The RTS2 Protocol

Petr Kubánek (a,b), Martin Jelínek (b), John French (c), Michal Pouza (d), Stanislav Vítek (b), Alberto J. Castro–Tirado (b) and Victor Reglero (a)

Remote Telescope System 2nd version (RTS2) is an open source project aimed at developing a software environment to control a fully robotic observatory. RTS2 consists of various components, which communicate via an ASCII-based protocol. As the protocol was from the beginning designed as an observatory control system, it provides some unique features, which are hard to find in the other communication systems. These features include advanced synchronization mechanisms and strategies for setting variables. This presentation describes the protocol and its unique features. It also assesses protocol performance, and provides examples how the RTS2 library can be used to quickly build an observatory control system.

Introduction

Remote Telescope System development started in 2001 as a student project at the Charles University in Prague. The goal was to develop a system capable of operating a telescope devoted to follow-up observations of γ-ray bursts (GRBs). This task was later extended to a development of a system for full control of the robotic observatory, with one of the observations targets being a quick follow-up observations of γ-rays in Prague. The goal was to develop a system capable of operating a telescope devoted to follow-up observations of γ-ray bursts (GRBs). This task was later extended to a development of a system for full control of the robotic observatory, with one of the observations targets being a quick follow-up observations of γ-rays in Prague.

The protocol

Protocol uses ASCII sentences sent over TCP/IP. It design was inspired by Simple Mail Transfer Protocol. The sentences contains sentence type and parameters. Protocol allows full two way communication between observatory components.

RTS2 was designed with easy extensibility of the code in mind. Following features allows fast and easy expansion of the code with new drivers and functionalities. Variables which are defined in single file. They are automatically registered in RTS2 file. Flags associated with variable specify when during the exposure cycle the variable value will be recorded.

States variables used for synchronization of operations carried on device. Scripting language which automatically sequence command execution and enable end users to easily carry different observations.

Synchronization functions which allows various synchronization scenarios. Following is step-by-step explanation of scenario used for telescope corrections synchronization:

1. Telescope driver receives correction from image processing process
2. Telescope driver sends to central daemon and all connected devices blocking mask, indicating that it is moving
3. Central daemon distributes moving mask to all connected devices. From now on the system will postpone all commands which required telescope to track the position.
4. Telescope driver sends query to central daemon, asking for its blocking state
5. Central daemon query device to all devices which might block telescope movement
6. Devices respond to central daemon. They previously received new blocking mask from telescope driver, so they will not start new exposure
7. Central daemon collect informations from the devices and send the resulting mask to telescope driver.
8. If telescope can move, algorithm continue with step 11
10. Once the blocking state update is received, algorithm continues with step 8
12. If during moving any new exposure is tried, it is postponed, as system is not ready to receive it.
13. Telescope ends moving, sends blocking state without moving bit set to the central daemon.
14. Central daemon distributes blocking state updates to the device.
15. Devices receives blocking state update. If there is any opened exposure, the exposure is started.

Example

Here is an example code, showing implementation of the simple sensor device, which has one integer and one double variable. The device is connected by serial port, so it provides a command line option to specify which serial port is used, with "/dev/ttyS0" being the default serial connection.

```c
#include ".../rts2devdevice.h"
#include ".../rts2cnnserial.h"

class Rts2DevSensorDummy:

private:
    Rts2ValueInteger *testInt;
    Rts2ValueDouble *testDouble;
    const char *serialDev;
    Rts2CnnSerial *serialConn;

protected:
    virtual int processOption (int in_opt) {
        switch (in_opt) {
        case 'f':
            serialDev = optarg;
            break;
        default:
            return Rts2Device::processOption (in_opt);
        }
        return 0;
    }

    virtual int init () {
        int ret;
        // first call init from parent class, as that will also parse
        // command line option
        ret = Rts2Device::init ();
        if (ret) return -1; // full version will write here to the device
        serialConn = new Rts2CnnSerial (serialDev, this, RS9600, C8, NONE, 40);
        return serialConn->init ();
    }

    virtual int setVal (Rts2Value *val, void *data) {
        if (val == testInt) return 0; // full version will write here to the device
        if (val == testDouble) return 0; // same as above
        return Rts2DevSensorDummy::setValue (val, data);
    }

public:
    Rts2DevSensorDummy (int in_argc, char **in_argv) :
        Rts2Device (in_argc, in_argv, "S1") {
        // createValue (testInt, "TEST_INT", "test integer value", true, RTS2_VERNER_RECORD_CHANGE, 0, false);
        createValue (testDouble, "TEST_DOUBLE", "test double value", true);
        addOption ('f', NULL, 1, "serial port used for device communication");
        serialDev = optarg;
        serialConn = NULL;
    }

    int main (int argc, char **argv) {
        Rts2DevSensorDummy device = Rts2DevSensorDummy (argc, argv);
        return device.run ();
    }
```

Supported devices

RTS2 have drivers for various devices which astronomers uses. Devices are integrated into an observatory environment using communication library which is part of the RTS2. New device drivers can be easily coded. Devices from following manufactures are supported in the current RTS2 code base:

- CCD-detectores
- Andor Technologies, Apogee, Finger Lake Instruments, SAO control board, SBIG, Starlight XPress
- Telescopes
- Losmandy Gemini, Meade LX200 and other LX200 variants, OpenTPL, Paramount by Software Bisque
- Focusers
- Finger Lake Instruments, OpenTPL, Optec, Robofocuser, Finger Lake Instruments
- Filter wheels
- Finger Lake Instruments, Optec, SBIG
- Photomultipliers
- Optec
- Domes
- Own design, OpenTPL
- Sensors and other devices
- Aerotech A3200 motion controller, Cyrocon cryogenic controller. Keylightle picosampermeters, MS257 monochromator, NewPort light source, Phytron stepper motor controller

References

RTS2 SourceForge site – http://rts2.sf.net