```
#!/usr/bin/env python
"""\
Parallax Propeller code uploader
Copyright (C) 2007 Remy Blank

This file is part of PropTools.

This program is free software; you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by the
Free Software Foundation, version 2.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General
Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software Foundation,
Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA
"""

import optparse
import os
import time
import serial

# Processor constants
lfsrRequestLen = 250
lfsrReplyLen = 250
lfsrSeed = ord("P")
cmdShutdown = 0
cmdLoadRamRun = 1
cmdLoadEeprom = 2
cmdLoadEepromRun = 3

# Platform defaults
defSerial = {
    "posix": "/dev/ttyUSB0",
    "nt": "COM1",
}


def lfsr(seed):
    """Generate bits from 8-bit LFSR with taps at 0xB2."""
    while True:
        yield seed & 0x01
        seed = ((seed << 1) & 0xfe) | (((seed >> 7) ^ (seed >> 5) ^ (seed >> 4) ^ (seed >> 1)) & 1)


def encodeLong(value):
    """Encode a 32-bit long as short/long pulses."""
    result = []
    for i in range(10):
        result.append(chr(0x92 | (value & 0x01) | ((value & 2) << 2) | ((value & 4) << 4)))
        value >>= 3
    result.append(chr(0xf2 | (value & 0x01) | ((value & 2) << 2)))
    return "".join(result)


def doNothing(msg):
    """Do nothing progress callback."""
    pass


class LoaderError(Exception): pass


class Loader(object):
    """Propeller code uploader."""
    eepromSize = 32768

    def __init__(self, port):
```

```
        self.serial = serial.Serial(baudrate=115200, timeout=0)
        self.serial.port = port

    # High-level functions
    def getVersion(self, progress=doNothing):
        """Connect to the Propeller and return its version."""
        self.open()
        try:
            version = self.connect()
            self.writeLong(cmdShutdown)
            time.sleep(0.010)
            self.reset()
            return version
        finally:
            self.close()

    def upload(self, code=None, path=None, eeprom=False, run=True, progress=doNothing):
        """Connect to the Propeller and upload code to RAM or EEPROM."""
        if path is not None:
            f = open(path, "rb")
            try:
                code = f.read()
            finally:
                f.close()
        self.open()
        try:
            version = self.connect()
            progress("Connected (version=%d)" % version)
            self.sendCode(code, eeprom, run, progress)
        finally:
            self.close()

    # Low-level functions
    def open(self):
        self.serial.open()

    def close(self):
        self.serial.close()

    def reset(self):
        self.serial.flushOutput()
        self.serial.setDTR(1)
        time.sleep(0.025)
        self.serial.setDTR(0)
        time.sleep(0.090)
        self.serial.flushInput()

    def calibrate(self):
        self.writeByte(0xf9)

    def connect(self):
        self.reset()
        self.calibrate()
        seq = []
        for (i, value) in zip(range(lfsrRequestLen + lfsrReplyLen), lfsr(lfsrSeed)):
            seq.append(value)
        self.serial.write("".join(chr(each | 0xfe) for each in seq[0:lfsrRequestLen]))
        self.serial.write(chr(0xf9) * (lfsrReplyLen + 8))
        for i in range(lfsrRequestLen, lfsrRequestLen + lfsrReplyLen):
            if self.readBit(False, 0.100) != seq[i]:
                raise LoaderError("No hardware found")
        version = 0
        for i in range(8):
            version = ((version >> 1) & 0x7f) | ((self.readBit(False, 0.050) << 7))
        return version

    def binToEeprom(self, code):
        if len(code) > self.eepromSize - 8:
            raise LoaderError("Code too long for EEPROM (max %d bytes)" % (self.eepromSize - 8))
        dbase = ord(code[0x0a]) + (ord(code[0x0b]) << 8)
        if dbase > self.eepromSize:
            raise LoaderError("Invalid binary format")
```

```
        code += "".join(chr(0x00) * (dbase - 8 - len(code)))
        code += "".join(chr(each) for each in [0xff, 0xff, 0xf9, 0xff, 0xff, 0xff, 0xf9, 0xff])
        code += "".join(chr(0x00) * (self.eepromSize - len(code)))
        return code

    def sendCode(self, code, eeprom=False, run=True, progress=doNothing):
        if len(code) % 4 != 0:
            raise LoaderError("Invalid code size: must be a multiple of 4")
        if eeprom and len(code) < self.eepromSize:
            code = self.binToEeprom(code)
        checksum = reduce(lambda a, b: a + b, (ord(each) for each in code))
        if not eeprom:
            checksum += 2 * (0xff + 0xff + 0xf9 + 0xff)
        checksum &= 0xff
        if checksum != 0:
            raise LoaderError("Code checksum error: 0x%.2x" % checksum)
        command = [cmdShutdown, cmdLoadRamRun, cmdLoadEeprom, cmdLoadEepromRun][eeprom * 2 + run]
        self.writeLong(command)
        if not eeprom and not run:
            return
        self.writeLong(len(code) // 4)
        progress("Sending code (%d bytes)" % len(code))
        i = 0
        while i < len(code):
            self.writeLong(ord(code[i]) | (ord(code[i + 1]) << 8) | (ord(code[i + 2]) << 16) |
(ord(code[i + 3]) << 24))
            i += 4
        if self.readBit(True, 8) == 1:
            raise LoaderError("RAM checksum error")
        if eeprom:
            progress("Programming EEPROM")
            if self.readBit(True, 5) == 1:
                raise LoaderError("EEPROM programming error")
            progress("Verifying EEPROM")
            if self.readBit(True, 2.5) == 1:
                raise LoaderError("EEPROM verification error")

    # Lowest-level functions
    def writeByte(self, value):
        self.serial.write(chr(value))

    def writeLong(self, value):
        self.serial.write(encodeLong(value))

    def readBit(self, echo, timeout):
        start = time.time()
        while time.time() - start < timeout:
            if echo:
                self.writeByte(0xf9)
                time.sleep(0.025)
            c = self.serial.read(1)
            if c:
                if c in (chr(0xfe), chr(0xff)):
                    return ord(c) & 0x01
                else:
                    raise LoaderError("Bad reply")
        raise LoaderError("Timeout error")


def upload(serial, path, eeprom=False, run=True, progress=doNothing):
    """Upload file on given serial port."""
    loader = Loader(serial)
    progress("Uploading %s" % path)
    loader.upload(path=path, eeprom=eeprom, run=run, progress=progress)
    progress("Done")


def watchUpload(serial, path, delay, eeprom=False, run=True, progress=doNothing):
    """Upload file on given serial port, and keep watching for changes and uploading."""
    loader = Loader(serial)
    firstLoop = True
    mtime = None
```

```
    while True:
        try:
            prevMTime = mtime
            try:
                mtime = os.stat(path).st_mtime
            except OSError:
                mtime = None
            if (mtime is not None) and (mtime != prevMTime):
                if not firstLoop:
                    progress("File change detected")
                    time.sleep(delay)
                progress("Uploading %s" % path)
                loader.upload(path=path, eeprom=eeprom, run=run, progress=progress)
                progress("Done\n")
            else:
                time.sleep(1)
        except LoaderError, e:
            progress(str(e) + "\n")
        firstLoop = False


class HelpFormatter(optparse.IndentedHelpFormatter):
    """Slightly customized option help formatter"""
    def format_usage(self, usage):
        return "Usage: %s\n" % usage

    def format_heading(self, heading):
        if heading == "options":
            heading = "Options"
        return optparse.IndentedHelpFormatter.format_heading(self, heading)

    def format_description(self, description):
        if not description:
            return ""
        return description


def printStatus(msg):
    """Print status messages."""
    print msg


def main(argv):
    """Execute command-line program."""
    import sys

    parser = optparse.OptionParser(
        prog=os.path.basename(argv[0]),
        usage="%prog [options] path",
        description="Parallax Propeller uploader\n",
        formatter=HelpFormatter(),
        add_help_option=False,
    )
    try:
        import PropTools
        parser.description += """\
This program is part of %(project)s %(version)s (%(date)s)
%(copyright)s
""" % PropTools._metadata_.__dict__
        parser.version = "%prog " + PropTools._metadata_.version
    except ImportError:
        parser.description += """\
Copyright (C) 2007 Remy Blank
"""

    parser.add_option("-d", "--delay", dest="delay", nargs=1, type="float", metavar="N", default=1.0,
        help="In watch mode, wait N seconds after detecting a file change before uploading. The
default is %default.")
    parser.add_option("-e", "--eeprom", action="store_true", dest="eeprom", default=None,
        help="Program device EEPROM. The default is to program the EEPROM only if the path ends with
'.eeprom'.")
    parser.add_option("-h", "--help", action="help",
```

```
        help="Show this help message and exit.")
    parser.add_option("-n", "--no-run", action="store_false", dest="run", default=True,
        help="Don't run the code after upload.")
    parser.add_option("-r", "--ram", action="store_false", dest="eeprom",
        help="Program device RAM. The default is to program the RAM except if the path ends with
'.eeprom'.")
    parser.add_option("-s", "--serial", dest="serial", nargs=1, type="string", metavar="DEVICE",
default=defSerial.get(os.name, "none"),
        help="Select the serial port device. The default is %default.")
    parser.add_option("",   "--version", action="version",
        help="Show the program version and exit.")
    parser.add_option("-w", "--watch", action="store_true", dest="watch", default=False,
        help="Continuously watch the file and upload it if it changes.")

    (options, args) = parser.parse_args(argv[1:])
    if len(args) != 1:
        sys.stderr.write("Invalid number of arguments\n")
        parser.print_help(sys.stderr)
        return 2
    path = args[0]
    if options.eeprom is None:
        options.eeprom = path.endswith(".eeprom")

    try:
        if options.watch:
            watchUpload(options.serial, path, options.delay, options.eeprom, options.run, printStatus)
        else:
            upload(options.serial, path, options.eeprom, options.run, printStatus)
    except (SystemExit, KeyboardInterrupt):
        return 3
    except Exception, e:
        sys.stderr.write(str(e) + "\n")
        return 1


if __name__ == "__main__":
    import sys
    sys.exit(main(sys.argv))
\00
```