

The Girr format for universal IR Commands and remotes.

Table of contents

1 Background and Introduction.....	3
1.1 Mission.....	4
2 Program support.....	4
3 Copyright.....	4
4 The name of the game.....	5
5 Requirements on a universal IR command/remote format.....	5
6 Demarcation.....	5
7 Informal overview of the Girr format.....	6
7.1 command.....	6
7.1.1 Fat Format.....	7
7.2 commandSet.....	7
7.3 remote.....	7
7.4 remotes.....	8
7.5 Embedded protocols.....	8
8 Detailed description of syntax and semantics of the Girr format.....	8
8.1 Version.....	8
8.2 Namespace.....	8
8.3 Imported namespaces.....	8
8.4 Schema.....	8
9 Stylesheets.....	8
10 Supporting Java library.....	9
11 GirrLib.....	9
12 Integration in Maven projects.....	9
13 Sources.....	10

14 Appendix. Parametrized IrSignals..... 10

Date	Description
2014-01-28	Initial version.
2016-04-28	Updated to current version.
2019-07-21	Updated to current version; IrpMaster replaced by IrpTransmogriifier. Say "Pronto Hex" instead of "CCF".
2020-05-29	Some minor updates. Added Maven integration (which previously was found in README.md).
2021-06-03	Several minor fixes and clarifications, in particular embedded protocols.

Table 1: Revision history

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 [Pronto Hex](#), sometimes called CCF, 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 Pronto Hex 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". However, 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 not a viable concept of [intro- and repeat sequences](#).

This leads to our mission:

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 Pronto Hex. 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 Program support

[IrScrutinizer](#) uses Girr as its preferred format for import and export of IR signals. It can interactively import and export from many different file formats and data bases. However, it presently supports only files with `remotes` as the root element. (It is planned to eliminate this restriction in the near future.)

[RMIR](#) (sometimes called RemoteMaster) is a powerful program for programming so-called [JPI](#)-remotes. Since version 2.11, it can export and import Girr files directly.

[Jirc](#) can generate Girr files from Lirc configuration files. (It is also included in IrScrutinizer.)

[IrpTransmogriifier](#) can generate the output from the `decode` and the `analyze` commands in Girr format. (In order to avoid circular dependencies, it does not use the [support library](#) described here.)

[GirrLib](#) is a small collection of Girr files.

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](#).

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.

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.

A remote is in principle nothing else than 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 "arbitrary", in any language or character set, using any printable characters including white space. However, it is recommended to use "simple" names in English, with no special characters or embedded whitespace. (A `displayName` can be used for localized names, with arbitrary characters etc.) A well defined semantic of command names is not granted. However, in some cases uniqueness in the purely syntactical sense is required, for example ensuring that all commands within a particular `commandSet` 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 parametrized protocols, in raw form, or in Pronto Hex 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.

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.

7 Informal overview of the Girr format

There are four different possible root element types in the format: `remotes`, `remote`, `commandSets`, and `commands`. All can be the root element of a conforming Girr document, although some software may not handle all of them. (The previous versions of our [supporting library](#) only supported `remotes` as root element.) Basically, the element `remotes` contains one or more `remotes`, each containing one or more `commandSets`, each containing a number of `commands`.

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
    0016 0041 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 0016 0016
    0041 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 Pronto Hex form may be given next, as above. Finally, one or may auxiliary formats of the signal can be given.

7.1.1 Fat Format

For the ease of further processing of the result, the sequences within the `<raw>` element can alternatively 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>
    </intro>
  </raw>
</command>
```

7.2 commandSet

`commandSets` bundles "related" commands together. They may contain `parameters` elements, in which case the protocol name and the parameters therein are inherited to the contained commands.

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.

Often, a device can be controlled in one of several "id-s", corresponding to different IR signals. See the [Oppo BDP-83](#) as example. The different id-s are modeled as different `commandSets`. These are (almost) identical, typically differing only in the device/subdevice parameter in the protocol. In other cases, devices implement a "new" and an "old" set of commands; see for example a [Philips TV](#) (RC5 and RC6 protocols) or a [Denon AVR receiver](#) (old "Denon" protocol and new "Denon-K" protocol).

7.3 remote

A `remote` is an abstract "clicker", containing a number of `commands`. The name of the contained commands must be unique within a `commandSet`, but the same name may be present in more than one `commandSet`.

7.4 remotes

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

7.5 Embedded protocols

The parameter form references to a protocol using its name. Normally, this is assumed to be known to a processing program. However, it is also possible to embed additional protocols in a Girr file, and to refer to it by its defined name in the declaration.

8 Detailed description of syntax and semantics of the Girr format

8.1 Version

This article describes the Girr format version 1.2, identified by the attribute `girrVersion`, expected in the root element of an instance. (Not to be confused with the version of the [support library](#).)

8.2 Namespace

The Girr [namespace](#) is `http://www.harctoolbox.org/Girr`.

It is recommended to parse instances with a namespace- and XInclude-aware parser.

8.3 Imported namespaces

Except for the namespace namespace (`http://www.w3.org/XML/1998/namespace`), the namespaces XInclude (`http://www.w3.org/2001/XInclude`) and html (`http://www.w3.org/1999/xhtml`) are imported. For embedding of additional protocols, the irp namespace (`http://www.harctoolbox.org/irp-protocols`) is used; imported only on demand. XInclude- and html elements can be used at appropriate places, see the schema.

8.4 Schema

The grammar of Girr is formally described as an [XML schema](#) residing in the file [girr_ns.xsd](#). It contains internal documentation of the semantics of the different elements. The official schema location is http://www.harctoolbox.org/schemas/girr_ns.xsd.

Here is [generated schema documentation](#) (thanks to Gerald Manger).

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.

XSLT style-sheets can however be used for other purposes than the name suggests. The export mechanism of IrScrutinizer consists essentially of the application of XSLT stylesheets on the Girr fat format.

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](#).

At the time of writing, the library carries the version number 2.2.10.

Previous versions only supported import and export of documents having `remotes` as root element.

The library requires the [IrpTransmogriifier](#) classes.

11 GirrLib

I maintain a small library, GirrLib, available at [GitHub](#). It consists of "my" collection of Girr files. Although not actively maintained, contributions are welcome. The files therein are in the public domain.

12 Integration in Maven projects

This project can be integrated into other projects using Maven. For this, include the lines

```
<dependency>
  <groupId>org.harctoolbox</groupId>
  <artifactId>Girr</artifactId>
  <version>1.2.3</version> <!-- or another supported version -->
</dependency>
```

in the `pom.xml` of the importing project. This will also include the `[IrpTransmogriifier]` (<http://harctoolbox.org/IrpTransmogriifier.html>) jar.

13 Sources

The sources, both the Java library, the schema, and the current document, are maintained at this [Github repository](#). API documentation (current development version) is available [here](#).

14 Appendix. Parametrized 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](#).