Using and Transforming XML export

Table of contents

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.

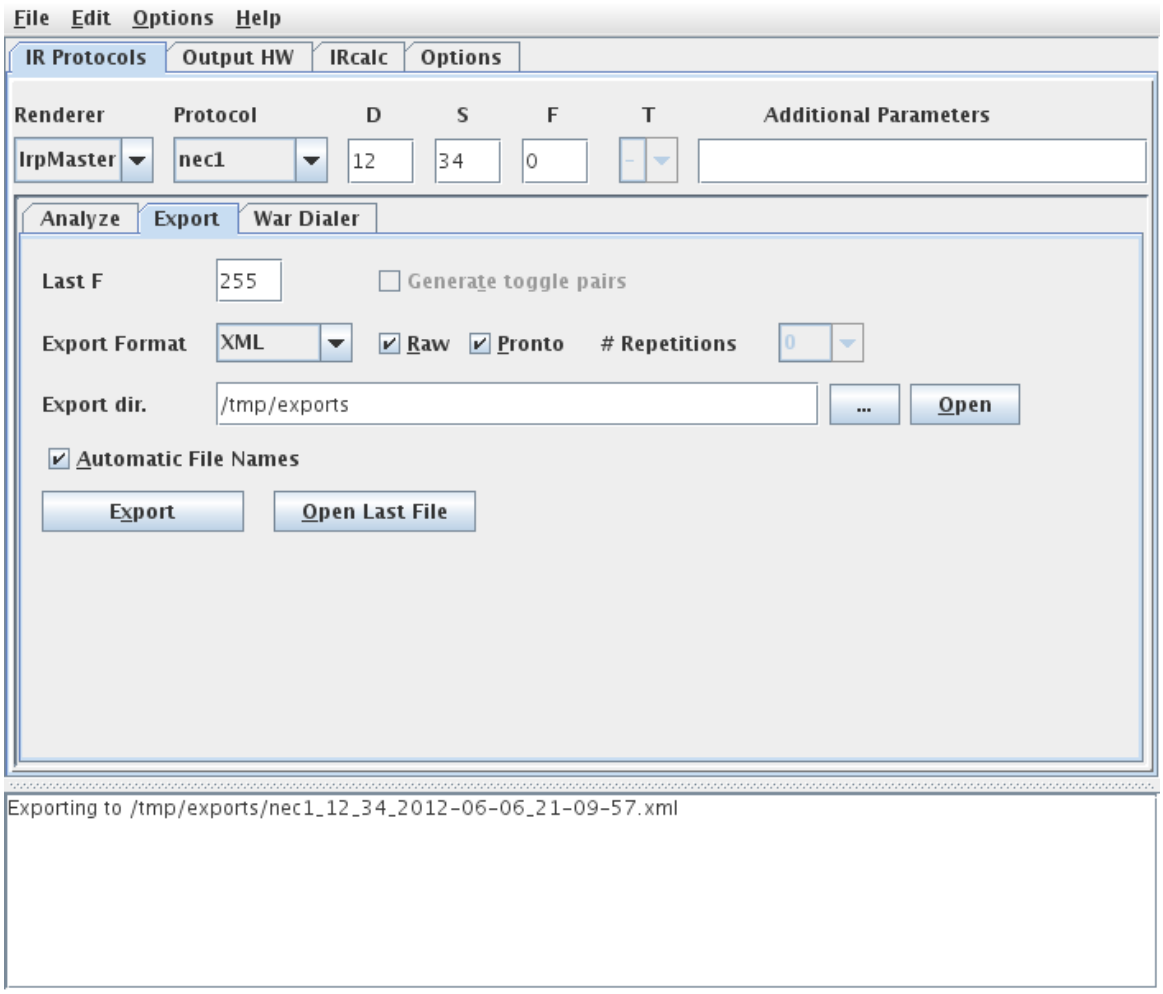
1 Description

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.

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 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 \*
```