Technical Report

STOCC - SCORE Control Software

Prepared by
Lino Mastrodomenico

June 2008
Version 1.0

Version management

No	Date	Changes	Author
1	2008-06-20	First version	L. Mastrodomenico

Project manager: S. Fineschi

Contents

Abstract

ASCI (Adaptable Space Control Interface) is a generic software framework for command and control of astronomical hardware (mainly telescope cameras and related optical devices, but can also be used for controlling ground test systems). It is cross-platform and designed to be flexible enough to be reused for different devices and different projects using a plugin system. The development of a new plugin has been kept as easy as possible and at the same time the user interface provided by plugins is seamlessly integrated within ASCI from an end-user's point of view.

STOCC (*SCORE* Test & Operations, Command & Control) is one of the first ASCI plugins. It offers a real-world test for the architecture and provide ground tests and calibration to the SCORE project (SCORE, Sounding CORonagraph Experiment, is a payload for solar corona imaging in the *HERSCHEL* sounding rocket mission).

1.Introduction

1.1.Background

The *SCORE* (Sounding CORonagraph Experiment) experiment is a set of two coronagraphs designed to provide full images of the extended corona in the EUV and visible light. SCORE is part of the scientific payload of the NASA *HERSCHEL* (the Helium Resonant Scattering in the Corona and Heliosphere) sounding rocket mission, which is composed of SCORE and the Extreme Ultraviolet Imaging Telescope (EIT).

The first HERSCHEL rocket launch date is August 1, 2008. Depending on the success of the first launch, a second and a third launch are foreseen upon NASA approval.

1.2.Objectives

The main goal of the project is the development of an integrated test software for space- and astronomy-related devices, to be used for calibration and both pre-/post- flight testing.

The first application of this system is within the SCORE project, but the software must be flexible enough to be used in future projects with similar requirements (but different hardware) and even applications not strictly related to space or astronomy, such as the control of ground test equipment (e.g.: a cleanroom).

This implies the necessity to interface to a wide range of hardware devices (connected to the control computer through disparate buses and electrical interfaces) and offering a great variety of different *GUIs* (graphical user interfaces), specialized for each specific application.

The project is sponsored by the Italian region Piedmont and, partly, by the European Union.

2.Architecture overview

To satisfy the above goals the overall architecture of the project is split in two big parts:

1. *ASCI* (Adaptable Space Control Interface) is a generic software framework for command and control of cameras and sensors; ASCI is not tied to specific devices or GUI, but can load hardware drivers and user interfaces using a plugin system;
2. *STOCC* (SCORE Test & Operations, Command & Control) is a collection of ASCI plugins that will offer all the specific capabilities required for the SCORE project.

New plugins will be developed as necessary for different devices or projects.

3.Requirements

Apart from the main ASCI/STOCC goals, the requirements for this project are:

1. portability across the three major operating systems (GNU/Linux, Mac OS X and Microsoft Windows);
2. simple image elaboration capabilities: adding and subtracting two images, in both 16 and 32 bits per pixel, displaying images on the screen in linear and logarithmic scale, with user interfaces for "navigating" on common 256-grayscale levels monitors the very high dynamic range of images captured by modern cameras;
3. ability to read and write images and related metadata from and to FITS files;
4. ability to save to file the full raw data received from the cameras and possibility to load the data back at any time (even with no external hardware connected) and convert it to any other supported format.

It must be also possible to write drivers for a wide variety of hardware interfaces, including SpaceWire networks, RS-232 serial ports, IEEE 1284 parallel ports, direct communication with custom-programmed Arduino boards over USB and cards on a PCI-compatible bus. A possible requirement for programming devices through JTAG (IEEE 1149.1) interfaces has been examined, but is currently considered not necessary.

4.The ASCI application

ASCI is at the same time an integrated end-user application for the control of astronomical hardware and a development framework that simplifies the integration of existing device drivers and the creation of new drivers and their corresponding user interfaces. The description below reflects this dualism.

4.1.The choice of the programming language

The first programming language that has been considered for the implementation of ASCI is the National Instruments development environment *LabVIEW*.

LabVIEW has a number of advantages for this role, including the availability of drivers for most of the hardware devices used by SCORE and the familiarity with the framework by a number of people involved with the project. But in the end Python was preferred as the choice of programming language, mostly because of concerns that reaching the required flexibility for ASCI and maintaining the software afterwards may have been significantly

harder with the use of LabVIEW instead of a more general purpose language. LabVIEW also has the disadvantage of being distributed only by a single vendor on a limited number of platforms and under a proprietary license with a pretty high price.

Python was selected instead of other common general purpose programming languages because it has a few features that make it more appropriate for the open and flexible nature of ASCI; see the sections "The use of metaprogramming in ASCI" and "The programming model" for more details.

Still, the interoperability with LabVIEW, C, C++ and Java is required, for accessing hardware devices that only have (possibly proprietary) drivers for a limited number of environments. E.g., it's frequent for astronomical devices vendors to provide only drivers for LabVIEW and/or binary DLL files for Microsoft Windows.

This requirement can be achieved in a number of ways:

- using a separate program for the external language: it runs in a different process and communicates with ASCI using one or more pipes or internet sockets;
- integrating both languages/environments in the same process: this usually requires the use of the extending or embedding API of both sides with a small bridge module written in C. The creation of this module can be simplified or completely automatized with existing tools such as SWIG, SIP, Boost.Python or Pyrex;
- direct loading and invoking of an existing shared library (e.g., a Windows DLL file) with the ctypes Python library.

The last case is currently and will probably continue to be the more common; its main advantage is that it does not require the presence of a compiler during the installation on the target platform and, starting with Python 2.5, it does not require any dependency outside of the standard CPython package.

4.2. The use of metaprogramming in ASCI

In this context, metaprogramming can be defined as the writing of a program that manipulates itself as its data.

A good quotation to keep in mind when doing metaprogramming in Python is the following one by the Python guru Tim Peters: "[Metaclasses] are deeper magic than 99% of users should ever worry about. If you wonder whether you need them, you don't (the people who actually need them know with certainty that they need them, and don't need an explanation about why)."

Having said that, ASCI is one of the few examples where a limited use of metaprogramming, far from being "black magic", can make the creation of plugins easier and more elegant. The main reason why this happens in ASCI is that a segment of its clients is composed by the end-programmers who write and maintain the plugin. Their work is simplified by custom metaclasses that intercept the class creation (which in Python occurs at runtime, not during the compilation) and transform it in two ways:

- modify class attributes, integrating them in the inter-thread event system (see the section "The programming model");
- automatically register the presence of the class, making it available in the GUI and allowing the creation of class instances when necessary.

The overall effect is that class properties that must be constant (e.g., the name of a driver, the hardware to which it gives access, the status variables and their range) can be created in a declarative style, usually simpler than an imperative style here, and communication between plugins and the rest of the application is very straightforward: if, e.g., a driver receives a status update from its hardware, it can simply assign the new value to the corresponding variable and the GUI (and potentially a running script) will be updated automatically.

4.3.The programming model

One of the basic requirements for ASCI is making the addition of new device drivers for arbitrary hardware as easy as possible. This has far reaching consequences for the choice of the ASCI programming model, because drivers are usually built around one or more event loops that send and/or receive data from the device. An underlying proprietary driver may impose the use of its event loop, potentially incompatible with other ASCI event loops (i.e., other drivers or the GUI event loop), but even when this is not the case, writing and maintaining a driver is usually simpler when a plain I/O loop can be used, without any of the complications that can arise from the use of a framework based on the select or poll system calls.

To achieve this degree of simplification, each driver must run in its own thread or process. Currently ASCI uses threads for concurrent execution because Python offers a powerful built-in threading module. The downside is that CPython, the Python implementation which is expected to be commonly used for running ASCI, has a Global Interpreter Lock (GIL). The GIL in many ways makes thread programming easier than it is in most languages by making sure that only one thread can manipulate the interpreter's Python objects at a time, but it also implies that on multiprocessor or multi-core systems a program can only make use of one processor at a time, except when the GIL is explicitly released by low-level CPU intensive tasks or blocking I/O operations.

If this proves to be a significant performance bottleneck on systems with multiple processor cores, switching from threads to multiple processes (each one with its independent GIL) should be relatively simple using the *processing* library, that is in many ways a drop-in replacement for the threading modules currently used in ASCI.

Unfortunately the use of a concurrent execution model in turn may introduce race conditions, and this is especially true in a framework such as ASCI, which can potentially have a high number of interacting threads, running code independently written by different people at different times. Most race conditions are avoided, at the cost of introducing potential deadlocks, because every non thread-safe object is either only used by a single thread or passed between different threads using the Python standard library Queue class (a multi-producer, multi-consumer FIFO queue, here always used with a single consumer). To reduce the possibility of deadlocks Queue is the only class in ASCI that uses locks and it is in turn only used in a single module that provides a simple proxy class. This class hides the complexity of the multi-threaded model from the plugin code: it can be used as a wrapper for any Python object and guarantees that all method call of the wrapped object will be executed in the same thread, independently from the calling thread (each thread that requires this kind of interactivity has two queues, for receiving method calls and call results from other threads respectively).

Furthermore if two or more resources may ever need any type of coordinated activity (e.g., a SpaceWire node and a serial port connected to the same device) and can be required by more than one thread (a pretty common situation in ASCI, where scripts and end-users can

concurrently control the hardware), they are always owned by a single thread that coordinates all the accesses to them. This model ensure that the remaining deadlocks arise only when a thread directly or indirectly calls a method in the same thread through the proxy class. These bugs almost always are not time-dependent but occur in a deterministic way, so they are easier to reproduce and fix.

While not technically a deadlock, a thread can also be indefinitely blocked while waiting for the completion of native code, usually I/O operations (e.g. a driver that waits data from an external SpaceWire devices that has been powered off). This class of bugs is largely avoided with the extensive use of relatively short timeouts (in the order of a second) for all low-level I/O operations except file accesses.

Other race conditions (such as livelocks) are still possible, but they are usually the result of bugs in the core code, not in plugins, coherently with the goal of making the creation of new plugins as simple as possible.

5.The STOCC plugin collection

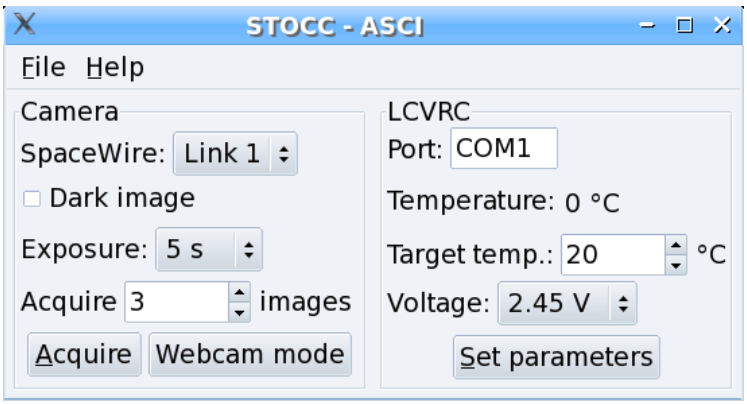
STOCC offers a real-world test for the ASCI architecture and provide two drivers and their corresponding user interfaces for the SCORE project ground tests and calibrations:

1. the coronagraph camera driver, that will control both the Visible Light Detector (VLD) and the UltraViolet Detector (UVD) cameras over a SpaceWire link;
2. the driver for the Liquid Crystal Variable Retarder (LCVR, a polarimeter) controller, which is usually connected to a RS-232 serial port but may also be operated through the cameras SpaceWire link during the last phase of the tests.

5.1.Graphical user interface

The GUI developed for STOCC is relatively simple and is controlled by a main window and any number of image windows that can be opened and closed by the user at any time, independently from each other.

The main window is composed of two main sections: the left one controls the cameras and the right one controls the LCVR controller.



The camera control section require the setting of four parameters before the start of an acquisition:

1. physical link on the USB-SpaceWire Brick to which the camera is connected;
2. optionally the request to acquire a dark image;

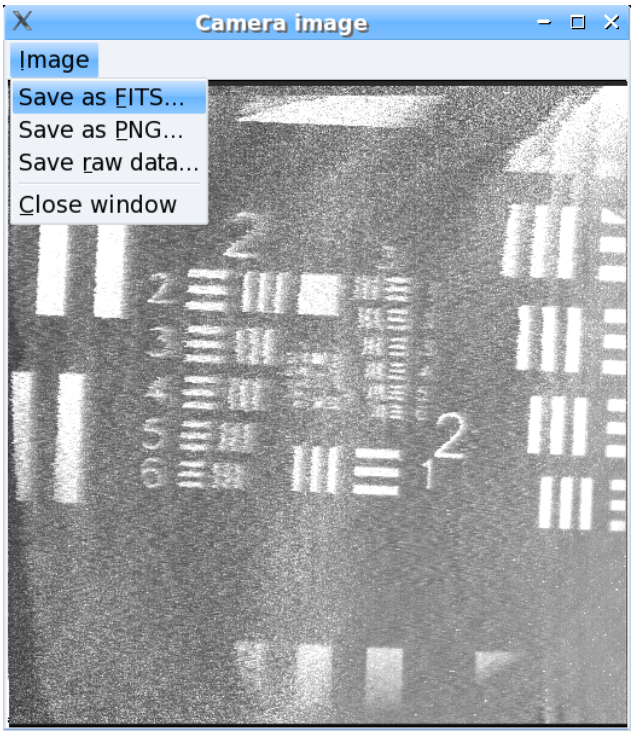
3. exposure time (the current hardware is limited to either 5 or 10 seconds;
4. number of images to acquire in the next sequence.

The LCVRC control section is similar, with three parameters:

1. RS-232 port used;
2. target temperature in degrees Celsius;
3. polarization voltage.

When the parameters are sent to the LCVRC, the Temperature field is updated with the actual value read from the controller.

An image window is used for displaying and converting an image acquired or loaded from file. Any number of image windows can be kept open at the same time, independently by each other.



5.2. Further improvements

STOCC, as of June 2008, is still under active development, mostly to adapt it to the ongoing hardware integration tests and small changes to the communication protocols.

But there are also a couple of more basic improvements that are foreseen in the near future: more extensive usage of parallel processing and controllability through batch files.

While the infrastructure for extensive multi-threading is already present in ASCI, STOCC still doesn't use it to its full potential; e.g. During the acquisition of a new image from a camera the GUI freezes for the duration of the acquisition (usually a few seconds).

Another addition that may be useful in a few circumstances is scriptability; i.e.: it should be possible for the end user to automate series of commands that are usually executed through the GUI in a batch file; the scripting language should be Turing complete, but at

the same time simple enough to write basic automation scripts with minimal training. The batch mode must not run as a separate environment but be tightly integrated with the interactive mode (e.g.: if a script modifies the value of a parameter, the new datum must be immediately visible in the corresponding GUI control; and, vice versa, if the end user modifies at any time, even during the batch commands execution, a parameter in the GUI, the corresponding script variable must be updated right away).

The batch files for ASCI will be written in a slightly modified superset of the Python 2 language. This choice has been driven by two reasons: the simple syntax offered by Python (that makes easier the creation of batch files by scientists without an extended programming background) and the availability by default of a Python compiler in any environment in which ASCI can run.

The only differences between the standard Python 2 and the language used for the batch files are:

- the standard division operator always mean true division (e.g. 1/2 returns 0.5);
- ASCI plugins can offer a number of new application-specific builtin funtions, classes, instances and constants.

Later ASCI versions may switch to Python 3 (currently scheduled for release in September 2008) as the language for batch files, but the changes to existing files will probably be trivial (if necessary at all) and an automatic conversion tool will be used if necessary for the porting.

6.License

ASCI is Free Software (open source) and is currently distributed under the GNU General Public License (version 2 or later). This license has been chosen because it encourages public diffusion of source code for scientific applications, it is compatible with the Python license (version 2.1.1 and newer version) and the licenses of all the libraries used by the core ASCI application, and is required for compatibility with libraries that may be used by specific plugins (e.g. the USPP cross-platform Python serial library is available under the GNU GPL).

Dropping support for version 2 of the license can be evaluated in future releases if the switch to GPL version 3 is useful, but little to no practical consequences are expected.

ASCI plugins (including STOCC) are considered derivative works of the main program, but they can still communicate with external proprietary programs and, if desired, they can be covered by a different GPL-compatible license (e.g. the X11/MIT license) as long as the terms of the GNU GPL are followed when they are distributed. No restrictions at all are imposed on plugins and/or modified ASCI versions that aren't distributed to third-parties.

7.Conclusions

In this short report, the main features of the ASCI system have been presented. In particular, it has been shown how the two-layer architecture gives high flexibility to the software, making it novel with respect to existing similar systems, which are typically bound to specific environments or projects. The ASCI framework instead can be easily re-used for different projects and hardware, with small amounts of reconfiguration work. This result has been obtained by using particular programming techniques, as illustrated in section 4.

8.Appendix A - Source code

8.1.asci.py

```
#!/usr/bin/python

# Copyright (C) 2008  Osservatorio Astronomico di Torino
# Copyright (C) 2008  Lino Mastrodomenico
# Copyright (C) 2006, 2007  Politecnico di Torino
#
# 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; either version 2
# of the License, or (at your option) any later version.
#
# 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.

import wx

from gui.CmdFrame import CmdFrame


if __name__ == '__main__':
    app = wx.PySimpleApp(0)
    wx.InitAllImageHandlers()
    cmd_frame = CmdFrame(None, -1, '')
    app.SetTopWindow(cmd_frame)
    cmd_frame.Show()
    app.MainLoop()
```

8.2.camera/cfgspacewire.py

```
# Copyright (C) 2008  Osservatorio Astronomico di Torino
# Copyright (C) 2008  Lino Mastrodomenico
# Copyright (C) 2006, 2007  Politecnico di Torino
#
# 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; either version 2
# of the License, or (at your option) any later version.
#
# 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.

import ctypes
import os

from star_dundee_types import *
from utils import Namespace, SimpleCLib


if os.name != 'nt':
    libname = 'libConfigLibraryUSB.so'
    api_version = 2.5
else:
    libname = 'RouterConfigLibraryDLL.dll'
    api_version = 2.4  # FIXME: check that everything corresponds to the .h file

lib = SimpleCLib(libname, 'CFGSpaceWire_')

get_API_version = lib.GetAPIVersion(ctypes.c_double)
```

```
assert get_API_version() == api_version  # if this fails recheck everything here

TRANSFER_SUCCESS = 0

BIT0 = 1
BIT2 = 4
BRK_CLK_200_MHZ = BIT2 | BIT0
BRK_DVDR_1 = 0

is_RMAP_enabled = lib.IsRMAPEnabled(ctypes.c_byte)
enable_RMAP = lib.EnableRMAP(None, ctypes.c_byte)
set_as_interface = lib.SetAsInterface(ctypes.c_int, star_device_handle,
                                      ctypes.c_byte, ctypes.c_byte)
set_RMAP_destination_key = lib.SetRMAPDestinationKey(None, U8)
addr_stack_push = lib.AddrStackPush(None, U8)
ret_addr_stack_push = lib.RetAddrStackPush(None, U8)
set_link_speed = lib.SetLinkSpeed(ctypes.c_int, star_device_handle, U32, U32)
set_brick_base_transmit_rate = lib.SetBrickBaseTransmitRate(ctypes.c_int,
                                                            star_device_handle,
                                                            U32, U32, U32)

# router control functions:
rc = Namespace()
get_router_control_register = lib.GetRouterControlRegister(ctypes.c_int,
                                                           star_device_handle,
                                                           ctypes.POINTER(U32))
rc.get_disable_on_silence = lib.RCGetDisableOnSilence(None, U32,
                                                      ctypes.POINTER(ctypes.c_by
te))
rc.get_start_on_request = lib.RCGetStartOnRequest(None, U32,
                                                  ctypes.POINTER(ctypes.c_byte))

# link control functions:
ls = Namespace()
get_link_status_control = lib.GetLinkStatusControl(ctypes.c_int,
                                                   star_device_handle, U32,
                                                   ctypes.POINTER(U32))
set_link_status_control = lib.SetLinkStatusControl(ctypes.c_int,
                                                   star_device_handle, U32,
                                                   U32)
ls.is_auto_start = lib.LSIsAutoStart(None, U32, ctypes.POINTER(ctypes.c_byte))
ls.is_link_running = lib.LSIsLinkRunning(None, U32,
                                         ctypes.POINTER(ctypes.c_byte))
ls.enable_disabled = lib.LSEnableDisabled(None, ctypes.POINTER(U32),
                                          ctypes.c_byte)

# other functions:  (FIXME: what's the exact category?)
start_link = lib.StartLink(ctypes.c_int, star_device_handle, U32)
stop_link = lib.StopLink(ctypes.c_int, star_device_handle, U32)
```

8.3.camera/monitor_parallel.py

```
#!/usr/bin/python

# Copyright (C) 2008  Osservatorio Astronomico di Torino
# Copyright (C) 2008  Lino Mastrodomenico
# Copyright (C) 2006, 2007  Politecnico di Torino
#
# 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; either version 2
# of the License, or (at your option) any later version.
#
# 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.

import os
```

```
import sys
from time import strftime

import parallel

if os.name == 'nt':  # MS Windows
    from parallel.parallelwin32 import _pyparallel
    def get_status(p):
        return _pyparallel.inp(p.statusRegAdr) // 8 & 15
else:
    def get_status(p):
        return p.PPRSTATUS() // 8 & 15  # FIXME: chk if this works!!!


assert len(sys.argv) == 2
p = parallel.Parallel(sys.argv[1])

old_value = None
while True:
    if False:  # FIXME: remove this
        value = p.getInAcknowledge() * 8
        value += p.getInPaperOut() * 4
        value += p.getInSelected() * 2
        value += p.getInError()
    else:
        value = get_status(p)
    if value != old_value:
        print strftime('%H:%M:%S'), value
        old_value = value
```

8.4.camera/receive_images.py

```
#!/usr/bin/python

# Copyright (C) 2008  Osservatorio Astronomico di Torino
# Copyright (C) 2008  Lino Mastrodomenico
# Copyright (C) 2006, 2007  Politecnico di Torino
#
# 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; either version 2
# of the License, or (at your option) any later version.
#
# 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.

import sys

from score_camera import SCORECamera
from star_dundee import SpaceWireDevice


class DumbSaver(object):

    def __init__(self, format):
        self.format = format
        self.n = 0

    def __call__(self, data):
        f = open(self.format % self.n, 'wb')
        f.write(data)
        f.close()
        self.n += 1


def main(argv):
    assert len(argv) == 3  # FIXME
```

```
    link = int(argv[1])  # physical SpaceWire link to use on the router
    num_images = int(argv[2])  # number of images to acquire
    spw = SpaceWireDevice(link, SCORECamera.RECEIVE_BUFFER)
    camera = SCORECamera(spw)
    camera.acquire_images(num_images, DumbSaver('img_%04d.bin'))
    spw.close()


if __name__ == '__main__':
    main(sys.argv)
```

8.5.camera/score_camera.py

```
# Copyright (C) 2008  Osservatorio Astronomico di Torino
# Copyright (C) 2008  Lino Mastrodomenico
# Copyright (C) 2006, 2007  Politecnico di Torino
#
# 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; either version 2
# of the License, or (at your option) any later version.
#
# 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.

import time


class CameraError(Exception): pass


class SCORECamera(object):

    #RECEIVE_BUFFER = 640 * 1024  # 640K ought to be enough for anybody
    RECEIVE_BUFFER = 3 * 1024 * 1024  # apparently it isn't :-(

    PACKET_TIMEOUT = 3  # default timeout in seconds
    IMG_READY_TIMEOUT = 20  # used while waiting for an IMG_READY

    # commands to the camera:
    CAMERA_ON = 1
    RESET = 2
    START_ACQ = 3
    START_He_ACQ = 4
    START_DARK = 5
    START_He_DARK = 6
    STOP_ACQ = 7
    TRANSMIT = 8

    # responses from the camera:
    CAMERA_READY = 1
    ACK_START = 2
    ACK_STOP = 3
    ACK_TRANS = 4
    IMG_READY = 5
    FAILURE = 6

    # relations between commands and responses:
    EXPECTED_RESPONSES = {CAMERA_ON: CAMERA_READY,
                          RESET: CAMERA_READY,
                          START_ACQ: ACK_START,
                          START_He_ACQ: ACK_START,
                          START_DARK: ACK_START,
                          START_He_DARK: ACK_START,
                          STOP_ACQ: ACK_STOP,
                          TRANSMIT: ACK_TRANS}
```

```
    def __init__(self, spacewire_device):
        self.spw = spacewire_device

    def send_command(self, cmd):
        s = '\x54' + chr(cmd * 16)
        self.spw.send_packet(s)
        self.check_response(self.EXPECTED_RESPONSES[cmd])

    def check_response(self, response):
        s = self.spw.receive_packet()
        if len(s) == 1:
            s = '12345678' + s
        if len(s) != 9:
            raise CameraError('wrong response lenght: %d' % len(s))
        n = ord(s[8])
        if n != response:
            raise CameraError('wrong response value: %d (%r)' % (n, s))

    def acquire_images(self, num_images, callbacks, dark=0, uvd=0):
        self.spw.timeout = self.PACKET_TIMEOUT
        try:
            self.send_command(self.CAMERA_ON)
        except Exception:
            pass  # FIXME: probably already initialized
        time.sleep(1)  # requested by Maurizio Pancrazzi
        self.send_command(self.START_ACQ + dark * 2 + uvd)

        for i in range(num_images):
            self.spw.timeout = self.IMG_READY_TIMEOUT
            self.check_response(self.IMG_READY)
            self.spw.timeout = self.PACKET_TIMEOUT  # restore the default
            time.sleep(1)  # requested by Maurizio Pancrazzi
            self.send_command(self.TRANSMIT)
            raw_image_data = self.spw.receive_packet()
            callbacks[i](raw_image_data)

        self.send_command(self.STOP_ACQ)
```

8.6.camera/star_dundee.py

```
# Copyright (C) 2008  Osservatorio Astronomico di Torino
# Copyright (C) 2008  Lino Mastrodomenico
# Copyright (C) 2006, 2007  Politecnico di Torino
#
# 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; either version 2
# of the License, or (at your option) any later version.
#
# 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.

# High level interface

import ctypes

import cfgspacewire
import usbspacewire


class SpaceWireError(Exception): pass


class SpaceWireDevice(object):

    def __init__(self, link, receive_buffer):
        self.port = link
```

```
        self.handle = usbspacewire.star_device_handle()
        if usbspacewire.open(self.handle, 0) != True:  # open the first device
            raise SpaceWireError("couldn't open the device")
        usbspacewire.enable_network_mode(self.handle, False)  # disable network
        if usbspacewire.register_receive_on_port(self.handle, link) != True:
            raise SpaceWireError("couldn't receive on port %d" % link)
        # WARNING: apparently _set_receive_buffer cannot be called after
        # enable_interface_mode! The STAR-Dundee libraries suck!
        self._set_receive_buffer(receive_buffer)
        self.enable_interface_mode()
        self.closed = False

    def close(self):
        if not self.closed:
            self.closed = True
            usbspacewire.unregister_receive_on_port(self.handle, self.port)
            usbspacewire.close(self.handle)

    def is_running(self):
        lsc = cfgspacewire.U32()
        cfgspacewire.get_link_status_control(self.handle, self.port, lsc)
        n = ctypes.c_byte()
        cfgspacewire.ls.is_link_running(lsc, n)
        return n.value != 0

    def start(self):  # calling this isn't usually necessary
        # FIXME: this seems to have a strange side effect: it enables the start
        #     bit (even if no cable is connected) and afterward every time
        #     another device is connected the link goes automatically into the
        #     running state
        cfgspacewire.start_link(self.handle, self.port)

    def stop(self):  # calling this isn't usually necessary
        cfgspacewire.stop_link(self.handle, self.port)
        # stop_link doesn't simply clear the start bit, but sets also the
        # disabled bit (YA Star-Dundee bug?), so reenable it:
        lsc = cfgspacewire.U32()
        cfgspacewire.get_link_status_control(self.handle, self.port, lsc)
        cfgspacewire.ls.enable_disabled(lsc, False)  # disable the disabled bit
        cfgspacewire.set_link_status_control(self.handle, self.port, lsc)

    def _set_timeout(self, seconds):
        usbspacewire.set_timeout(self.handle, seconds)

    def _get_timeout(self):
        return usbspacewire.get_timeout(self.handle)

    timeout = property(_get_timeout, _set_timeout)

    def _set_receive_buffer(self, receive_buffer):
        self.receive_buffer = receive_buffer
        magic = 510  # from the API docs
        usbspacewire.set_driver_read_buffer_size(self.handle, receive_buffer)
        usbspacewire.set_driver_read_structs_num(self.handle,
                                                 receive_buffer // magic + 3)

    def send_packet(self, data):
        packet_id = usbspacewire.ID()
        # send the packet and wait on it completing
        f = open('spw_log.txt', 'a')
        f.write('sending packet: %s\n' % repr(data))
        f.close()
        res = usbspacewire.send_packet_to(self.handle, data, len(data),
                                          chr(self.port), 1, True, packet_id)
        if res != usbspacewire.TRANSFER_SUCCESS:
            raise SpaceWireError("couldn't send the packet")
        f = open('spw_log.txt', 'a')
        f.write('success\n')
        f.close()
        usbspacewire.free_send(self.handle, packet_id)
```

```
    def receive_packet(self):
        packet_id = usbspacewire.ID()
        properties = usbspacewire.PACKET_PROPERTIES()
        buffer = ctypes.create_string_buffer(self.receive_buffer)

        # setup the receive and wait on it completing
        f = open('spw_log.txt', 'a')
        f.write('waiting packet\n')
        f.close()
        res = usbspacewire.read_packets(self.handle, buffer, len(buffer), 1,
                                        True, properties, packet_id)
        if res != usbspacewire.TRANSFER_SUCCESS:
            raise SpaceWireError("couldn't receive the packet")
        data = buffer.raw
        assert ord(data[ 0]) == self.port
        data = data[ 1:properties.len]
        f = open('spw_log.txt', 'a')
        f.write('received packet: %s (len %d)\n' % (repr(data[ :16]), len(data)))
        f.close()
        usbspacewire.free_read(self.handle, packet_id)
        return data

    def enable_interface_mode(self, add_identifier=True):
        if not cfgspacewire.is_RMAP_enabled():
            cfgspacewire.enable_RMAP(True)
            # magic values for the SpaceWire-USB Brick:
            cfgspacewire.set_RMAP_destination_key(0x20)
            cfgspacewire.addr_stack_push(0)
            cfgspacewire.addr_stack_push(254)
            cfgspacewire.ret_addr_stack_push(254)

        if (cfgspacewire.set_as_interface(self.handle, True, add_identifier) !=
            cfgspacewire.TRANSFER_SUCCESS):
            raise SpaceWireError("couldn't set the device to be an interface")
```

8.7.camera/star_dundee_types.py

```
# Copyright (C) 2008  Osservatorio Astronomico di Torino
# Copyright (C) 2008  Lino Mastrodomenico
# Copyright (C) 2006, 2007  Politecnico di Torino
#
# 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; either version 2
# of the License, or (at your option) any later version.
#
# 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.

import ctypes


__all__ = [ 'star_device_handle', 'U8', 'U32']

class star_device_handle(ctypes.c_void_p): pass

U8 = ctypes.c_uint8
U32 = ctypes.c_uint32
```

8.8.camera/sub.py

```
#!/usr/bin/python

# Copyright (C) 2008  Osservatorio Astronomico di Torino
# Copyright (C) 2008  Lino Mastrodomenico
# Copyright (C) 2006, 2007  Politecnico di Torino
```

```
#
# 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; either version 2
# of the License, or (at your option) any later version.
#
# 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.

from __future__ import division

import sys

import Image
import numpy


header_size = 8
w, h = 536, 514
#w, h = 514, 536
# scartare 12 pixel a sx ed a dx e 2 in alto (alla fine dei byte)

def merge_bytes(values):
    assert len(values) % 2 == 0
    new_values = []
    for i in range(0, len(values), 2):
        new_values.append(values[i] + values[i + 1] * 256)
    return new_values


def normalize(values):
    n1 = min(values)  # FIXME!!!
    n2 = max(values)
    print n1, n2
    values = [(n - n1) * 256 // (n2 - n1) for n in values]
    return values


def load_bin(filename):
    f = open(filename)
    s = f.read()
    f.close()
    if len(s) <= 551016:  # assume a binary file
        values = [ord(c) for c in s]
    else:  # probably a text file
        values = s.split()
        values = [int(s) for s in values]
        assert min(values) >= -128
        assert max(values) < 128
        for i, n in enumerate(values):
            if n < 0:
                values[i] = n + 256
    print len(values)
    if 551007 <= len(values) <= 551008:
        values[:0] = range(8)
    if len(values) == 551015:
        values.append(0)
    print values[:12]
    print values[-8:]
    assert min(values) >= 0
    assert max(values) < 256
    values = values[8:]
    values = merge_bytes(values)
    values = numpy.array(values, dtype=numpy.int32)
    values.shape = h, w
    return 65535 - numpy.transpose(values)
```

```
def main(argv):
    assert len(argv) == 3
    f = open(argv[1])
    s = f.read()
    f.close()
    if len(s) <= 551016:  # assume a binary file
        values = [ord(c) for c in s]
    else:  # probably a text file
        values = s.split()
        values = [int(s) for s in values]
        assert min(values) >= -128
        assert max(values) < 128
        for i, n in enumerate(values):
            if n < 0:
                values[i] = n + 256
    print len(values)
    if 551007 <= len(values) <= 551008:
        values[:0] = range(8)
    if len(values) == 551015:
        values.append(0)
    print values[:12]
    print values[-8:]
    assert min(values) >= 0
    assert max(values) < 256
    values = values[8:]
    values = merge_bytes(values)
    print 'len, min, max, average'
    print len(values), 65535 - max(values), 65535 - min(values), 65535 -
int(round(sum(values) / len(values)))
    values = normalize(values)
    im = Image.new('L', (w, h))
    i = 0
    for y in range(h - 1, -1, -1):
        for x in range(w - 1, -1, -1):
            im.putpixel((x, y), values[i])
            i += 1
    assert len(values) == i
    im = im.rotate(270)  # FIXME: this may be wrong
    #im = im.transpose(Image.FLIP_LEFT_RIGHT)
    im = im.point(list(reversed(range(256))))
    im.save(argv[2])


if __name__ == '__main__':
    main(sys.argv)
```

8.9.camera/usbspacewire.py

```
# Copyright (C) 2008  Osservatorio Astronomico di Torino
# Copyright (C) 2008  Lino Mastrodomenico
# Copyright (C) 2006, 2007  Politecnico di Torino
#
# 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; either version 2
# of the License, or (at your option) any later version.
#
# 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.

import ctypes

from star_dundee_types import *
from utils import SimpleCLib


lib = SimpleCLib('libSpaceWireUSBAPI.so', 'USBSpaceWire_')
```

```
get_API_version = lib.GetAPIVersion(ctypes.c_double)
assert get_API_version() == 1.1  # if this fails recheck everything here

class ID(ctypes.c_void_p): pass

EOP_TYPE = ctypes.c_int  # enum
TRAFFIC_TYPE = ctypes.c_int  # enum
STATUS = ctypes.c_int  # enum
TRANSFER_SUCCESS = 2


class PACKET_PROPERTIES(ctypes.Structure):

    _fields_ = [ ('len', ctypes.c_ulong),
                 ('eop', EOP_TYPE),
                 ('type', TRAFFIC_TYPE)]


PPACKET_PROPERTIES = ctypes.POINTER(PACKET_PROPERTIES)

count_devices = lib.CountDevices(U8)
open = lib.Open(ctypes.c_byte, ctypes.POINTER(star_device_handle),
                ctypes.c_int)
close = lib.Close(None, star_device_handle)
enable_network_mode = lib.EnableNetworkMode(None, star_device_handle,
                                            ctypes.c_byte)
register_receive_on_port = lib.RegisterReceiveOnPort(ctypes.c_byte,
                                                     star_device_handle, U8)
unregister_receive_on_port = lib.UnregisterReceiveOnPort(ctypes.c_byte,
                                                         star_device_handle,
                                                         U8)
send_packet_to = lib.SendPacketTo(STATUS, star_device_handle, ctypes.c_void_p,
                                  U32, ctypes.c_void_p, U32, ctypes.c_byte,
                                  ctypes.POINTER(ID))
free_send = lib.FreeSend(ctypes.c_byte, star_device_handle, ID)
read_packets = lib.ReadPackets(STATUS, star_device_handle, ctypes.c_void_p,
                               U32, U32, ctypes.c_byte, PPACKET_PROPERTIES,
                               ctypes.POINTER(ID))
free_read = lib.FreeRead(ctypes.c_byte, star_device_handle, ID)
set_driver_read_buffer_size = lib.SetDriverReadBufferSize(None,
                                                          star_device_handle,
                                                          ctypes.c_ulong)
set_driver_read_structs_num = lib.SetDriverReadStructsNum(None,
                                                          star_device_handle,
                                                          ctypes.c_ulong)

set_timeout = lib.SetTimeout(None, star_device_handle, ctypes.c_double)
get_timeout = lib.GetTimeout(ctypes.c_double, star_device_handle)
```

8.10.camera/utils.py

```
# Copyright (C) 2008  Osservatorio Astronomico di Torino
# Copyright (C) 2008  Lino Mastrodomenico
# Copyright (C) 2006, 2007  Politecnico di Torino
#
# 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; either version 2
# of the License, or (at your option) any later version.
#
# 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.

import ctypes
```

```
class Namespace(object):

    pass  # KISS


class SimpleCLib(object):

    def __init__(self, name, prefix=''):
        self.__lib = ctypes.CDLL(name)
        self.__prefix = prefix

    def __getattr__(self, name):
        c_func = getattr(self.__lib, self.__prefix + name)
        def set_types(restype, *argtypes):
            c_func.restype = restype
            c_func.argtypes = argtypes
            return c_func
        return set_types
```

8.11.gui/AboutDialog.py

```
# Copyright (C) 2008  Osservatorio Astronomico di Torino
# Copyright (C) 2008  Lino Mastrodomenico
# Copyright (C) 2006, 2007  Politecnico di Torino
#
# 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; either version 2
# of the License, or (at your option) any later version.
#
# 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.

import wx

from gui import AboutDialogUI


class AboutDialog(AboutDialogUI):

    def __init__(self, *args, **kwds):
        super(AboutDialog, self).__init__(*args, **kwds)
        self.bind_events()

    def bind_events(self):
        self.close_button.Bind(wx.EVT_BUTTON, self.on_close)

    def on_close(self, event):
        self.Close()
```

8.12.gui/CmdFrame.py

```
# Copyright (C) 2008  Osservatorio Astronomico di Torino
# Copyright (C) 2008  Lino Mastrodomenico
# Copyright (C) 2006, 2007  Politecnico di Torino
#
# 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; either version 2
# of the License, or (at your option) any later version.
#
# 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.

import wx
```

```
import gui
from AboutDialog import AboutDialog
from ImageFrame import ImageFrame
from camera.score_camera import SCORECamera
from camera.star_dundee import SpaceWireDevice
from lcvrc.score_lcvrc import set_parameters


class CmdFrame(gui.CmdFrameUI):

    def __init__(self, *args, **kwargs):
        super(CmdFrame, self).__init__(*args, **kwargs)
        self.about_dialog = AboutDialog(self)  # FIXME: should be created by
wxGlade!
        self.dirname = ''  # FIXME
        self.seq_num = 0
        self.bind_events()
        # FIXME: quick hack:
        self.temperature_text_ctrl.SetBackgroundColour(self.GetBackgroundColour(
))

    def bind_events(self):
        # system events
        self.Bind(wx.EVT_CLOSE, self.on_close)

        # File menu
        #self.Bind(wx.EVT_MENU, self.on_test, self.test_menuitem)
        self.Bind(wx.EVT_MENU, self.on_open_raw, id=gui.OPEN_RAW_ID)
        self.Bind(wx.EVT_MENU, self.on_exit, id=wx.ID_EXIT)
        # Help menu
        self.Bind(wx.EVT_MENU, self.on_about, id=wx.ID_ABOUT)

        # buttons
        #self.Bind(wx.EVT_TOGGLEBUTTON, self.on_polarimeter_status_toggle,
        #          self.acquire_button)
        self.acquire_button.Bind(wx.EVT_BUTTON, self.on_camera_acquire)
        self.set_lcvrc_button.Bind(wx.EVT_BUTTON, self.on_lcvrc_set_params)
        #self.temperature_spin_ctrl.Bind(wx.EVT_SPINCTRL,
        #                                self.on_temperature_change)
        #self.radio_box_linlog.Bind(wx.EVT_RADIOBOX, self.on_scale_change)

##    def on_polarimeter_status_toggle(self, event):
##        self.window_shown[0] = not self.window_shown[0]
##        self.polarimeter_status_dialog.Show(self.window_shown[0])
##    def on_temperature_change(self, event):
##        print self.temperature_spin_ctrl.GetValue()

    def on_open_raw(self, event):
        dlg = wx.FileDialog(self, 'Open raw data', self.dirname, '', 'Raw files
(*.bin)|*.bin', wx.OPEN)
        if dlg.ShowModal() == wx.ID_OK:
            frame = ImageFrame(self)
            frame.Show()
            f = open(dlg.GetPath(), 'rb')
            raw = f.read()
            f.close()
            frame.load_raw_data(raw)
            self.dirname = dlg.GetDirectory()
        dlg.Destroy()

    def on_lcvrc_set_params(self, event):
        port = self.port_text_ctrl.GetValue()
        temperature = self.temperature_spin_ctrl.GetValue()
        voltage = self.voltage_choice.GetSelection()
        voltage = [2.45, 3.05, 4.15, 9.996][voltage]  # FIXME: ugly
        #print 'set', port, voltage, temperature  # FIXME
        actual_temperature = set_parameters(port, temperature, voltage)

        old_value = self.temperature_text_ctrl.GetValue()
```

```
        value = str(actual_temperature) + ' ' + old_value.split()[1]
        self.temperature_text_ctrl.SetValue(value)
        dlg = wx.MessageDialog(self, 'Parameters successfully set', 'LCVRC',
wx.OK)
        dlg.ShowModal()

    def on_camera_acquire(self, event):
        link = self.link_choice.GetSelection() + 1
        dark = self.dark_checkbox.GetValue()
        exposure = self.exposure_choice.GetSelection()
        num_images = self.num_images_spin_ctrl.GetValue()
        #print 'acquire', link, dark, exposure, num_images  # FIXME
        callbacks = []
        for i in range(num_images):
            frame = ImageFrame(self)
            frame.SetTitle(frame.GetTitle() + ' %d' % self.seq_num)
            self.seq_num += 1
            frame.Show()
            if False:  # FIXME
                frame.load_raw_data(open('z.bin').read())
            else:
                callbacks.append(frame.load_raw_data)

        spw = SpaceWireDevice(link, SCORECamera.RECEIVE_BUFFER)
        try:
            camera = SCORECamera(spw)
            camera.acquire_images(num_images, callbacks, dark, exposure)
        finally:
            spw.close()  # always do this, otherwise the drivers may bug out

    def on_about(self, event):
        self.about_dialog.Show()

    def on_close(self, event):
        # FIXME: insert code here if necessary
        self.Destroy()

    def on_exit(self, event):
        self.Close()
```

8.13.gui/gui.py

```
# Copyright (C) 2008  Osservatorio Astronomico di Torino
# Copyright (C) 2008  Lino Mastrodomenico
# Copyright (C) 2006, 2007  Politecnico di Torino
#
# 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; either version 2
# of the License, or (at your option) any later version.
#
# 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.

#!/usr/bin/env python
# -*- coding: utf-8 -*-
# generated by wxGlade 0.6.3 on Tue Jun  3 21:09:37 2008

import wx

# begin wxGlade: extracode
# end wxGlade



class CmdFrameUI(wx.Frame):
    def __init__(self, *args, **kwds):
```

```
        # begin wxGlade: CmdFrameUI.__init__
        kwds["style"] = wx.DEFAULT_FRAME_STYLE
        wx.Frame.__init__(self, *args, **kwds)
        self.sizer_8_staticbox = wx.StaticBox(self, -1, "LCVRC")
        self.sizer_10_staticbox = wx.StaticBox(self, -1, "Camera")

        # Menu Bar
        self.cmd_frame_menubar = wx.MenuBar()
        global OPEN_FITS_ID; OPEN_FITS_ID = wx.NewId()
        global OPEN_RAW_ID; OPEN_RAW_ID = wx.NewId()
        wxglade_tmp_menu = wx.Menu()
        wxglade_tmp_menu.Append(OPEN_FITS_ID, "Open &FITS...", "",
wx.ITEM_NORMAL)
        wxglade_tmp_menu.Append(OPEN_RAW_ID, "Open &raw data...", "",
wx.ITEM_NORMAL)
        wxglade_tmp_menu.AppendSeparator()
        wxglade_tmp_menu.Append(wx.ID_EXIT, "E&xit", "", wx.ITEM_NORMAL)
        self.cmd_frame_menubar.Append(wxglade_tmp_menu, "&File")
        wxglade_tmp_menu = wx.Menu()
        wxglade_tmp_menu.Append(wx.ID_ABOUT, "About...", "", wx.ITEM_NORMAL)
        self.cmd_frame_menubar.Append(wxglade_tmp_menu, "&Help")
        self.SetMenuBar(self.cmd_frame_menubar)
        # Menu Bar end
        self.label_5 = wx.StaticText(self, -1, "SpaceWire:")
        self.link_choice = wx.Choice(self, -1, choices=["Link 1", "Link 2"])
        self.dark_checkbox = wx.CheckBox(self, -1, "Dark image")
        self.label_7 = wx.StaticText(self, -1, "Exposure:")
        self.exposure_choice = wx.Choice(self, -1, choices=["5 s", "10 s"])
        self.label_8 = wx.StaticText(self, -1, "Acquire")
        self.num_images_spin_ctrl = wx.SpinCtrl(self, -1, "3", min=1, max=100)
        self.label_9 = wx.StaticText(self, -1, "images")
        self.acquire_button = wx.Button(self, -1, "&Acquire")
        self.webcam_button = wx.Button(self, -1, "Webcam mode")
        self.label_1 = wx.StaticText(self, -1, "Port:")
        self.port_text_ctrl = wx.TextCtrl(self, -1, "COM1")
        self.label_4 = wx.StaticText(self, -1, "Temperature:")
        self.temperature_text_ctrl = wx.TextCtrl(self, -1, u"0 °C",
style=wx.TE_READONLY|wx.NO_BORDER)
        self.label_2 = wx.StaticText(self, -1, "Target temp.:")
        self.temperature_spin_ctrl = wx.SpinCtrl(self, -1, "20", min=0, max=40)
        self.label_3 = wx.StaticText(self, -1, u"°C")
        self.label_11 = wx.StaticText(self, -1, "Voltage:")
        self.voltage_choice = wx.Choice(self, -1, choices=["2.45 V", "3.05 V",
"4.15 V", "9.996 V"])
        self.set_lcvrc_button = wx.Button(self, -1, "&Set parameters")

        self.__set_properties()
        self.__do_layout()
        # end wxGlade

    def __set_properties(self):
        # begin wxGlade: CmdFrameUI.__set_properties
        self.SetTitle("STOCC - ASCI")
        self.link_choice.SetSelection(0)
        self.exposure_choice.SetSelection(0)
        self.voltage_choice.SetSelection(0)
        # end wxGlade

    def __do_layout(self):
        # begin wxGlade: CmdFrameUI.__do_layout
        sizer_7 = wx.BoxSizer(wx.HORIZONTAL)
        sizer_8 = wx.StaticBoxSizer(self.sizer_8_staticbox, wx.VERTICAL)
        sizer_15 = wx.BoxSizer(wx.HORIZONTAL)
        sizer_6 = wx.BoxSizer(wx.HORIZONTAL)
        sizer_4 = wx.BoxSizer(wx.HORIZONTAL)
        sizer_5 = wx.BoxSizer(wx.HORIZONTAL)
        sizer_9 = wx.BoxSizer(wx.HORIZONTAL)
        sizer_10 = wx.StaticBoxSizer(self.sizer_10_staticbox, wx.VERTICAL)
        sizer_14 = wx.BoxSizer(wx.HORIZONTAL)
        sizer_13 = wx.BoxSizer(wx.HORIZONTAL)
```

```
        sizer_12 = wx.BoxSizer(wx.HORIZONTAL)
        sizer_11 = wx.BoxSizer(wx.HORIZONTAL)
        sizer_11.Add(self.label_5, 0, wx.RIGHT|wx.ALIGN_CENTER_VERTICAL|
wx.ADJUST_MINSIZE, 5)
        sizer_11.Add(self.link_choice, 0, wx.ALIGN_CENTER_VERTICAL|
wx.ADJUST_MINSIZE, 0)
        sizer_10.Add(sizer_11, 1, wx.EXPAND, 0)
        sizer_10.Add(self.dark_checkbox, 0, wx.BOTTOM|wx.ADJUST_MINSIZE, 5)
        sizer_12.Add(self.label_7, 0, wx.RIGHT|wx.ALIGN_CENTER_VERTICAL|
wx.ADJUST_MINSIZE, 5)
        sizer_12.Add(self.exposure_choice, 0, wx.ALIGN_CENTER_VERTICAL|
wx.ADJUST_MINSIZE, 0)
        sizer_10.Add(sizer_12, 1, wx.EXPAND, 0)
        sizer_13.Add(self.label_8, 0, wx.RIGHT|wx.ALIGN_CENTER_VERTICAL|
wx.ADJUST_MINSIZE, 5)
        sizer_13.Add(self.num_images_spin_ctrl, 0, wx.ALIGN_CENTER_VERTICAL|
wx.ADJUST_MINSIZE, 0)
        sizer_13.Add(self.label_9, 0, wx.LEFT|wx.ALIGN_CENTER_VERTICAL|
wx.ADJUST_MINSIZE, 5)
        sizer_10.Add(sizer_13, 1, wx.EXPAND, 0)
        sizer_14.Add(self.acquire_button, 0, wx.RIGHT|wx.ALIGN_CENTER_VERTICAL|
wx.ADJUST_MINSIZE, 5)
        sizer_14.Add(self.webcam_button, 0, wx.ALIGN_CENTER_VERTICAL|
wx.ADJUST_MINSIZE, 0)
        sizer_10.Add(sizer_14, 1, wx.ALIGN_CENTER_HORIZONTAL, 0)
        sizer_7.Add(sizer_10, 1, wx.ALL|wx.EXPAND, 5)
        sizer_9.Add(self.label_1, 0, wx.RIGHT|wx.ALIGN_CENTER_VERTICAL|
wx.ADJUST_MINSIZE, 5)
        sizer_9.Add(self.port_text_ctrl, 0, wx.ADJUST_MINSIZE, 0)
        sizer_8.Add(sizer_9, 1, wx.BOTTOM|wx.EXPAND, 5)
        sizer_5.Add(self.label_4, 0, wx.RIGHT|wx.TOP|wx.EXPAND|
wx.ALIGN_CENTER_VERTICAL|wx.ADJUST_MINSIZE, 5)
        sizer_5.Add(self.temperature_text_ctrl, 0, wx.ALIGN_CENTER_VERTICAL|
wx.ADJUST_MINSIZE, 0)
        sizer_8.Add(sizer_5, 1, wx.EXPAND, 0)
        sizer_4.Add(self.label_2, 0, wx.RIGHT|wx.ALIGN_CENTER_VERTICAL|
wx.ADJUST_MINSIZE, 5)
        sizer_4.Add(self.temperature_spin_ctrl, 0, wx.ALIGN_CENTER_VERTICAL|
wx.ADJUST_MINSIZE, 0)
        sizer_4.Add(self.label_3, 0, wx.LEFT|wx.ALIGN_CENTER_VERTICAL|
wx.ADJUST_MINSIZE, 5)
        sizer_8.Add(sizer_4, 1, wx.EXPAND, 0)
        sizer_6.Add(self.label_11, 0, wx.RIGHT|wx.ALIGN_CENTER_VERTICAL|
wx.ADJUST_MINSIZE, 5)
        sizer_6.Add(self.voltage_choice, 0, wx.ALIGN_CENTER_VERTICAL|
wx.ADJUST_MINSIZE, 0)
        sizer_8.Add(sizer_6, 1, wx.BOTTOM|wx.EXPAND, 5)
        sizer_15.Add(self.set_lcvrc_button, 0, wx.ALIGN_CENTER_VERTICAL|
wx.ADJUST_MINSIZE, 0)
        sizer_8.Add(sizer_15, 1, wx.ALIGN_CENTER_HORIZONTAL|
wx.ALIGN_CENTER_VERTICAL, 0)
        sizer_7.Add(sizer_8, 1, wx.ALL|wx.EXPAND, 5)
        self.SetSizer(sizer_7)
        sizer_7.Fit(self)
        self.Layout()
        # end wxGlade

# end of class CmdFrameUI


class AboutDialogUI(wx.Dialog):
    def __init__(self, *args, **kwds):
        # begin wxGlade: AboutDialogUI.__init__
        kwds["style"] = wx.DEFAULT_DIALOG_STYLE
        wx.Dialog.__init__(self, *args, **kwds)
        self.label_6 = wx.StaticText(self, -1, "ASCI version 0.52")
        self.close_button = wx.Button(self, -1, "&OK")

        self.__set_properties()
        self.__do_layout()
```

```
        # end wxGlade

    def __set_properties(self):
        # begin wxGlade: AboutDialogUI.__set_properties
        self.SetTitle("About ASCI")
        self.close_button.SetFocus()
        self.close_button.SetDefault()
        # end wxGlade

    def __do_layout(self):
        # begin wxGlade: AboutDialogUI.__do_layout
        sizer_1 = wx.BoxSizer(wx.VERTICAL)
        sizer_1.Add(self.label_6, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL|
wx.ADJUST_MINSIZE, 10)
        sizer_1.Add(self.close_button, 0, wx.BOTTOM|wx.ALIGN_CENTER_HORIZONTAL|
wx.ADJUST_MINSIZE, 10)
        self.SetSizer(sizer_1)
        sizer_1.Fit(self)
        self.Layout()
        # end wxGlade

# end of class AboutDialogUI


class ImageFrameUI(wx.Frame):
    def __init__(self, *args, **kwds):
        # begin wxGlade: ImageFrameUI.__init__
        kwds["style"] = wx.DEFAULT_FRAME_STYLE
        wx.Frame.__init__(self, *args, **kwds)

        # Menu Bar
        self.frame_2_menubar = wx.MenuBar()
        global SAVE_FITS_ID; SAVE_FITS_ID = wx.NewId()
        global SAVE_PNG_ID; SAVE_PNG_ID = wx.NewId()
        global SAVE_RAW_ID; SAVE_RAW_ID = wx.NewId()
        wxglade_tmp_menu = wx.Menu()
        wxglade_tmp_menu.Append(SAVE_FITS_ID, "Save as &FITS...", "",
wx.ITEM_NORMAL)
        wxglade_tmp_menu.Append(SAVE_PNG_ID, "Save as &PNG...", "",
wx.ITEM_NORMAL)
        wxglade_tmp_menu.Append(SAVE_RAW_ID, "Save &raw data...", "",
wx.ITEM_NORMAL)
        wxglade_tmp_menu.AppendSeparator()
        wxglade_tmp_menu.Append(wx.ID_EXIT, "&Close window", "", wx.ITEM_NORMAL)
        self.frame_2_menubar.Append(wxglade_tmp_menu, "&Image")
        self.SetMenuBar(self.frame_2_menubar)
        # Menu Bar end
        self.camera_bitmap = wx.StaticBitmap(self, -1, wx.NullBitmap)

        self.__set_properties()
        self.__do_layout()
        # end wxGlade

    def __set_properties(self):
        # begin wxGlade: ImageFrameUI.__set_properties
        self.SetTitle("Camera image")
        self.Hide()
        self.camera_bitmap.SetMinSize((512, 513))
        # end wxGlade

    def __do_layout(self):
        # begin wxGlade: ImageFrameUI.__do_layout
        sizer_3 = wx.BoxSizer(wx.VERTICAL)
        sizer_3.Add(self.camera_bitmap, 0, wx.ADJUST_MINSIZE, 0)
        self.SetSizer(sizer_3)
        sizer_3.Fit(self)
        self.Layout()
        # end wxGlade

# end of class ImageFrameUI
```

```
if __name__ == "__main__":
    app = wx.PySimpleApp(0)
    wx.InitAllImageHandlers()
    cmd_frame = CmdFrameUI(None, -1, "")
    app.SetTopWindow(cmd_frame)
    cmd_frame.Show()
    app.MainLoop()
```

8.14.gui/ImageFrame.py

```
# Copyright (C) 2008  Osservatorio Astronomico di Torino
# Copyright (C) 2008  Lino Mastrodomenico
# Copyright (C) 2006, 2007  Politecnico di Torino
#
# 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; either version 2
# of the License, or (at your option) any later version.
#
# 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.

import os

import numpy
import pyfits
import wx

import gui


W, H = 512, 513
PACKET_SIZE = W * H * 2 + 8 - 1


class ImageFrame(gui.ImageFrameUI):

    def __init__(self, *args, **kwargs):
        super(ImageFrame, self).__init__(*args, **kwargs)
        self.dirname = ''
        self.bind_events()

    def bind_events(self):
        # File menu
        #self.Bind(wx.EVT_CLOSE, self.on_close)
        self.Bind(wx.EVT_MENU, self.on_save_fits, id=gui.SAVE_FITS_ID)
        self.Bind(wx.EVT_MENU, self.on_save_raw, id=gui.SAVE_RAW_ID)
        self.Bind(wx.EVT_MENU, self.on_exit, id=wx.ID_EXIT)

    def on_exit(self, event):
        self.Close()

    def on_save_fits(self, event):
        #print wx.FileDialog(self, wildcard='FITS files (*.fits)|*.fits',
style=wx.wxFD_SAVE).Path
        dlg = wx.FileDialog(self, 'Save as FITS', self.dirname, '', 'FITS files
(*.fits)|*.fits', wx.SAVE)
        if dlg.ShowModal() == wx.ID_OK:
            self.save_fits(dlg.GetPath())
            self.dirname = dlg.GetDirectory()
        dlg.Destroy()

    def on_save_raw(self, event):
        dlg = wx.FileDialog(self, 'Save raw data', self.dirname, '', 'Raw files
(*.bin)|*.bin', wx.SAVE)
```

```
        if dlg.ShowModal() == wx.ID_OK:
            self.save_raw(dlg.GetPath())
            self.dirname = dlg.GetDirectory()
        dlg.Destroy()
        # vld|uvd-080511-130059.raw
        # 20080511

    def load_raw_data(self, raw):
        #print len(raw)
        if len(raw) == 524295 and False:
            raw += '\0' * (551015 - 524295)  # FIXME!!!
        if len(raw) == 551015:
            raw = raw[:PACKET_SIZE]  # FIXME!!!
        #print len(raw)
        self.raw_data = raw
        self.camera_bitmap.SetBitmap(self._raw_data_to_bitmap(raw))

    def save_fits(self, filename):
        s = self.raw_data[8:] + '\0'
        a = 65535 - self.string16_to_array32(s, W, H)
        try:
            os.remove(filename)
        except OSError:
            pass
        fits = pyfits.PrimaryHDU(a)
        header = fits.header
##        header.update('DARK', dark)
##        header.update('EXPTIME', exp_time, 's')
##        header.update('LCVRTEMP', 25, 'degrees Celsius')
##        header.update('LCVRVOLT', lcvr_voltage, 'mV')
##        header.update('PREPOLAR', prepolarization, 'deg')
        #header.update('', , '')
        fits.writeto(filename)

    def save_raw(self, filename):
        f = open(filename, 'wb')
        f.write(self.raw_data)
        f.close()

    def string16_to_array32(self, s, w, h):
        a = numpy.fromstring(s, numpy.uint16)
        a.shape = w, h
        return numpy.array(numpy.transpose(a), numpy.int32)

    def _raw_data_to_bitmap(self, raw):
        assert len(raw) == PACKET_SIZE
        raw = raw[8:] + '\0'  # remove header and fix data size
        a = numpy.fromstring(raw, dtype=numpy.uint16)
        a.shape = W, H
        a = 65535 - a.transpose()[::-1].reshape(W * H)
        a //= 257  # 65535 -> 255
        a = numpy.array(a, dtype=numpy.uint8)
        im = numpy.array([a, a, a]).transpose()  # convert to RGB
        im = wx.ImageFromData(W, H, im.tostring())
        return wx.BitmapFromImage(im)
        #return wx.BitmapFromBuffer(w, h, im.tostring())  # FIXME: wxPy 2.8???
```

8.15.lcvrc/score_lcvrc.py

```
# Copyright (C) 2008  Osservatorio Astronomico di Torino
# Copyright (C) 2008  Lino Mastrodomenico
# Copyright (C) 2006, 2007  Politecnico di Torino
#
# 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; either version 2
# of the License, or (at your option) any later version.
#
# 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.

from __future__ import division

import serial


if False:
    # prototype board
    vp_min = -0.001
    vp_max = 10
    vm_max = -0.004
    vm_min = -10.007
else:
    # flight model
    vp_min = -0.001
    vp_max = 9.996
    vm_max = -0.001
    vm_min = -9.998
v_min = 0.01
v_max = min(vp_max, -vm_min)


def interpolate(x, x1, x2, y1, y2):
    # float linear interpolation
    y = ((x - x1) * (y2 - y1)) / (x2 - x1) + y1
    return int(round(y))


def encode_data(t, v0, v1, v2, v3, texp):
    t = int(round(t * 10))
    voltages = v0, v1, v2, v3
    voltages = [ max(v_min, min(v_max, n)) for n in voltages]
    v = [ interpolate(n, vp_min, vp_max, 0, 4095) for n in voltages]
    v += [ interpolate(n, -vm_min, -vm_max, 0, 4095) for n in voltages]
    assert min(v) >= 0, v
    assert max(v) < 4096, v
    values = [ t]
    for n in v:
        values.append(n % 256)
        values.append(n // 256)
    texp = 65535 - texp * 2000
    values.append(texp % 256)
    values.append(texp // 256)
    return ('$00:%d' + ',%d' * 18 + '\r') % tuple(values)


def set_parameters(port, temperature, voltage):
    ser = serial.Serial(port, 9600, timeout=2)
    ser.flushInput()
    ser.readline()  # discard a possibly incomplete line
    line = ser.readline()
    assert line != ''
    try:
        actual_temperature = int(line) / 10
    except:
        actual_temperature = 0
    v = voltage
    #print encode_data(temperature, v, v, v, v, 3)
    ser.write(encode_data(temperature, v, v, v, v, 3))
    ser.close()
    return actual_temperature
```