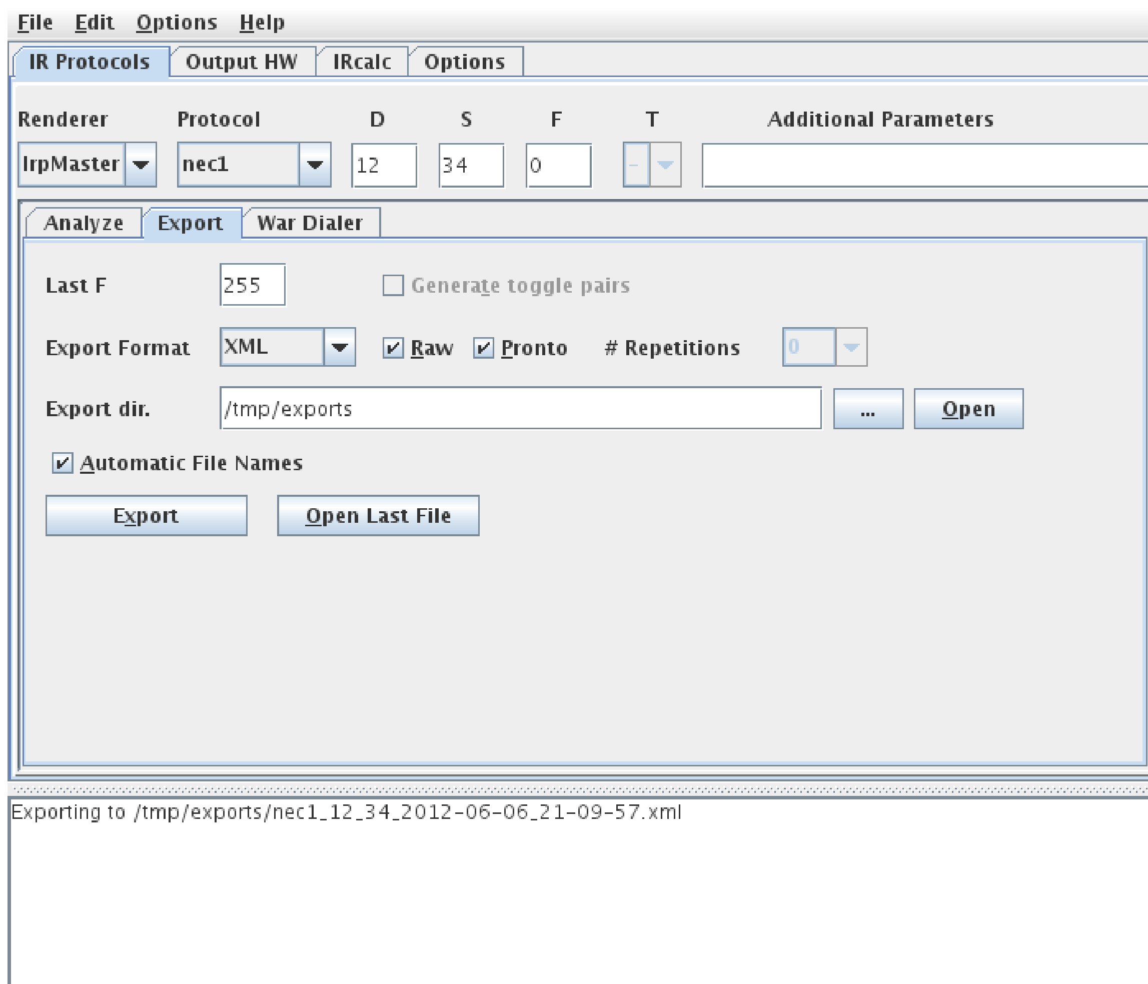
3.3.1 Description

IrpMaster and IrMaster can generate several different export formats. However, the wish may arise to either support yet another format, or to do something special with the generated IR signals. For this purpose, Ir(p)master can generate export in the form of an XML file. XML is an text based format that is readable both by humans and machines. Most importantly, there is a large number of parsers and other tools available, also as free software. As one of many alternatives, we will here look at the rather well known and reasonably easy to learn is XSLT (Extensible Stylesheet Language Transformations). This is a language, itself in XML, for transforming an XML document to either another XML document, to an HTML document, or to a text file.

In this tutorial article, I will demonstrate how to use XSLT to generate something reasonably nontrivial, namely C code, from the XML export. The C file is required to contain both the Pronto signal as C string, as well as integer arrays or durations in micro seconds. The length of the intro sequence, repeat sequence, and, if present, the ending sequence should be contained as well. Finally, we require it to be "neatly" formatted and indented.

The emphasis in this article is to show how this can be easily achieved, without going into details, either on XML, XSLT, C, or anything else. The reader interesting in a similar task can likely just adopt the XSLT file given here. For this, the XSLT-file is put in the public domain.

We want to export all NEC1 codes with device number (D) 12, sub device number (S) 34, and all the possible function (F) numbers ranging from 0 to 255. The XML export is created either with IrMaster, like this



or by using IrpMaster:

```
irpmaster  --xml --raw --outfile ircode.xml nec1 12 34 \*
```

Note that we will need both the raw form (durations in microseconds), as well as the Pronto CCF form, therefore leaving both the "Raw" and the "Pronto" selection checked, or using both `--raw` and `--pronto` options to IrpMaster. This results in the following XML export. On this XML file, the XSLT program (often called *stylesheet*) is invoked, resulting in this generated C code.

A similar, fairly straight-forward, project would be to generate `.rem` configuration files using the CCF format for the IrTrans devices.

3.3.2 irpxml2c.xsl

In this section, the code of the stylesheet irpxml2c is presented. There are a number of tutorials on XSLT available, so the code will not be explained in detail.

The code contains embedded text strings, where whitespace are preserved. We require that the generated C file to looks "neat", i.e., properly indented. Therefore, the indentation of the XSLT-file by necessity looks "ugly", and not neatly indented.

There are a number of different XSLT processors available, most of them free software. How to invoke the XSLT processesor is different from different processors, so it will not be discussed here.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Copying and distribution of this file, with or without modification,
     are permitted in any medium without royalty provided the copyright
     notice and this notice are preserved.  This file is offered as-is,
     without any warranty.
-->

<!--

This file serves as an example on how to use the XML export from Ir(p)Master
to relatively easily generate new output formats. Here we, mostly as an example,
generates C code from the XML file. Although possibly useful as-is, it is intended
as an example, not as a productive solution.

Author: Bengt Martensson

-->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="text" />

    <xsl:template match="/protocol">
        <xsl:text>/* This file was automatically generated by irpxml2c.xsl */

const int frequency = </xsl:text>
<xsl:value-of select="@frequency"/><xsl:text>;
</xsl:text>
<xsl:text>const char *protocol = "</xsl:text><xsl:value-of select="@name"/><xsl:text>";
</xsl:text>
<xsl:text>const int intro_burst_length = </xsl:text><xsl:value-of
 select="count(signal[position()=1]/raw/intro/flash)"/>
<xsl:text>;
const int repeat_burst_length = </xsl:text><xsl:value-of
 select="count(signal[position()=1]/raw/repeat/flash)"/>
<xsl:text>;
const int ending_burst_length = </xsl:text><xsl:value-of
 select="count(signal[position()=1]/raw/ending/flash)"/>
<xsl:text>;

</xsl:text>
        <xsl:apply-templates select="signal" mode="pronto"/>

        <xsl:apply-templates select="signal" mode="raw"/>
    </xsl:template>

    <xsl:template match="signal" mode="pronto">
<xsl:text>/* Signal D = </xsl:text>
        <xsl:value-of select="@D"/>
        <xsl:text>; S = </xsl:text>
        <xsl:value-of select="@S"/>
        <xsl:text>; F = </xsl:text>
        <xsl:value-of select="@F"/>
```

```
        <xsl:text> */
const char *pronto_</xsl:text>
        <xsl:value-of select="/protocol/@name"/>
        <xsl:text>_</xsl:text>
        <xsl:value-of select="@D"/>
        <xsl:text>_</xsl:text>
        <xsl:value-of select="@S"/>
        <xsl:text>_</xsl:text>
        <xsl:value-of select="@F"/>
        <xsl:text> = "</xsl:text>
        <xsl:value-of select="pronto"/>
        <xsl:text>";
</xsl:text>
    </xsl:template>

    <xsl:template match="signal" mode="raw">
       <xsl:text>const int raw_</xsl:text>
        <xsl:value-of select="/protocol/@name"/>
        <xsl:text>_</xsl:text>
        <xsl:value-of select="@D"/>
        <xsl:text>_</xsl:text>
        <xsl:value-of select="@S"/>
        <xsl:text>_</xsl:text>
        <xsl:value-of select="@F"/>
        <xsl:text>[] = { </xsl:text>
        <xsl:apply-templates select="raw"/>
        <xsl:text> };
</xsl:text>
    </xsl:template>

    <xsl:template match="raw">
        <xsl:apply-templates select="*/*"/>
    </xsl:template>

    <xsl:template match="flash"><xsl:value-of select="."/><xsl:text>,</xsl:text></
xsl:template>

    <xsl:template match="raw/*[position()=last()]/gap[position()=last()]"><xsl:text>-</
xsl:text><xsl:value-of select="."/></xsl:template>
    <xsl:template match="gap"><xsl:text>-</xsl:text><xsl:value-of select="."/
><xsl:text>,</xsl:text></xsl:template>

</xsl:stylesheet>
```

3.3.3 Automatic generation using make

It may be desirable to use some sort of automated procedure to generate the C code, for example as part of a project build. The (still) most common build system is Make, so we will show an example utilizing a Makefile. This will, provided that the pathnames are correct, invoke IrpMaster to generate an XML file, and subsequently Apache Xalan as an XSLT-processor to transform it into C code.

```
JAVA=/opt/jdk1.6.0_30/bin/java
XALAN=$(JAVA) -jar /usr/local/apache-forrest-0.9/lib/endorsed/xalan-2.7.1.jar
IRPMASTER=irpmaster

ircode.c: ircode.xml irpxml2c.xsl
 $(XALAN) -IN $< -XSL irpxml2c.xsl -OUT $@
```

```
ircode.xml:
 $(IRPMASTER) --pronto --raw --xml --outfile $@ nec1 12 34 \*
```

3.4 Harctoolbox: Home Automation and Remote Control

FIXME (BM):

In the background of the two programs IrpMaster and IrMaster, the work reported herein is to some extent obsolete, in particular the IR protocol descriptions. In the near future, I intend to restructure this project slightly, possibly splitting it in several subprojects. The current "release", called 0.7.0, is really nothing but a snapshot to support IrMaster. It differs from 0.6.0 in particular since the peculiar IR engine has been replaced by IrpMaster, but also through a number of small fixes.

3.4.1 Revision history

Date	Description
2009-07-18	Initial version, as `harc.html`.
2011-01-10	Restructured. Main part now called `harctoolbox_doc.html`.
2011-04-29	Somewhat updated. The document is presently not quite up-to-date, see the fixme section.

3.4.2 Introduction

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 Harctoolbox at most very roughly. Much is not described at all. Some information in this file may even refer to the previous version, and no more be accurate.

Since I make no assumptions on reaching a large number of users with this release, I think that my creativity is better spent on something else than writing detail documentation for something only experts will use; and these experts probably do not need detailed documentation anyhow...

3.4.3 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.

3.4.4 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.

3.4.4.1 The command names

Harctoolbox 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 Harctoolbox, 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.

3.4.4.2 The protocol files

> **FIXME (BM):**
> This section is obsolete. The IR engine has been replaced by IrpMaster.

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.

3.4.4.3 Device files

A device file is an XML file describing the commands for controlling a device. In Harctoolbox 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.

Example

As an example, consider the Oppo DV-983HDVD player with serial support. This is supported by Harctoolbox 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.

Importers

Since Harctoolbox 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, Harctoolbox 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.

Exporters

Writing an exporter is in principle easier. Harctoolbox 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).

3.4.4.4 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...

3.4.5 Scripting and macros

In the previous version, a simple XML macro facility was implemented. In this version, it has been replaced by the incorporating the scripting engined Jython. This is simply a Java implementation of the very popular and wide spread scripting language Python. This approach has some definite advantages to the previous "solution": It is possible for the user to "script" the Harctoolbox engine, "macros", using a well established and documented syntax and semantics. It also gives access to internal Java objects as Python objects (which of course may introduce some problems too...).

3.4.6 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.

3.4.7 Program usage

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

```
Usage: one of
        harctoolbox --version|--help
        harctoolbox [OPTIONS] [-P] [-g|-r|-l [<portnumber>]]
        harctoolbox [OPTIONS] -P <pythoncommand>
        harctoolbox [OPTIONS] <device_instance> [<command> [<argument(s)>]]
        harctoolbox [OPTIONS] -s <device_instance> <src_device_instance>
where OPTIONS=-A,-V,-M,-C <charset>,-h <filename>,-t ir|rf|www|web_api|tcp|udp|serial|
bluetooth|on_off|ip|special|telnet|sensor,-T 0|1,-# <count>,-v,-d <debugcode>,-
a <aliasfile>, -b <browserpath>, -p <propsfile>, -w <tasksfile>, -z <zone>,-c
 <connectiontype>.
```

Using the `-g` (as well as no argument at all, to allowing for double clicking the `jar`-file) starts Harctoolbox in GUI mode, described in the next section. Invoking Harctoolbox with the `-r`, `-l` *portnumber* starts the readline and port listening mode respectively. Otherwise Harctoolbox will run in non-interactive mode, executing one command or macro, and then exit. If the `-P`-argument is given, commands are interpreted by the Python interpreter, otherwise they are interpreted as *device command [argument(s)...]*.

3.4.7.1 Non-interactive mode

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)

3.4.7.2 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, Harctoolbox'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.

3.4.7.3 Port listen mode

Starting Harctoolbox in port listening mode starts a multithreaded server, listening for commands on the TCP port given as argument, default 9999. The present version also listens to UDP connections using the same port number. 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.

If the `-P`-argument is given, the issued commands are interpreted though the Python interpreter. In this case, the default portnumber is 9998.

3.4.7.4 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.

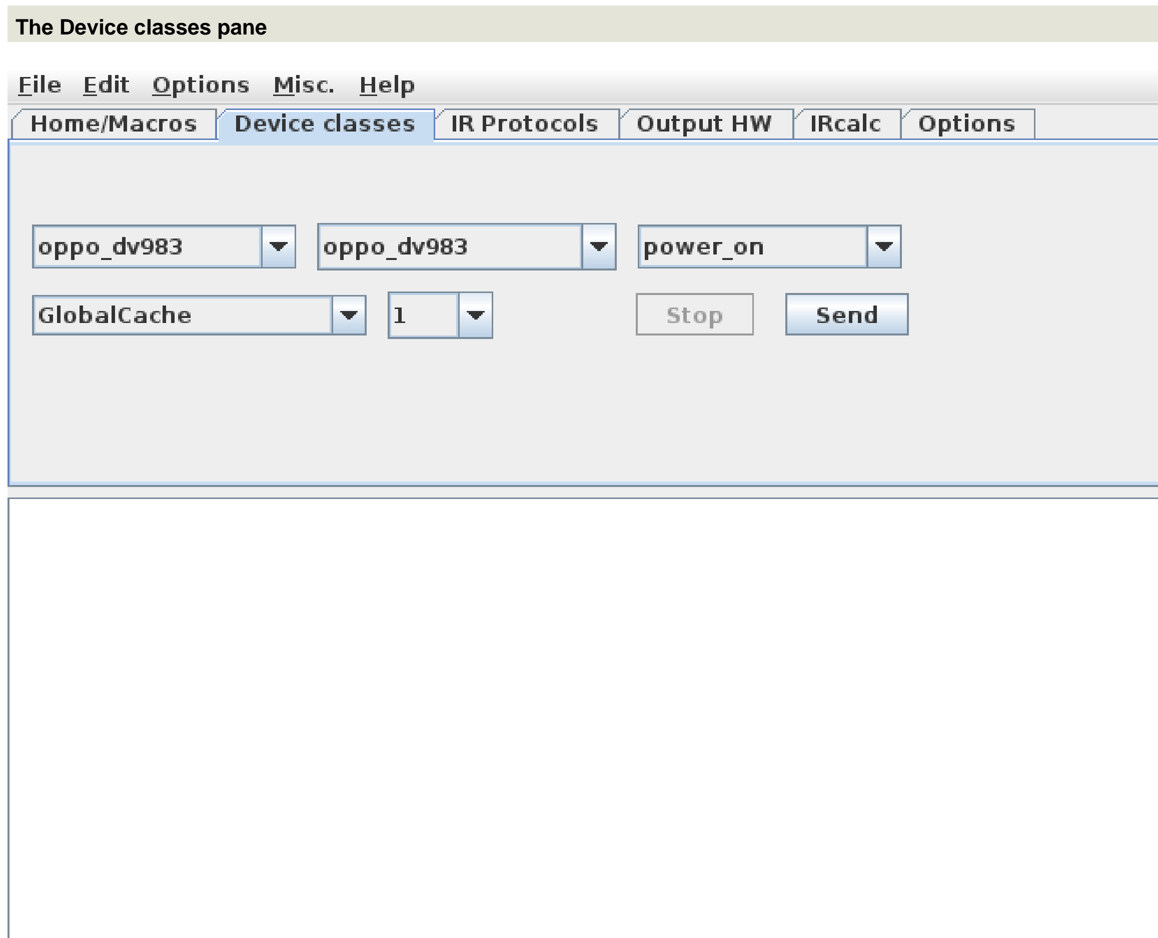
The GUI resides almost completely within the file `gui_main.java`. It was designed using the Netbeans IDE version 6.5.

The Home/Macros pane

This pane corresponds to using Harctoolbox 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".

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.

The IR Protocols pane

File Edit Options Misc. Help

Home/Macros | Device classes | IR Protocols | Output HW | IRcalc | Options

nec1 | 1 | 2 | 3 | no_toggle | 1 | GlobalCache | Encode

Cooked 900A 006D 0000 0001 0102 03fc | Decode

Raw Code

```
0000 006b 0022 0002 0150 00a8 0015 003f 0015 0015 0015 0015
0015 0015 0015 0015 0015 0015 0015 0015 0015 0015 0015 0015
0015 003f 0015 0015 0015 0015 0015 0015 0015 0015 0015 0015
0015 0015 0015 003f 0015 003f 0015 0015 0015 0015 0015 0015
0015 0015 0015 0015 0015 0015 0015 0015 0015 0015 0015 003f
0015 003f 0015 003f 0015 003f 0015 003f 0015 003f 0015 0666
0150 0054 0015 0e31
```

ict... | Send | Stop | Clear

protocol = NEC1, device = 1, subdevice = 2, obc = 3

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.

The Output HW pane

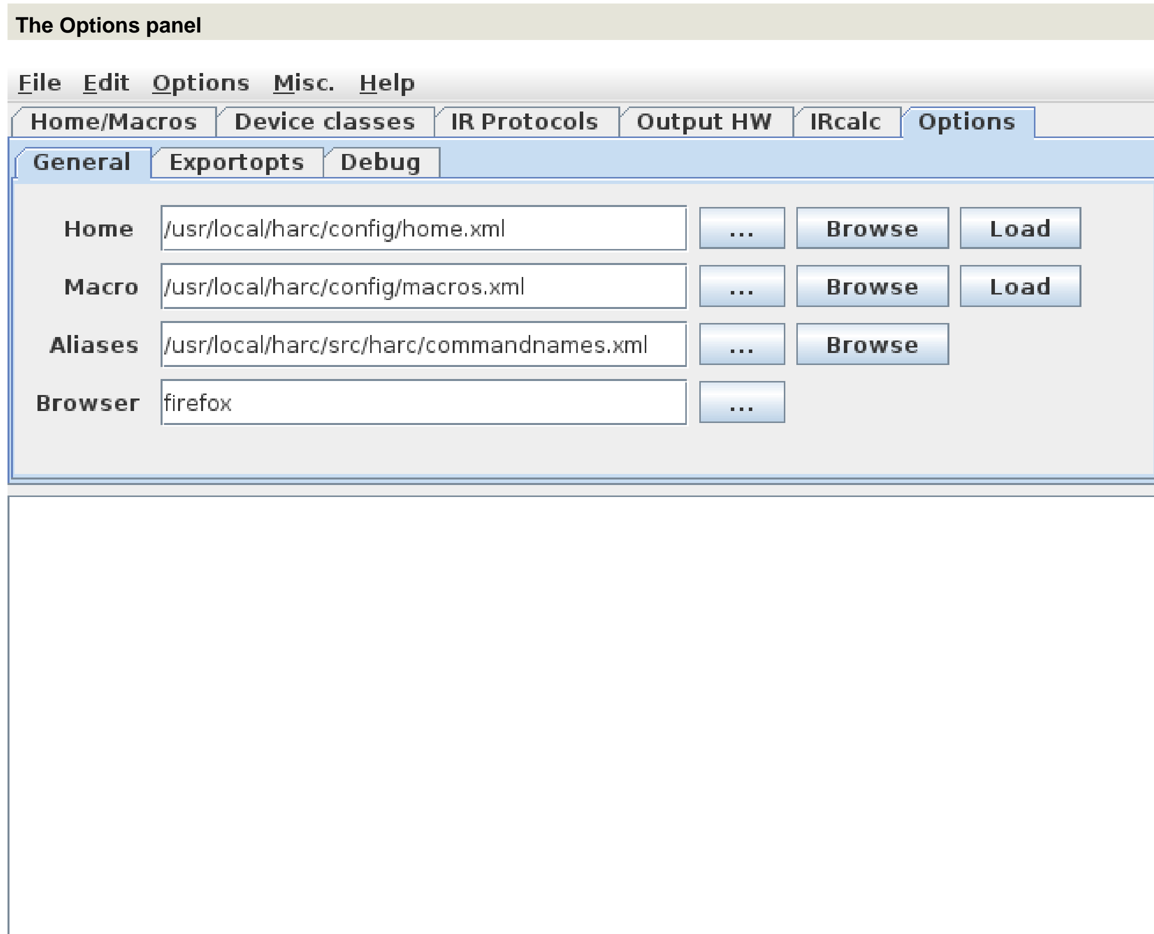
```
File  Edit  Options  Misc.  Help
Home/Macros | Device classes | IR Protocols | Output HW | IRcalc | Options
GlobalCache | IRTrans | EZControl

Intertechno   A   1   1   On   Off
4   Screen   off   On   Off   Update

IP-Address   192.168.1.42   Browse   Get Status   Get Timers

8.      water_wall: off
9.      large_blind: off
10.     small_blind: on
11.     balcony_blind: off
12.     kitchen_blind: off
13.     Bedroom_light: ?
14.     Bedroom_blind: off
15.     Kitchen: off
16.     Hall: off
17.     Aux_1: off
18.     Aux_2: off
19.     Aux_3: ?
20.     conr. 1324:1: ?
21.     x10:N1: ?
22.     x10:N2: ?
23.     x10:N3: ?
```

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.

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...

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.

3.4.7.5 Properties

Harctoolbox 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).

3.5 lirc2xml: Extracting information from LIRC configuration files

Date	Description
2010-03-20	Initial version.
2012-04-21	Moved from www.bengt-martensson.de/harc/lirc2xml.html to www.harctoolbox.org. Compatibility with current version or Lirc, 0.9.0, documented. Minor errors in document fixed.

Date	Description
2012-05-01	Created version 0.1.2, only small changes. Created binaries. Minor improvements.
2013-02-17	Created version 0.1.3, containing an important bugfix. Updated patch and Linux 64-bit executable (not the other binaries).

Table 1: Revision history

3.5.1 Introduction

Converting LIRC configuration files to a raw representation is no easy task. The meaning of the different parameters are to some extent documented through comments in the code, but are not at all straightforward. (Interestingly, the LIRC documentation is referring to WinLIRC for documentation...) I have therefore written `lirc2xml` as an extension to LIRC itself. It uses internal LIRC functions from `transmit.c` to render the signals. For this reason, it is available as a patch to the LIRC sources.

The program generates the CCF (often called "Pronto format") representation of the raw signal, embedded in an XML file. Optionally, the DecodeIR library is invoked to attempt to decode the signals.

The program was announced on the Lirc mailing list on 2009-08-22. It did not receive any attention. Ideally, I would like to see the program in the future LIRC distribution, however I have a strong suspicion that LIRC-ers would rather see *foobar*`2lirc` than `lirc2`*foobar*...

The program runs from the command line in a Linux environment. Current version is called 0.1.3. It was developed under Lirc 0.8.6, but works without changes with the current version, 0.9.0.

3.5.2 Usage

The usage is given by:

```
Usage: lirc2xml [options] [config-file]
         -h --help                      display this message
         -v --version                   display version
         -o --output=filename           XML output filename
         -r --remote=remotename         Include only this remote in the export
         -d[debug_level] --debug[=debug_level]
```

The Lirc configuration file, defaulting to `lirc.conf` in the current directory, is read and translating into an output file, defaulting to `./`*basename-of-configfile*`.xml`. Per default, all remotes in the configuration file are translated to the output file, unless, using the `-r`/`--remote`-option is given, in which case only the selected remote is processed.

If configured, John Fine's library DecodeIR is invoked for each signal. It will try to identify the signal as one of a number of known IR signal protocol.

3.5.2.1 Limitation

The present version does not render toggle signals (more precisely: it renders them only for one value of the toggle). Also, the treatment of repeats may be incorrect. A bug related to repeats showed up in NEC1-like signal, and was fixed in version 0.1.3.

3.5.3 Installation

The following instruction is intended for users with some experience with installation programs under Linux.

1. (Optional, but highly recommended.) Install `libDecodeIR.so` to a directory like `/usr/local/lib`, where the linker will find it.
2. Download and unpack lirc-0.9.0. Probably other, similar, versions will also work.
3. Change to the top of the recently unpacked tree and apply the patch by `patch -p1 < path-to-lirc2xml.patch`.
4. Regenerate some autoconfig files by `autoreconf -f -i -s`.
5. Configure by, e.g., `./configure --with-driver=none --enable-decodeir --enable-debug`. Error messages from configure concerning `lirc_tell_me_what_version_is` can be ignored.
6. The command `make` will now, in addition to all the usual Lirc programs, create the `lirc2xml` program, located in the `tools` subdirectory.
7. `make install` will install it together with the rest of the programs in the Lirc package, per default in `/usr/local/bin`. Just copying `tools/lirc2xml` should (in this case) suffice (although in general not recommended practice).

3.5.4 Known bug

Command names in Lirc files containing less-than characters (`<`) and ampersands (`&`) will generate nonvalid XML. I have no intention to fix this in the short time span. It is hard to write waterproof code for that without using xml libraries; that kind of characters in command names is silly anyhow, and broken XML it can also easily be fixed with a text editor.

3.5.5 lirc2rmdu

Mainly as a proof of concept, I wrote a simple back-end to the lirc2xml program: *lirc2rmdu*, written in Python. (Possibly some would argue, that it should be called xml2rmdu, however I felt that would be more misleading.) It transforms the xml file from lirc2xml to a rmdu-file to be used by RemoteMaster program to design a "device upgrade" in the JP1 context. Note that, for several different reasons, the output of the program is not meant to be a perfect rmdu file, but rather a first starting point, however saving a lot of work transforming what can be automatically transformed. In particular,

the device number and -parameters need some fixing. possibly with a text editor outside of RemoteMaster -- actually I manage to "hang" the program on a few occations. Fixing this may be possibly by parsing RemoteMaster's `protocols.ini` file, but I am not sure that is a wise way of spending my time...

```
Usage:
        lirc2rmdu.py [-r remote] <xmlfile>
```

The program will extract the remote given by the -r argument (or the first one if not given), and write it to a file in the current directory, using a named taken from the remote's name.

3.5.6 Downloads

3.5.6.1 Sources etc

- lirc2xml.patch current version 0.1.3.
- lirc2rmdu

3.5.6.2 Binaries

- Binary for Linux 64bin. Just dropping in (e.g.) `/usr/local/bin`, while dropping `libDecodeIR.so` (64 bit version) (see below) in `/usr/local/lib` should be enough.
- Binary for Linux 32bin. (Unfortunately, the old version 0.1.2) Just dropping in (e.g.) `/usr/local/bin`, while dropping `libDecodeIR.so` (32 bit version) (see below) in `/usr/local/lib` should be enough.
- Cygwin binaries for Windows. (Unfortunately, the old version 0.1.2) This executable was created using a lot of dirty tricks... It was compiled under Cygwin, but contains the necessary Cygwin libraries, so it should be usable without Cygwin. Just unpack in an empty directory, start a DOS-box, and cd to said directory. Cygwin users should probably delete the Cygwin dlls, since they should already reside in their system, and may otherwise cause a conflict. It contains DecodeIR, but hard linked in, not as a dll.

3.5.6.3 Third party software

- lirc in current version, 0.9.0. Also "sufficiently similar" versions will do.
- DecodeIR; either the precompiled `libDecodeIR.so` (`DecodeIR.dll`) (version 2.43) for Windows, Linux (32 and 64 bit), and Mac OS X can be downloaded from the JP1-forum. The source code is also found in the JP1 forum. (Update: Current version is 2.44; the links go to 2.43 (which however is still usable) and needs to be updated.)

4 API Documentation

4.1 HarctoolboxBundle 1.1.3 API

4.2 Girr 1.0.3 API

4.3 HarctoolboxBundle 1.1.3 API

4.4 HarctoolboxBundle 1.1.3 API

4.5 HarctoolboxBundle 1.1.3 API

4.6 HarctoolboxBundle 1.1.3 API

5 Downloads

5.1 HARCToolbox downloads

Date	Description
2012-05-01	Initial version.
2012-06-06	Updated for the releases 0.2.0 of IrMaster and IrpMaster.
2012-08-19	Updated for the release 0.3.0 of IrMaster and 0.2.1 of IrpMaster.
2012-11-18	Updated for the release 0.3.1 of IrMaster and 0.2.2 of IrpMaster.
2013-02-17	Updated for the release 0.1.3 of lirc2xml.
2014-02-02	Updated for the current setup.
2014-06-12	Updated for the IrScrutinizer 1.1.0 etc.
2014-09-27	Updated for the IrScrutinizer 1.1.1 etc.
2015-04-16	Updated for the IrScrutinizer 1.1.2 etc.
2015-09-10	Updated for the IrScrutinizer 1.1.3 etc.

Table 1: Revision history

5.1.1 Current software

Most of the software is also available on Github, including sources for the most current, non-released, version.

5.1.1.1 Binaries

- IrScrutinizer 1.1.3 binaries, including third-party libraries,
- IrScrutinizer 1.1.3 setup.exe for Windows,
- IrScrutinizer 1.1.3 as Mac OS X App in compressed format,
- IrMaster 1.0.0 binaries, including third-party libraries,
- IrMaster 1.0.1 setup.exe for Windows.
- lirc_rpi.ko Improved Raspberry Pi Lirc driver, compiled.

5.1.1.2 Sources

- HarctoolboxBundle-1.1.3-sources, the combined sources for IrScrutinizer 1.1.3, IrpMaster, Girr, HarcHardware.
- IrMaster 1.0.0 sources
- lirc_rpi.c Improved Raspberry Pi Lirc driver, source.

5.1.2 Old versions

Older versions of my software is made available here. The current version is always found above, under Current software.

- IrMaster
  - IrMaster 1.0.0 sources
  - IrMaster 1.0.0 binaries, including third-party libraries
  - IrMaster version 0.1.2 sources.
  - IrMaster version 0.1.2 binaries, including IrpMaster binaries and third party libraries.
  - setup.exe for Windows of IrMaster version 0.1.2, including IrpMaster binaries and third party libraries.
  - IrMaster version 0.2.0 sources.
  - IrMaster version 0.2.0 binaries, including IrpMaster binaries and third party libraries.
  - setup.exe for Windows of IrMaster version 0.2.0, including IrpMaster binaries and third party libraries.
  - IrMaster setup.exe for Windows of IrMaster version 0.3.0, including IrpMaster binaries and third party libraries.
  - IrMaster version 0.3.0 sources.
  - IrMaster version 0.3.0 binaries, including IrpMaster binaries and third party libraries.

    - IrMaster version 0.3.1 sources.
    - IrMaster version 0.3.1 binaries, including IrpMaster binaries and third party libraries.
    - IrMaster setup.exe for Windows of IrMaster version 0.3.1, including IrpMaster binaries and third party libraries.
- IrScrutinizer
    - IrScrutinizer 1.1.2 binaries, including third-party libraries
    - IrScrutinizer 1.1.2 setup.exe for Windows
    - IrScrutinizer 1.1.2 as App in compressed disk image for Mac OS X
    - IrScrutinizer 1.1.1 sources
    - IrScrutinizer 1.1.1 binaries, including third-party libraries
    - IrScrutinizer 1.1.1 setup.exe for Windows
    - IrScrutinizer 1.1.0 setup.exe for Windows
    - IrScrutinizer 1.1.0 sources
    - IrScrutinizer 1.1.0 binaries, including third-party libraries
    - IrScrutinizer 1.0.0 sources
    - IrScrutinizer 1.0.0 binaries, including third-party libraries
    - IrScrutinizer 1.0.0 and IrMaster 1.0.0 setup.exe for Windows, in one package!
    - IrScrutinizer version 0.1.1 sources.
    - IrScrutinizer version 0.1.1 binaries, including IrpMaster binaries and third party libraries.
    - IrScrutinizer setup.exe for Windows of IrMaster version 0.1.1, including IrpMaster binaries and third party libraries.
- IrpMaster
    - IrpMaster 1.0.2 sources
    - IrpMaster 1.0.1 sources
    - IrpMaster 1.0.0 sources
    - IrpMaster version 0.1.2 sources.
    - IrpMaster version 0.2.0 sources.
    - IrpMaster version 0.2.1 sources.
    - IrpMaster version 0.2.2 sources.
- Misc
    - HarctoolboxBundle-1.0.0-sources, the combined sources for IrScrutinizer 1.1.2, IrpMaster, Girr, HarcHardware. (GuiComponents has been integerated into IrScrutinizer.)
    - Girr 1.0.2 sources
    - Girr 1.0.1 sources
    - Girr 1.0.0 sources
    - Jirc 0.3.0 sources
    - Jirc 0.2.0 sources
    - GuiComponents 0.2.2 sources
    - GuiComponents 0.2.1 sources
    - GuiComponents 0.2.0 sources

  - HarcHardware 0.9.2 sources
  - HarcHardware 0.9.1 sources
  - HarcHardware 0.9.0 sources
  - HarcHardware 0.8.0 sources
  - IrCalc 0.2.0 sources from my Subversion repository.
  - Lirc 0.9.0 patch for CCF processing.
  - lirc2xml.patch version 0.1.3.
  - lirc2rmdu.py. A post-processor for lirc2xml.
- Old HarcToolbox
  - Harctoolbox 0.5.0 (then called "Harc"). Source and binaries included.
  - Harctoolbox 0.6.0 sources.
  - Harctoolbox 0.6.0 binaries.
  - Harctoolbox 0.7.0 sources from my Subversion repository.
  - Harctoolbox 0.7.0 binaries, including third part jar-archives.
- lirc2xml
  - lirc2xml-0.1.2-windows.zip
  - lirc2xml-0.1.2.patch
  - lirc2xml-0.1.3.patch
  - lirc2xml.patch
  - lirc2xml_amd64
  - lirc2xml_amd64_0.1.2
  - lirc2xml_i386

5.1.3 Old IR protocols

> **Note:**
> These files are now considered obsolete, since IrScrutinizer, IrMaster and IrpMaster can generate these protocols, and export them in a number of different formats.

- Intertechno.xml Pronto codes for the Intertechno (and similar) RF switches ("with code wheel").
- conrad.txt Pronto codes for the RF switches once manufactured by Conrad electronics, sometimes known as RS-200.

5.1.4 Third party sources with modifications

In one case, I have made minor modifications, (not even necessesarily "improvements" :-) to third party sources. The modified sources are available here, in the spirit of GPL.

- PtPlot 5.8, a fix for using "engineering scales", i.e. selecting the scaling as ten to the power of a multiple of three.

5.1.5 Third party jars

For convenience, the used jar-archives can be downloaded here. Corresponding sources can either be downloaded from the Internet using URLs published in this page, or (as required by the GPL) requested from me.

- `libreadline-java.jar`, version 0.8.0. Homepage.
- `sunrisesunset.jar`. Homepage. I have made some improvements, which has been sent to the author and subsequently included.

More information may be found in the other projects page.

5.1.6 Third party dynamic libraries

5.1.6.1 JavaReadline

JavaReadline, which is a wrapper around GNU Readline, making it usable from Java programs, has been "maven-ized" and "git-tified" by myself on Github.

An old version of `libJavaReadline.so` for 64-bit Linux is available here. Precompiled dynamic libraries `libreadline.so` and `libhistory.so` should be available from any Linux distributor, package name probably `readline` or so.

Note that these components are needed only to enable some special functionality. 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.

6 All

6.1 Site Linkmap Table of Contents

This is a map of the complete site and its structure.

- MyProj ___________________ *site*
  - Home ___________________ *home*
    - Welcome ___________________ *index* : Homepage
    - News ___________________ *news* : News
    - Other projects ___________________ *other_projects* : Other projects, links
    - Impressum ___________________ *impressum* : Over the site
    - Legal ___________________ *legal* : Legal blurbl
  - Current ___________________ *current*
    - Current program index ___________________ *index* : Current programs and documents

    - IrScrutinizer ________________ *irscrutinizer* : Very advanced IR program
    - IrpMaster ________________ *irpmaster* : Rendering of IR signals
    - Girr ________________ *girr* : General IR Remote format
    - Girs ________________ *girs* : General IR Server specification
    - Architecture ________________ *architecture* : Courses architecture concept
    - Glossary ________________ *glossary* : Glossary and terms
    - HarcHardware ________________ *harchardware* : Library for hardware access
    - Jirc ________________ *jirc* : Lirc translated to Java
    - Improved lirc_rpi driver ________________ *lirc_rpi* : Improved LIRC driver for the Raspberry Pi
    - Raspberry Pi IR & RF hardware ________________ *rpi_daughterboard* : Raspberry Pi IR and RF hardware daughter board
    - Lirc CCF patch ________________ *lirc_ccf* : Making Lirc server handle CCF signals
- Old programs and docs ________________ *old*
    - Old program index ________________ *index* : Some old programs and documents
    - IrMaster ________________ *irmaster* : GUI IR program
    - Transforming XML ________________ *xslt_tutorial* : Transforming the XML export into other formats
    - Old Harctoolbox ________________ *harctoolbox* : Old main project
    - Lirc2xml ________________ *lirc2xml* : Making Lirc files usable for the rest of us
- API Documentation ________________ *api_documentation*
    - API doc. index ________________ *index* : API documentation (javadoc)
    - Girr API doc ________________ *girr_doc* : Girr API documentation
    - HarcHardware API doc ________________ *harchardware_doc* : HarcHardware API documentation
    - IrpMaster API doc ________________ *irpmaster_doc* : IrpMaster API documentation
    - IrScrutinizer API doc ________________ *irscrutinizer_doc* : IrMaster API documentation
    - Jirc API doc ________________ *jirc_doc* : Jirc API documentation

- Downloads ___________________ *downloads*
    - Download index ___________________ *index* : Download cool software!
- All ___________________ *all*
    - Sitemap ___________________ *linkmap* : Site Linkmap
    - Whole Site HTML ___________________ *whole_site_html*
    - Whole Site PDF ___________________ *whole_site_pdf*