VersArray driver Оглавление

0.43

Создано системой Doxygen 1.3.2

Wed Sep 13 02:20:07 2006

Оглавление

Глава 1

VersArray driver Титульная страница

Программа-сервер для работы с ПЗС-камерой PI VersArray. Предназначена для работы с камерой в текстовом режиме как в режиме отладки, так и при научных исследованиях (напрямую, из скриптов с client.tcl, или из программы-графического интерфейса) и позволяет организовать сложные алгоритмы работы загрузкой необходимых параметров камеры и режимов работы.

Данный раздел описывает работу программы **VA**, управляющую непосредственно ПЗС-камерой; программа VAP для управления её блоком затвора и фильтров описана в разделе **VAP: драйвер блока фильтров**.

Программа поддерживает протокол обмена Supervisor по сокетным соединениям с протоколом TCP/IP и имеет номер порта по умолчанию 16100. Также обслуживается консольный ввод с этим же протоколом. Параметр запуска -p (например, ./VA -p 16123) позволяет установить другой номер порта сокета. Также поддерживаются параметры -h для вывода подсказки и -v для вывода названия и версии программы.

Следующие команды обрабатываются:

- INIT, PARK, RUN, STOP, QUIT - стандартно, без параметров
- RUN NEXP=n - запуск серии n экспозиций (вместо одной если без параметра)
- RUN CONT [NEXP=n] - запуск непрерывного режима накопления/считывания
- INIT TEMP=[-]dd.dd - инициализация с заданием точки температуры, градС.
- GET PARAM_*[суффиксы] - считать параметр из контроллера
- GET STATUS IDENT ERMSG FILE TLEFT NLEFT - статус, версия, текст ошибки, имя последнего файла, время до конца текущей экспозиции и число оставшихся экспозиций
- GET ROI BINNING STOPCCS, EXPTIME - установленные ROI, бинирование, выдержка и сигнал останова экспозиции/изменения затвора
- SET PARAM_*=value - установка параметра в контроллере
- SET ROI="x0 y0 x1 y1" BINNING="xb yb" EXPTIME=sec NROI=n - установки 0-го ROI, бинирования и выдержки, явная деактивация с n-го по (MAXNROI-1)-й ROI
- SET ROIi="x0 y0 x1 y1" ROIj=0 - установка i-го и деактивация j-го ROI

- SET STOPCCS=stop_ccs_cam_state EXPMODE=mode - установка сигнала останова экспозиции/изменения затвора, режима старта экспозиции
- SET FITSkeyt=strvalue FITSkeyt="strvalue//comment" - установка добавочного поля для записи в заголовки выходных FITS-файлов
- SET CHECKSTATUS=OFF - отмена проверки статуса перед исполнением команды

Команда SET ROI0=... и SET ROI=... эквивалентны. j не может превышать значения **VA::MAXNROI**-1. Помните также, что **порядок обработки параметров** (например, в приведенных командах SET) **не определен!**

В этой версии программы возможны прямая активация и деактивация любого номера ROI вне зависимости от активности предыдущих номеров. Если ранее для удаления j-го ROI приходилось сдвигать все остальные (присваивать по цепочке с j+2-й j+1-му и т.д.), то теперь достаточно сказать SET ROIj=0. Команда SET NROI=i по прежнему позволяет деактивировать все области начиная с i-й; SET NROI=0 поставит одну область по умолчанию (во всю матрицу).

Установки SET FITS... позволяют добавить параметры любого типа в заголовок выходного FITS-файла и не влияют на работу камеры. Здесь FITS - зарезервированная приставка, key - название ключа, a t - символ типа ключа (I - целый, L - логический, F - вещественный, S - строковый, любой другой - комментарий при условии, что key=="comment" или key=="history"); a strvalue - строковое представление значения ключа, преобразовываемое в нужный тип Fits::key в момент записи заголовка. Эти пары keyt,strvalue будут храниться в программе с момента INIT до PARK. Значение зарезервировано для удаления ключа из массива. Например:

SET FITSRAF="21.4567/Right ascension"

добавит ключ "RA = 21.4567 / Right ascension " в заголовок в момент записи файла, а

SET FITSRAF=

удалит его из коллекции записываемых дополнительных ключей. На каждый ввод параметра типа комментария в заголовке будет выделено не менее одной строки.

Примеры работы (идентификаторы команд обязательны, но частью опущены):

1. Инициализация камеры на температуру охлаждения в -35.5 градусов:

```
INIT TEMP=-35.5
```

и подождать, пока

`GET PARAM_TEMP` не вернет `OK PARAM_TEMP=-3550` (температура в PVCAM исчисляется в сотых градуса).

2. Изменение установочной температуры:

```
SET PARAM_TEMP_SETPOINT=-4000
```

и опять дождаться устаканивания температуры.

3. Установка двух регионов оцифровки и 2-кратного бинирования, сброс 0-го ROI и проверка:

```
SET ROI1="0 0 49 99" ROI3="200 20 499 119" BINNING="2 2" ROI=0
```

```
GET ROI -> OK ROI="#1: 0 0 49 99 #3: 199 19 499 119"
```

4. Установка малошумного порта с медленным чтением и подробного GAIN:

```
SET PARAM_SPDTAB_INDEX=2 PARAM_GAIN_INDEX=2
```

4. Запуск пяти экспозиций, слежение и чтение имени результата:

```
1 SET EXPTIME=12.34 -> 1 OK

2 RUN NEXP=5 -> 2 OK WAIT=12

3 GET STATUS -> 3 OK STATUS=BUSY

4 GET TLEFT -> OK TLEFT=7.87

...

<- 2 OK STATUS=READY

9 GET STATUS -> 9 OK STATUS=READY

GET FILE -> OK FILE=050601_0123.fits
```

То есть можно ожидать ответа OK STATUS=READY на команду запуска, а можно "долбить" статус. Альтернативно - можно прервать экспозицию (без записи файла):

```
7 STOP -> 7 OK STATUS=READY

<- 2 OK STATUS=READY
```

5. Запуск режима непрерывного чтения до 50 кадров без закрытия затвора и закрытие затвора в обход проблемы с PRE_SEQUENCE-режима:

```
SET PARAM_SHTR_OPEN_MODE=2 STOPCCS=2 PARAM_CUSTOM_CHIP=1 PARAM_PAR_SIZE=40
STOPCCS=2

SET ROI="0 0 1339 39" EXPTIME=0.1

RUN NEXP=50 CONT

GET FILE...

GET TLEFT NLEFT

GET FILE... (серия закончилась)

STOP

RUN

STOP
```

Последние три команды являются "магией". Последний `STOP` подается сразу после `RUN` и срывает ненужную экспозицию, закрывая затвор. После этого снова требуется команда `SET PARAM_SHTR_OPEN_MODE=2` для следующей серии.

6. Возврат к нормальному режиму:

```
SET PARAM_PAR_SIZE=1300 PARAM_CUSTOM_CHIP=0 ROI="0 0 1339 1299" NROI=1
```

7. Установка режима экспозиции для работы по внешнему триггеру

```
SET EXPMODE=1
```

8. Разбор ошибок:

```
GET ERMSG -> OK ERMSG="сообщение" или "код_ошибки расшифровка"
```

Если при ошибке возвращается статус `ERSYN`, то это ошибка набора команды или параметра. Если `ERPAR` - то данный параметр недопустим или текущая комбинация параметров не годится для запуска экспозиции (в случае `RUN -> ERROR STATUS=ERPAR`). Что сделано неправильно - возвращается в ERMSG. Статусы ERSYN и ERPAR временные и не возвращаются `GET STATUS`. Код_ошибки имеет вид `Ci_BRIEF_REASON`, он предваряет текстовый вариант ошибки. Например:`ERMSG="CO_CAM_NAME_NOT_FOUND This is not a valid name for opening the camera"` - вариант попытки инициализации на компьютере без уста-

новленного драйвера и устройства платы обмена, а `ERMSG="CO_CNTRL_CREATE_FAILED Could not create controller object for camera"` - вариант запуска без установленного драйвера платы связи (делается администратором как `"#insmod -f pipci"` если драйвер инсталлирован в системе).

Если при ошибке возвращается статус `ERFAT`, то это неудача вызова функции библиотеки PVCAM и чаще всего свидетельствует о проблеме в программе или (хуже) - в камере.

Более подробно о правилах работы и особенностях процессов смотри в VersArray User Manual и PVCAM Manual.

Замечания по структуре программы.

Основной модуль программы (main) содержит в себе цикл опроса сервера (обслуживающего сокетные соединения и консоль) и вызов обработчика поступивших команд, а также вывод протокола обмена командами и ответами на консоль. Обработчик команд, находящийся здесь же, разбирает команды и, в зависимости от ключевых слов команд и параметров, вызывает соответствующие процедуры из пространства **VA**. По результатам их выполнения вызывается **SVServ::acknow()** с соответствующим признаком успеха. Кроме того, в структуру полученного задания могут добавляться результирующие значения запрошенных у программы параметров.

Обработчик вызывает следующие процедуры **VA** по поступившим командам (i - целый параметр, d - вещественный, [...]-необязательная часть):

- `INIT TEMP=d` : init(d)
- `PARK, QUIT` : park()
- `GET PARAM_pname[_suf]` : getparam(PARAM_pname_suf)
- `GET ERMSG` : get_error_msg()
- `GET FILE` : get_last_frame()
- `GET ROI` : get_roi()
- `GET BINNING` : get_binning()
- `GET EXPTIME` : get_exptimei()
- `GET STOPCCS` : get_stop_cam_state()
- `SET ROI[i]="i1 i2 i3 i4"` : set_roi(i1,i2,i3,i4,i)
- `SET ROI[i]=0` : unset_roi(i)
- `SET PARAM_pname=val` : setparam(PARAM_pname,val)
- `SET NROI` : set_nroi()
- `SET STOPCCS=i` : set_stop_cam_state(i)
- `SET EXPMODE=i` : set_exp_mode(i)
- `SET EXPTIME=d` : set_exptime(d)
- `SET BINNING="i1 i2"` : set_binning(i1,i2)
- `SET FITSkeyt="val[//com]"` : add_fits_key(keyt,val//com)
- `RUN [NEXP=i]` : run_stdacq(i)

- `RUN CONT [NEXP=i]` : run_cont(i)
- `STOP` : stop()

Про содержание упомянутых процедур смотрите их описание в пространстве **VA**. Работа с сервером описана в **SVServ**.

Также программа, кроме модулей va и svserv, использует следующие:

- svcons: для консольного ввода команд через FIFO в программу
- fits: для сохранения изображений в FITS-формате (C)M.Kornilov
- frame: для создания уникальных имен файлов (C)V.Kornilov
- configs: для разбора команд на слова (C)V.Kornilov
- astrotime: для точной фиксации времени начала экспозиции (C)V.Kornilov

Эти модули в настоящем описании не документированы, так как являются универсальными и используемыми в других программах.

Глава 2

VersArray driver Алфавитный указатель пространства имен

2.1 VersArray driver Пространства имен

Полный список документированных пространств имен.

Глава 3

VersArray driver Алфавитный указатель классов

3.1 VersArray driver Классы

Классы с их кратким описанием.

Глава 4

VersArray driver Алфавитный указатель тематических описаний

4.1 VersArray driver Описания

Полный список дополнительных описаний.

Глава 5

VersArray driver Пространства имен

5.1 Пространство имен SVCons

Функции

- bool **consoleInitialization** (const char *fifo_name, int *fifo_desc)
- void * **console** (void *)

5.1.1 Подробное описание

Набор независимых процедур для посылки команд программе с консоли через FIFO.

5.1.2 Функции

5.1.2.1 void* console (void *)

Опрос консоли на ввод команды и запись ее в CON_FIFO.

Вводимая с консоли строка снабжается автоматически инкрементируемым номером команды и посылается в CON_FIFO. Длина вводимой команды ограничена примерно 10 строками (790 символов, которые после добавления номера команды вырастают до 796).

Производится в бесконечном цикле в отдельном потоке.

5.1.2.2 bool consoleInitialization (const char * *fifo_name*, int * *fifo_desc*)

Запуск нового потока Console для выполнения процедуры console

Аргументы:

fifo_name имя файла типа ФИФО для обмена консоли с программой

fifo_desc возвращаемый дескриптор открытого ФИФО

Возвращает:

Успех открытия файла

Заметки:

Ошибка чаще всего возникает в связи с отсутствием файла CON_FIFO (сообщение ": No such file or directory".)

5.2 Пространство имен VA

Функции

- std::string **get_error_msg** ()

  сообщение о деталях ошибочной ситуации

- std::string **get_last_frame** ()
- int **init** (double temp=-30.00)
- int **park** (void)
- int **setparam** (const std::string parname, const std::string strvalue)
- int **getparam** (const std::string parname, std::string *strvalue)
- double **run_stdacq** (int nexp=1)
- double **run_cont** (int nframe=2)
- void **stop** (void)
- void **set_exptime** (double time)
- int **set_binning** (int hbin, int vbin)
- int **set_roi** (int xmin, int ymin, int xmax, int ymax, int iroi=0)
- int **unset_roi** (int iroi)
- int **add_fits_key** (const std::string key, const std::string valcom)
- int **set_nroi** (int n)
- std::string **get_roi** (void)
- std::string **get_binning** (void)
- double **get_exptime** (void)
- int **set_stop_cam_state** (int camstate)
- int **get_stop_cam_state** (void)
- int **set_exp_mode** (int mode)
- double **get_time_left** (void)
- int **get_nexp_left** (void)

Переменные

- const int **MAXNROI** = 10

  Максимальное число областей кадра.

5.2.1 Подробное описание

В данном пространстве имен собраны процедуры для инициализации, парковки, установки и считывания параметров и выполнения стандартной экспозиции с ПЗС-системой VersArray с помощью библиотеки PVCAM. Входная информация в процедуры во многом представляется в текстовом виде, что ориентировано на использование в интерактивном режиме с помощью протокола Supervisor. Многие процедуры переформатированы из примеров программ GetParam_GetEnum.c, SetParam.c, StandardAcq.c, поставляемых вместе с библиотекой работы с камерой PI VersArray PVCAM.

5.2.2 Функции

5.2.2.1 int add_fits_key (const std::string *key*, const std::string *valcom*)

Вставка или удаление ключа в массив добавочных ключей FITS-заголовка

Аргументы:

key Название ключа плюс символ типа ключа

valcom Значение ключа и, возможно, комментарий к нему

Возвращает:

1: успех 0: ошибка

Параметр *key* несет двойную функцию - определяет название ключа и его тип. Тип определяется последним символом в *key* и может быть одним из следующих:

- 'I' - целый
- 'F' - вещественный
- 'L' - логический
- 'S' - строковый
- '?' - комментарий (любой из неперечисленных выше символов)

То, что обозначение типа комментария может быть любым символом, позволяет определять несколько комментариев в заголовке. То есть любой нераспознанный тип отождествляется как комментарий если название ключа совпадает со словами "COMMENT" или "HISTORY" (если нет - возвращается 0).

Последовательность появления комментариев определяется алфавитным порядком символов типа. Пример записи комментария из двух строк:

add_fits_key("COMMENT1"," this is the first line of comment,...");

add_fits_key("COMMENT2"," ... and the second one");

Значение ключа *valcom* дешифруется в переменную заданного типа (см. выше) непосредственно перед записью и НЕ проверяется на предмет принадлежности множеству соответствующих значений. В строке *valcom* последовательность символов "//" распознается как начало текста комментария, удаляется из неё и оставшаяся часть *valcom* при записи ключа становится комментарием. Например:

add_fits_key("RAF", "12.3456// Right Ascension in degrees");

Функция возвращает 1, если пара ключ_тип - значение была успешно помещена в ассоциативный массив. Ошибка возникает, если ключ *key* при value=="" не найден, или если при вставке нового ключа его тип (последний символ *key*) не оказался одним из символов 'F','I','L','S' при остальной части ключа, не являющейся строкой "COMMENT" или "HISTORY" (т.е. вставить "COMMENTA" можно, а "ANOTHERA" нельзя).

Заметки:

Поскольку тип является частью имени ключа до момента записи заголовка, то, в принципе, можно создать несколько ключей разного типа и одинаковым именем. Однако, за исключением описанного случая COMMENT, это запрещено стандартом FITS и делать этого, соответственно, не стоит.

5.2.2.2 std::string get_binning (void)

Получить в строковом представлении установленное бинирование

Возвращает:
Строка "xbin ybin"

5.2.2.3 std::string get_error_msg (void)

сообщение о деталях ошибочной ситуации

Вернуть смысл последней произошедшей ошибки

Возвращает:
Текстовое содержание последней ошибки устройств фотометра

Если ошибок не было, возвращаемая строка содержит текст "No error".

5.2.2.4 double get_exptime (void)

Получить текущую установленную экспозицию в секундах

5.2.2.5 std::string get_last_frame ()

имя последнего записанного файла изображения

Заметки:
Для экспозиции с несколькими ROI через пробел перечисляются имена всех NROI файлов.

5.2.2.6 int get_nexp_left (void)

Вернуть число оставшихся экспозиций в запущенной серии

Возвращает:
Число оставшихся экспозиций считая текущую

5.2.2.7 std::string get_roi (void)

Получить в строковом представлении все установленные активные области

Возвращает:
Строка "#0 xmin0 ymin0 xmax0 ymax0 ...#(nroi-1) xmin(nroi-1)..."

5.2.2.8 int get_stop_cam_state (void)

Вернуть текущее состояние камеры после abort

5.2.2.9 double get_time_left (void)

Вернуть оставшееся до конца экспозиции время

Возвращает:
Время до конца экспозиции в секундах или 0 если экспозиция уже прошла

Заметки:
Возвращаемое во время текущей серии время 0 свидетельствует о стадии считывания или записи в файл. Если и get_nexp_left и get_time_left возвращают 0, но статус еще BUSY, то это стадия записи последнего изображения в файл.

5.2.2.10 int getparam (const std::string *parname*, std::string * *strvalue*)

Считать параметр из ПЗС-системы

Аргументы:
parname имя параметра (обычно с префиксом PARAM_)

strvalue строковое представление параметра

Возвращает:
флаг считывания параметра (1: считан, 0: не считан, -1: не существует)

У ПЗС-системы запрашивается существование, доступ на чтение и тип параметра. Если чтение возможно, то происодится чтение значения из ПЗС-системы и преобразование его в строковое представление *strvalue*. Причина неудачи чтения параметра (если таковой существует в системе) сохраняется в VA::error_msg в случае возврата 0.

Если имя параметра имеет суффикс _DEF, _MIN, _MAX, _INC, то вместо текущего значения параметра считывается, соответственно, значение по умолчанию, минимальное, максимальное значение, или инкремент значения. Права доступа, наличие параметра и тип запрашиваются с суффиксами _ACC, _AVA и _TYP, соответственно. Имен самих параметров, оканчивающихся на такие суффиксы, в PVCAM не существует. Возвращаемые права доступа могут быть одним из следующих значений: "ERR", "RO", "RW", "CHK", "WO"; а типы параметров - одним из "INT8", "INT16", "INT32", "FLT64", "UNS8", "UNS16", "UNS32", "UNS64", "ENUM", "BOOL" и "CHAR_P".

Специальный суффикс _ENU предназначен для возврата всех значений типа TYPE_ENUM вместе с сопровождающими их строковыми представлениями. В возвращаемом значении параметра будет перечень значений типа ";<enum1>:<enum1_text>;<enum2>:<enum2_text>...". Например, для параметра PARAM_EDGE_TRIGGER_ENU будет возвращено ";2:+ edge;3:-edge".

Значения логического типа возвращаются как "1" или "0", остальные в обычном представлении.

5.2.2.11 int init (double *temp* = -30.00)

Инициализировать ПЗС-систему

Аргументы:
temp Значение точки установки температуры

Возвращает:
1: инициализация успешна 0: ошибка инициализации

Инициализируется библиотека PVCAM, связь с камерой и проводится диагностика. Проверяется наличие файла va_cntrl_file. Выставляется один активный ROI размером в полный кадр ПЗС и установочная температура охлаждения кристалла. При этом и начинается охлаждение. В конце инициализируется имя файла следующего кадра текущей датой. Открытая камера (0-я) при этом имеет установки по умолчанию, не зависимо от того, что было установлено для нее в предыдущие команды до последних команд PARK и INIT.

Текстовые сообщения при ошибке помещаются в строке VA::error_msg, которая читается с помощью **get_error_msg()**. Начинать экспозицию можно после того, как температура камеры (возвращаемая getparam("PARAM_TEMP")) достигнет точки установки и пройдет еще минимум 1 минута (полная стабильность достигается через 20 минут после достижения точки установки). Поэтому тот факт, что процедура возвращает управление фактически сразу после вызова (установки параметров и подготовительные операции в программе занимают мало времени) не означает, что инициализация системы действительно проведена в полной мере - надо выждать дополнительное время, хотя за этим может проследить и пользователь - оператор системы.

Заметки:
В настоящее время по невыясненной причине библиотека PVCAM не поддерживает повторную инициализация, поэтому вызов pl_pvcam_init() делается только в первый вызов INIT-процедуры, а вызов pl_pvcam_uninit() вынесен из PARK-процедуры в atexit()-вызов. Текущая версия тестированной библиотеки PVCAM - 2.6.5, драйвера - 400 (под ядра 2.4.7-10).

5.2.2.12 int park (void)

Запарковать ПЗС-систему

Возвращает:
1: парковка успешна 0: ошибка инициализации

Связь с камерой закрывается, камера в состоянии ожидания. Однако, если затвор был открыт, он не закроется до следующего **init()** и **run_stdacq()** или **run_cont()**.

Заметки:
Текстовое сообщение при ошибке помещается в строке VA::error_msg.

См. также:
init()

5.2.2.13 double run_cont (int *nframe* = 2)

Запустить накопление в режиме непрерывной работы

Аргументы:
nframe число накапливаемых до останова кадров с вертикальным размером матрицы

Возвращает:
nexp полное время выполнения в секундах, 0 при несоответствии текущих параметров системы, -1 при ошибке работы системы

Режим непрерывной работы - экспозиция,считывание,экспозиция, считывание,... и т.д. Считывание данных ведётся в кольцевой буфер. Запись файла производится по накоплении такого количества строк данных, которое ещё не превышает реального вертикального размера приёмника, хранящегося в параметре PARAM_PAR_SIZE_DEF. Режим Track Delay Integration (TDI) является частным случаем такой работы, для чего требуется выключить очистку перед экспозицией (PARAM_CLEAR_CYCLES=0), выставить PARAM_CUSTOM_CHIP=1, PARAM_PAR_SIZE=1 и ROI="0 0 xmax-1 0". Таким образом, в каждом выходном файле сохраняется PARAM_PAR_SIZE_DEF / PARAM_PAR_SIZE экспозиций и таких файлов будет накоплено *nframe* штук, если накопление не будет прервано **stop()**.

Вообще, назначение этой процедуры - быстрое синхронное накопление в режиме изменённого логического вертикального размера приёмника. Между экспозициями есть только пауза для считывания изображения, поэтому они отстоят друг от друга на равные промежутки времени. Открытие затвора (в начале серии, каждой экспозиции или вообще без работы затвора) должно регулироваться установкой параметров PARAM_SHTR_OPEN_MODE и вызовами **set_stop_cam_state()**.

Длина кольцевого буфера определяется из отношения оценки времени записи одного кадра ко времени экспозиции, с запасом в 3 экспозиции. Считанные экспозиции сохраняются пристыкованными друг к другу по вертикали ("в столбик"). Кроме того, если определено несколько ROI, после каждой экспозиции происходит дешифровка (самого старого) элемента кольцевого буфера в очередные элементы буферов ROI таким образом, что к моменту накопления необходимого для записи файла количества данных в каждом буфере ROI сохранены "в столбик" соответствующие экспозиции только этого ROI.

Накопление отдельных экспозиций производится в отдельном потоке. Процедура возвращает управление после запуска экспозиции и потока считывания. Установка длительности экспозиции одного накопления (для TDI

- одной строчки), бинирования и ROI производятся функциями **set_exptime()**, **set_binning()**, **set_roi()**.

Число кадров *nframe* имеет смысл ставить более 1, так в первом кадре строки экспонируются разное время для разных строк и он обычно не анализируется.

Длительность элементарной экспозиции задается через команду SET EXPTIME=t_line, но реальный темп экспозиций складывается из t_line и временем считывания, полученным через запрос PARAM_READOUT_TIME. В выходной файл записывается время экспозиции EXPTIME∗NINTEGR, где NINTEGR - число накоплений в одном файле (для TDI - число строк), также записываемое в одноименном поле заголовка.

Процедура не-реентерабельная, защищена от повторного входа до окончания экспозиции мутексом (На верхнем логическом уровне эта защита дублирована в классе **SVServ**, запрещающем запуск команды RUN при статусе программы BUSY). В случае более чем одного активного ROI, их имена добавляются в last_frame через пробел. Кроме того, в текущей директории постоянно поддерживается ссылка last.fits на последний записанный файл. Имя файла копируется также в переменную last_frame. В конце работы она запускает внешнюю функцию va_acknow_exp(success) (см. **run_stdacq()**), чтобы сигнализировать пользователю об окончании работы режима непрерывного накопления.

Заметки:

В настоящее время (версия программы 0.4, версия ядра 2.4.20-4GB, драйвер pipci.o под 2.4.4 (0x400) и библиотека PVCAM 0x263) вызов pl_exp_unravel() останавливает работу камеры по заполнению кольцевого буфера, поэтому возможно получение лишь одной группы файлов (NROI штук) для NROI>1.

5.2.2.14 double run_stdacq (int *nexp* = 1)

Запустить одну или несколько экспозиций и считывание изображений в стандартном режиме

Аргументы:

nexp число экспозиций

Возвращает:

полное время выполнения в секундах, 0 при несоответствии текущих параметров системы, -1 при ошибке работы системы

Режим классической экспозиции: запуск накопления, ожидание окончания и считывания, запись файла - все в цикле 1..*nexp*. Экспозиции в контроллере запускаются по-отдельности, то есть с длиной серии в 1, и поэтому не синхронизованы по времени. Режим экспозиции (затвора) меняется функцией **set_exp_mode()** и по умолчанию установлен в 0 (TIMED_MODE). Ожидание конца экспозиций и считывание результатов производится в отдельном потоке. Процедура возвращает управление после запуска экспозиции и потока ожидания конца ее считывания. Установка длительности экспозиции, бинирования и ROI производятся функциями **set_exptime()**, **set_binning()**, **set_roi()**.

Процедура не-реентерабельная, защищена от повторного входа до окончания экспозиции мутексом (На верхнем логическом уровне эта защита дублирована в классе **SVServ**, запрещающем запуск команды RUN при статусе программы BUSY.) Процедура ожидания конца экспозиции, запускаемая в отдельном потоке после ее старта, считывает и записывает кадр в файл, имя которого копируется также в переменную last_frame. В случае более чем одного активного ROI, их имена добавляются в last_frame через пробел. Кроме того, в текущей директории постоянно поддерживается ссылка last.fits на последний записанный файл. По окончании серии экспозиций запускается внешняя функция va_acknow_exp(success), которую должен определить пользователь. Эта процедура дает знать верхнему логическому уровню, что экспозиция закончена и система свободна для начала новой. В программе, управляемой протоколом типа Supervisor, эта функция установит необходимый статус в соответствии с успешностью окончания считывания и записи изображений и разрешит, таким образом, вызвать **run_stdacq()** снова.

При определении в системе нескольких активных ROI (был вызов **set_roi()** с последним параметром >0 и/или set_nroi(n) с n>1) по окончании считывания поток пикселей дешифруется с помощью pl_exp_unravel() в буфера ROI и каждый буфер записывается в отдельный файл. При этом содержание заголовков этих файлов будет идентичным за исключением параметров NAXIS1, NAXIS2 (размерности области) и CRVAL1, CRVAL2 (координаты ее угла, прилежащего к началу координат ПЗС, начинающиеся с 1,1). Имена файлов отличаются так же, как и отличаются файлы разных экспозиций.

5.2.2.15 int set_binning (int *hbin*, int *vbin*)

Установить бинирование

Аргументы:

hbin фактор бинирования по X (1,2,3...)

vbin фактор бинирования по Y (1,2,3...)

Возвращает:

1: бинирование установлено 0: бинирование несовместимо с каким-либо из активных ROI

Процедура обновляет текущие размеры матрицы (если был вызов **setparam()**) и проверяет допустимость параметров всех активных ROI с новыми факторами бинирования hbin, vbin. Если все ROI проходят проверку, у них выставляются эти факторы.

Заметки:
Факторы бинирования одинаковы для всех ROI.

5.2.2.16 int set_exp_mode (int *mode*)

Установить режим экспозиции EXPMODE

Аргументы:
mode Значение перечисления *_MODE

Возвращает:
1: значение допустимо и установлено 0: значение недопустимо

Устанавливаемое целое значение является аргументом *mode* функций pl_exp_setup_cont() и pl_exp_setup_seq().

Для простоты приведем значения: 0=TIMED_MODE, 1=STROBED_MODE, 2=BULB_MODE, 3=TRIGGER_FIRST_MODE, 4=FLASH_MODE, 5=VARIABLE_TIMED_MODE, 6=INT_STROBE_MODE. Не все эти значения допустимы! Допустимые возвращаются командой getparam(PARAM_EXPOSURE_MODE_ENU), а текущее установленное - командой getparam(PARAM_EXPOSURE_MODE).

5.2.2.17 void set_exptime (double *time*)

Установить длительность экспозиции в текущих единицах

Аргументы:
time длительность экспозиции (ms, us или sec)

Установка нулевой экспозиции допустима. Отрицательные рассматриваются как нулевые.

Заметки:
Текущие единицы длины экспозиции устанавливаются параметром PARAM_EXP_RES_INDEX.

5.2.2.18 int set_nroi (int *n*)

Установить число активных областей считывания

Аргументы:
n Новое число областей (1..MAXNROI)

Возвращает:
1: установлено 0: не установлено

Функция имеет смысл для изменения числа областей, которые уже были ранее инициализированы **set_roi()**. Перед изменением числа активных областей проводится проверка соответствия всех ROI в диапазоне индексов 0..n-1 текущему бинированию.

5.2.2.19 int set_roi (int *xmin*, int *ymin*, int *xmax*, int *ymax*, int *iroi* = 0)

Установить область оцифровки (ROI - Region Of Iinterest)

Аргументы:

xmin левый край области (первый пиксел = 0!)

ymin нижний край области (первый пиксел = 0!)

xmax правый край

ymax верхний край

iroi номер области (в диапазоне 0..MAXNROI-1)

Возвращает:

1: область установлена, 0: параметры ошибочны

Производится проверка допустимости границ региона при текущем бинировании и размере считываемой области ПЗС и принятие их в качестве текущих, если проверка успешная.

Число активных областей автоматически увеличивается, если ранее iroi-я область не была активна.

Заметки:

Текущие параметры бинирования одинаковы для всех областей! Проверки пересечения областей не производится!

5.2.2.20 int set_stop_cam_state (int *camstate*)

Установить состояние камеры после экспозиции или STOP

Аргументы:

camstate Значение перечисления CCS_...

Возвращает:

1: значение допустимо и установлено 0: значение недопустимо

Устанавливаемое целое значение является последним аргуметом pl_exp_abort() и pl_exp_stop_cont(), вызываемые для окончания экспозиций (кроме обычного завершения нормального режима).

Для простоты приведем значения: 0=CCS_NO_CHANGE, 1=CCS_HALT, 2=CCS_HALT_CLOSE_SHTR, 3=CCS_CLEAR, 4=CCS_CLEAR_CLOSE_SHTR, 5=CCS_OPEN_SHTR, 6=CCS_CLEAR_OPEN_SHTR.

По умолчанию стоит 0 (NO_CHANGE).

Заметки:

Возможно, камера PI VersArray 1340x1300B не реагирует на эти установки. Кроме того, для SHTR_OPEN_MODE=2 (PRE_SEQUENCE) затвор открывается в начале последовательности и не закрывается, пока не будет установлен его режим SHTR_OPEN_MODE=1 и не будет произведена 1 экспозиция.

5.2.2.21 int setparam (const std::string *parname*, const std::string *strvalue*)

Установить параметр в ПЗС-системе

Аргументы:

parname имя параметра (обычно с префиксом PARAM_)

strvalue строковое представление параметра

Возвращает:

флаг записи параметра (1: записан, 0: не записан, -1: не существует)

У ПЗС-системы запрашивается существование, доступ на запись и тип параметра. Если запись возможна, то происходится преобразование параметра в значение определенного типа и попытка его записи в камеру. Причина неудачной записи параметра (если таковой существует в системе) сохраняется в VA::error_msg в случае возврата 0.

Заметки:

Значения логического типа выставляются как 0 или 1! (в отличии от **getparam()**)

5.2.2.22 void stop (void)

Прервать экспозицию или считывание

Процедура вызывает pl_exp_abort(), которая выставляет состояние CCS-системы камеры в значение, определяемое **set_stop_cam_state()** (по умолчанию - CCS_NO_CHANGE) и выставляет флаг stopped для синхронизации с работой процедур накопления данных.

В соответствии с описанием библиотеки PVCAM, вызванная вне экспозиции функция exp_abort может перевести систему камеры в другое состояние по отношению к текущему, если задать перед этим новое значение CCS вызовом **set_stop_cam_state()**. Однако вероятно, что камеры Princeton Instruments не реагируют на эти переключения (так, их CCS занимается непрерывной очисткой матрицы в состоянии ожидания вне зависимости от каких-либо условий).

5.2.2.23 int unset_roi (int *iroi*)

Деактивация iroi-го ROI

Аргументы:

iroi номер деактивируемого ROI (от 0 до MAXNROI-1)

Возвращает:

1: область деактивирована 0: номер ошибочен

Если область ранее была активна, число активных областей уменьшается на единицу. Если деактивирована последняя область, то автоматически вызывается set_roi(0,0,maxx-1, maxy-1), где maxx и maxy определяются как максимальные размеры, вписывающиеся в установленные текущие размеры приемника (размеры матрицы, если не включена опция PARAM_CUSTOM_CHIP) с учетом текущего бинирования.

5.3 Пространство имен VAd

Функции

- double **init** (void)
- int **park** (void)
- int **set_filter** (int ifilt)
- int **set_filter** (const char *name)
- int **get_nfilter** (void)
- std::string **get_filter** (int ifilt=-1)
- std::string **get_error_msg** (void)
- int **set_shutter** (int state)
- int **set_sync_shutter** (int sync)
- int **get_sync_shutter** (void)
- int **get_shutter** (void)
- double **get_filter_temp** (void)
- double **get_filter_heat_duty** (void)

Переменные

- const char **FILTER_MOVING** [] = "moving"

  возврат **get_filter()** *при неокончennом движении колеса фильтров*

- const char **FILTER_ERROR** [] = "undefined"

  возврат **get_filter()** *при неточной установке фильтра*

5.3.1 Подробное описание

Коллекция процедур для работы с устройствами ПЗС-фотометра VersArray 1340x1300B (исключая саму камеру). Фотометр включает колесо фильтров и затвор, управляемые шаговыми двигателями с микроконтроллерами на основе м/сх ATmega8 (Atmel). Кроме того есть возможность контроля температуры фильтров. Управление ведётся через линию RS-485 по компактному протоколу (ГАИШ).

Функции являются интерфейсами к модулям нижнего уровня для управления микроконтроллерами шаговых двигателей и контроля температуры. При ошибке при обращении к ним функции возвращают признак неудачи операции (0), а текст ошибки помещается в строку, возвращаемую функцией **get_error_msg()**.

5.3.2 Функции

5.3.2.1 std::string get_error_msg (void)

Вернуть смысл последней произошедшей ошибки

Возвращает:
Текстовое содержание последней ошибки устройств фотометра

Если ошибок не было, возвращаемая строка содержит текст "No error".

5.3.2.2 std::string get_filter (int *ifilt* = -1)

Вернуть имя фильтра на заданной позиции колеса или имя текущего фильтра

Аргументы:

ifilt Номер позиции (в диапазоне от 0 до **get_nfilter()**-1) или -1 для текущего

Позиции нумеруются начиная с 0 и до значения **get_nfilter()**-1. Функцию можно использовать для мониторинга установки фильтра, запускаемого на фоне запущенной **set_filter()**. Если возвращается значение равно FILTER_MOVING, то колесо фильтров еще не установилось в заданное положение.

Имена фильтров хранятся не в контроллере, а в отдельной подсекции "Glass" секции "VA adapter" конфигурационного файла device.cfg в виде пар значений Filter0 = "name0", Filter1="name0" и т.д. Ответственность за полноту и достоверность этих данных берет на себя редактор файла device.cfg, который и устанавливает стекла в колесо. В случае отсутствия искомого параметра Filter<ifilt> (при ifilt>=0) в указанной секции возвращается пустая строка.

Перед выдачей имени фильтра в случае отсутствия движения колеса проверяется состояние датчика фильтра. Если он не замкнут (нет точного совпадения колеса с заданным стопом), то выдаётся FILTER_ERROR.

Данная функция используется как для выяснения перечня установленных фильтров так и для получения имени установленного фильтра с контролем точности установки.

5.3.2.3 double get_filter_heat_duty (void)

Считать мощность нагрева в долях от максимальной

Возвращает:

Используемая доля максимально возможной мощности нагрева (от 0 до 1.0)

5.3.2.4 double get_filter_temp (void)

Считать температуру колеса фильтров

Возвращает:

Температура в градусах Цельсия

5.3.2.5 int get_nfilter (void)

Вернуть число доступных позиций колеса фильтров.

5.3.2.6 int get_shutter (void)

Вернуть состояние затвора по датчику лепестка

Возвращает:

1 - открыт 0 - не открыт полностью или закрыт

5.3.2.7 int get_sync_shutter (void)

Get synchro-status of shutter

Возвращает:
1 : shutter was set synchro, 0 : set non-synchro

Заметки:
There is no request to shutter-controller about synchro, so returned is the last set-synchro history.

5.3.2.8 double init (void)

Инициализация устройств фотометра.

Возвращает:
>0: макс. время инициализации в случае успеха её начала, -1 при ошибке

Адреса модулей и их установочные параметры считываются из файла device.cfg в текущей директории. Происходит установка связи с заданной в device.cfg в параметре "VA adapter"/Exchange/Baudrate скоростью обмена и связи модулями фотометра. В случае успеха запускается процедура инициализации колёс в отдельном потоке и возвращается максимальное время её выполнения. Последняя итеративно ищет нуль-пункты колёс (пока их невязки с предыдущими значениями не станут менее 2 микрошагов по модулю), затем затвор закрывается (если вдруг найден незакрытым), колесо фильтров ставится в положение фильтра по умолчанию, проверяется температура нагрева фильтров; по окончании вызывается процедура сигнализации конца инициализации vap_acknow_init с флагом успеха.

5.3.2.9 int park (void)

Парковка фотометра

Возвращает:
Успех операции

Закрывается связь с модулями. Ставится фильтр по умолчанию ("парковочный фильтр"), затвор закрывается.

5.3.2.10 int set_filter (const char * *name*)

Установка фильтра

Аргументы:
name Имя фильтра (одно из считанных из конфигурационного файла)

Возвращает:
Успех операции

Второй вариант функции установки фильтра, использующий ассоциативный массив номеров позиций фильтра в зависимости от имени фильтра. Регистр имени *name* не имеет значения.

5.3.2.11 int set_filter (int *ifilt*)

Установка фильтра

Аргументы:
ifilt Номер позиции колеса фильтров (номер фильтра от 0 до **get_nfilter()**-1)

Возвращает:
Успех операции

Позиции шагового двигателя управления фильтрами зашиты в микрокоде его контроллера, поэтому для установки фильтра надо лишь знать его номер *ifilt*. Процедура возвращает управление лишь после окончания движения колеса, что может занять от доли до нескольких секунд, в зависимости от установок скорости движения. Если "отзывчивость" программы критична во время установки фильтра, то эту функцию необходимо запускать в отдельном потоке, а состояние колеса фильтров при этом контролировать функцией **get_filter()**.

5.3.2.12 int set_shutter (int *state*)

Установить затвор

Аргументы:
state 1 - открыть, 0 - закрыть

Возвращает:
Успех установки затвора

Данная функция позволяет непосредственно контролировать затвор ПЗС-фотометра. Возврат из функции происходит по завершении движения затвора (если оно требовалось). Поскольку вращение шпинделя мотора, на котором установлен лепесток затвора, очень быстрое (доли секунды), возврат из функции происходит также "очень скоро".

5.3.2.13 int set_sync_shutter (int *sync*)

Установить синхронность работы затвора с внешним стробом

Аргументы:
sync 1 - работать синхронно, 0 - игнорировать строб

Возвращает:
Успех установки/сброса флага синхронности

При установке синхронной работы контроллер затвора открывает его при переходе внешнего сигнала из 0 в 1 и закрывает при сбросе его обратно в 0. Внешний сигнал подается на специальный разъём фотометра из выхода NOSCAN контроллера ПЗС VersArray коаксиальным кабелем.

Глава 6

VersArray driver Классы

6.1 Класс SVServ

`#include <svserv.h>`

Открытые типы

- typedef std::map< std::string, std::string >::const_iterator **param_ci**
- typedef std::map< std::string, std::string >::iterator **param_i**

Открытые члены

- **SVServ** (const char *id)

  Создать объект сервера и присвоить ему имя prog_id.

- int **open** (unsigned short port, int maxclient=1, int *fifo=0, int maxfifo=0, double timeout=0.3)

  Создать и открыть серверный сокет.

- void **close** (void)

  Закрыть серверный сокет и все открытые сокеты клиентов.

- ~**SVServ** (void)

  Деструктор сервера просто вызывает **close()**.

- const char * **ok** (**type_task** *task)

  Вернуть источнику команды сообщение об её успешном исполнении.

- const char * **error** (**type_task** *task)

  Вернуть источнику команды сообщение об ошибке её исполнения или разбора.

- int **discard** (**type_task** *task)

  Игнорировать команду (удалить из списка текущих команд).

- const char * **acknow** (bool success, **type_task** *t)

Функция посылки ответа.

- **type_task** * **getCommand** (char **eventstr)

  Получение команды из разных источников.

Открытые атрибуты

- type_status **status**

  Текущий статус компонента, запрашивается стандартной командой GET STATUS.

6.1.1 Подробное описание

Этот класс реализует функции обмена через сокет-соединения для программы, управляемой посредством протокола типа Supervisor. Подразумевается, что объект этого класса должен быть создан (статически или динамически) в **main()**. Далее происходит открытие серверного сокета и/или FIFO(s) для получения команд с помощью **open()**. После этого следует в цикле опрашивать **getCommand()** и вызывать с полученными заданиями их обработчик и по окончании обработки каждого задания (а также перед началом их продолжительного выполнения) вызывать **SVServ::ok()** или **SVServ::error()**. Сервер закрывается с помощью **close()** или деструктора.

6.1.2 Методы

6.1.2.1 const char* SVServ::acknow (bool *success*, type_task * *t*)

Функция посылки ответа.

Аргументы:

success признак успешности команды

t указатель на задание

Возвращает:

ссылка на строку, выданную источнику команды или 0 при ошибке отправки сообщения

Осуществляет формирование и посылку ответа, т.е. реализует функции **ok()** и **error()**, в зависимости от параметра *success*.

Можно пользоваться как этой функцией, так и ее вариантами **ok()** или **error()**, в зависимости от того, что удобнее по контексту.

6.1.2.2 void SVServ::close (void)

Закрыть серверный сокет и все открытые сокеты клиентов.

Заметки:

Вызов этой процедуры для закрытого сервера не вызывает никаких действий.

6.1.2.3 int SVServ::discard (type_task * *task*)

Игнорировать команду (удалить из списка текущих команд).

Аргументы:
task полученный из **getCommand()** указатель на (игнорируемое) задание

Возвращает:
0: задание удалено -1: задание не найдено среди текущих

Этот метод используется в тех (редких) случаях, когда полученную команду необходимо проигнорировать без посылки подтверждения ее источнику.

6.1.2.4 const char* SVServ::error (type_task * *task*) [inline]

Вернуть источнику команды сообщение об ошибке её исполнения или разбора.

Аргументы:
task полученный из **getCommand()** указатель на (невыполненное) задание

Возвращает:
ссылка на строку, выданную источнику команды или 0 при ошибке отправки сообщения

Выдача сообщения "<COMID> ERROR STATUS=status\n" источнику команды, вызываемая в случае ее неуспешного выполнения.

Значение параметра STATUS при формировании сообщения берется из переменной status объекта. Однако, если присвоить параметру STATUS какое-либо иное значение (обычно - "ERSYN" или "ERPAR" в результате обнаружения проблем с именами параметров или их значениями), то оно будет выдано как есть. Этот "локальный" статус не изменяет статуса сервера.

6.1.2.5 type_task* SVServ::getCommand (char ** *eventstr*)

Получение команды из разных источников.

Аргументы:
eventstr ссылка на строку, где записан строковый результат выполнения функции

Возвращает:
указатель на задание на выполнение команды в случае получения команды, 0 в случае отсутствия

В цикле идет ожидание поступления команды из одного из открытых сокетов или из FIFO консоли. Кроме того, идет опрос, нет ли новых сокет-соединений и их открытие если они есть и их открытие возможно. Возвращаемый указатель есть признак приема команды без обнаружения ошибки.

В случае открытия/закрытия нового порта клиента возвращается 0, а в *eventstr* сохраняется указатель на сообщение об этом событии (типа: "Connect from 127.0.0.1 port 12345", или "Client disconnected").

По поступлении команды создается задание типа **type_task** и возвращается указатель на него. При этом *eventstr* будет ссылаться на строковое представление полученной команды, которое можно использовать для визуализации обмена и протоколирования. Внешний обработчик должен выполнить это задание и, в течение короткого промежутка времени, не большего установленного в протоколе обмена с клиентом таймаута (обычно не более 1-3 секунд), воспользоваться одной из функций **SVServ::ok()** или **SVServ::error()**.

Если выполнение продолжительное (как разрешено для команд INIT, RUN, PARK), то обработчик обязан оценить (максимальное) время выполнения в секундах, присвоить это время t->param["WAIT"] как целочисленное значение в строковом представлении и вызвать SVServ::ok(t), а затем запустить команду на выполнение. По окончании выполнения команды следует опять вызвать SVServ::ok(t), но уже не создавая параметра "WAIT" (он удаляется после посылки ответа с ним источнику, но соответствующее задание не удаляется).

При возникновении ошибки разбора команды или несоответствия типа команды и текущего статуса возвращается 0, а соответствующее сообщение источнику команды отправляется автоматически. В *eventstr* будет лежать строка с ошибочной командой, отделенная знаком символом возврата каретки ('\n') от строки ответа на эту ошибочную команду: например "1 FET IDENT\n1 ERROR STATUS=ERSYN".

При возникновении таймаута приема команды возвращается 0 и *eventstr* также будет ссылаться на 0.

Заметки:

Следующие правила касаются случаев НЕсоответствия команд и статуса:

- INIT, PARK, QUIT: (status==STATUS_BUSY)
- RUN: (status!=STATUS_READY)
- STOP, SET: (status!=STATUS_READY && status!=STATUS_BUSY)

Эти правила применяются только в случае, если флаг *checkstatus* установлен в *true* (значение по умолчанию). В противном случае проверки не производится (режим работы при неисправностях).

6.1.2.6 const char* SVServ::ok (type_task * *task*) [inline]

Вернуть источнику команды сообщение об её успешном исполнении.

Аргументы:

task полученный из **getCommand()** указатель на (выполненное) задание

Возвращает:

ссылка на строку, выданную источнику команды или 0 при ошибке отправки сообщения

Выдача сообщения "<COMID> OK[parameters]\n" источнику команды, вызываемая в случае ее успешного выполнения.

При окончании выполнения команд RUN, INIT, PARK и STOP выдается только параметр статуса (например, "123 OK STATUS=READY").

Все параметры выдаются только в случае ответа на команду GET. При этом, если в строке значения параметра присутствует пробел и первый символ не является знаком двойной кавычки, то значение при выдаче ставится в двойные кавычки.

После выдачи ответа задание удаляется из контейнера текущих заданий. Однако, если среди параметров в *task* присутствует и параметр WAIT, то команда не удаляется из контейнера

текущих команд, а параметр WAIT=значение выдается вместо статуса, подразумевая статус BUSY. В этом случае после выдачи такого ответа (например: "123 OK WAIT=12") задание не удаляется из памяти, но параметр WAIT удаляется из массива task->param. Присваивание этого параметра WAIT можно выполнять сколько угодно раз перед вызовами данной функцияя – это может использоваться, если время выполнения команды оказалось дольше запланированного вначале. Если же в очередной раз параметр WAIT не был создан, то при следующем вызове **ok()** (или **error()**) с этим же заданием *task* параметра WAIT среди других уже не будет и задание будет считаться выполненным и будет удалено из памяти.

При разборе команды типа GET (task->cmd=="SVServ::CMD_GET) параметры IDENT, CHECKSTATUS и STATUS можно игнорировать, поскольку их значения заполняются из внутренних переменных prog_id, checkstatus и status при формировании ответа (в отличие от **error()**).

См. также:
SVServ::acknow() SVServ::type_task

6.1.2.7 int SVServ::open (unsigned short *port*, int *maxclient* = 1, int * *fifo* = 0, int *maxfifo* = 0, double *timeout* = 0.3)

Создать и открыть серверный сокет.

Аргументы:

port Номер порта серверного сокета

maxclient Максимальное число клиентов, обслуживаемых одновременно

fifo Массив дескрипторов открытых FIFO как источников команд

maxfifo Число открытых FIFO (длина *fifo*)

timeout длительность таймаута при ожидании событий на сокете или FIFO

Возвращает:
успех открытия порта сервера (1: открыт, 0: ошибка)

По умолчанию, одновременно может открываться только один сокет клиента и ни одного FIFO для подачи команд (например, с консоли).

Ответы клиентам посылаются в их открытые порты, ответы на команды из FIFO просто выдаются на консоль.

Объявления и описания членов класса находятся в файле:

- /home/victor/versarray/svserv.h

6.2 Структура SVServ::type_task

`#include <svserv.h>`

Открытые атрибуты

- unsigned short **source**
  номер источника команды (FIFO или клиента)
- unsigned short **comid**
  номер команды
- type_command **cmd**
  команда
- std::map< std::string, std::string > **param**
  пары параметр-значение

6.2.1 Подробное описание

Тип задания на выполнение команды.

В результате успешного разбора входящего сообщения серверу (команды) во внутреннем контейнере заданий создается и заполняется данная структура. Она содержит данные об источнике команды (*comid* и *source*), тип команды и набор ассоциированных параметров. Согласно протоколу SV, для команды SET эти параметры всегда сопровождаются устанавливаемыми для них значениями; для команд INIT, PARK, STOP, RUN могут появляться параметры как со значениями, так и без таковых. Последние играют роль ключей и их формальное значение при разборе команды в **SVServ::getCommand()** присваивается строке "<switch>". Напротив, для команд GET **все** параметры должны быть присланы без значений и их значения при разборе становятся равными "<switch>".

Задание, полученное с помощью команды **getCommand()**, содержит всю информацию о пришедшей команде (см. описания полей структуры), необходимую для ее выполнения. Защиты от изменения полей номера команды и ее типа не предусмотрено для простоты реализации, но их изменение при разборе может повлиять на правильность посылки ответа, когда с указателем на это задание будет использована функция **ok()** или **error()**. Для перебора имеющихся параметров следует использовать функции find() ассоциативного контейнера **type_task::param**, или итераторы. Помните, что обращение к param с несуществующим до этого ключом приводит к его созданию со значением по умолчанию (пустой строкой) и это значение появится среди других при ответе на команду GET. Имена параметров всегда односложные и представляются в верхнем регистре.

После выполнения задания используется функция **ok()** или **error()**, которые отсылают ответ источнику команды и удаляют задание из памяти. Исключением является случай, когда в результате выполнения команды RUN среди параметров появляется параметр "WAIT", значение которого (по протоколу) должно быть строковым представлением времени выполнения назначенной команды в секундах (целое число). Тогда это задание НЕ удаляется из контейнера, а его выполнение должно в течение этого времени быть повторно подтверждено вызовом функции **ok()** или **error()**.

Источник команды нумеруется исходя из следующего правила: сначала, начиная с 0, все открытые FIFO, затем открытые клиенты.

Заметки:

Ни создавать специально объекты этого типа, ни уничтожать их не следует. Создаются они только функцией **getCommand()**, а уничтожаются **ok()**, **error()** или **discard()**.

Объявления и описания членов структуры находятся в файле:

- /home/victor/versarray/svserv.h

Глава 7

VersArray driver Тематические описания

7.1 VAP: драйвер блока фильтров

Программа-сервер для работы с фотометром ПЗС-камеры PI VersArray. Предназначена для работы с устройствами фотометра (колесо фильтров и затвор) в текстовом режиме как в режиме отладки, так и при научных исследованиях (напрямую, из скриптов с client.tcl, или из программы-графического интерфейса) и позволяет организовать сложные алгоритмы работы загрузкой необходимых параметров камеры и режимов работы. Функции работы с самой камерой реализует программа **VA**; таким образом, работа с комплексом ПЗС-фотометра организована по двухъядерной схеме (программы **VA** и VAP могут работать на разных машинах).

Программа поддерживает протокол обмена Supervisor по сокетным соединениям с протоколом TCP/IP и имеет номер порта по умолчанию 16101. Также обслуживается консольный ввод с этим же протоколом. Параметр запуска -p (например, ./VAP -p 16124) позволяет установить другой номер порта сокета. Также поддерживаются параметры -h для вывода подсказки и -v для вывода названия и версии программы.

Следующие команды обрабатываются:

- INIT, PARK, QUIT - стандартно, без параметров
- GET STATUS IDENT ERMSG - статус, версия, текст ошибки
- SET FILTER=name FILTER=n TFILTER=dd.dd SHUTTER=n SYNC=l - установки в фотометре: фильтра по имени или номеру, температуры фильтра, состояния и синхронности затвора
- GET FILTER FILTERi NFILTER TFILTER PFILTER SHUTTER - чтение состояния фотометра: имени текущего фильтра, имени фильтра в позиции i, числа позиций, температуры и относительной мощности нагрева фильтров, затвора
- SET CHECKSTATUS=OFF - отмена проверки статуса перед исполнением команды

Следует обратить особое внимание, что установаленный фильтр и состояние затвора необходимо получать командами GET FILTER и GET SHUTTER для передачи в программу управления камерой с помощью команд SET FITSFILTERS=, SET FITSSHUTTERI= для записи этой информации в заголовки FITS-файлов получаемых изображений!

Команды SET/GET с параметрами *FILTER*, SYNC и SHUTTER позволяют обращаться к устройствам фотометра, таким как термостатируемое колесо фильтров и затвор. Конфигурация соответствующих модулей записана в файле device.cfg. Там же дана привязка стекол фильтров к позициям колеса.

Примеры работы (идентификаторы команд обязательны, но частью опущены):

1. Инициализация фотометра (поиск нуль-пунктов колес):

INIT

и подождать, пока

GET FILTER не вернет OK FILTER=<default-filter> и GET SHUTTERне вернет OK SHUTTER=0 (несколько секунд).

2. Считывание доступных фильтров (числа и имён): GET NFILTER GET FILTER0 GET FILTER1 ...<

2. Установка фильтра: SET FILTER= где name - это строковое название фильтра, указанное в файле device.cfg, или номер фильтра от 0 до Nf-1.

8. Разбор ошибок:

GET ERMSG -> OK ERMSG="сообщение" или "код_ошибки расшифровка"

Если при ошибке возвращается статус ERSYN, то это ошибка набора команды или параметра. Если ERPAR - то данный параметр недопустим. Что сделано неправильно - возвращается в ERMSG. Статусы ERSYN и ERPAR временные и не возвращаются GET STATUS.

Замечания по структуре программы.

Основной модуль программы (main) содержит в себе цикл опроса сервера (обслуживающего сокетные соединения и консоль) и вызов обработчика поступивших команд, а также вывод протокола обмена командами и ответами на консоль. Обработчик команд, находящийся здесь же, разбирает команды и, в зависимости от ключевых слов команд и параметров, вызывает соответствующие процедуры из пространства **VAd** для работы с устройствами фотометра. По результатам их выполнения вызывается **SVServ::acknow()** с соответствующим признаком успеха. Кроме того, в структуру полученного задания могут добавляться результирующие значения запрошенных у программы параметров.

Обработчик вызывает следующие процедуры **VAd** по поступившим командам (i - целый параметр, d - вещественный, [...]-необязательная часть):

- INIT : init()
- PARK, QUIT : park()
- GET ERMSG : get_error_msg()
- GET FILTER[i] : get_filter(i)
- GET SHUTTER : get_shutter()
- GET TFILTER : get_filter_temp()
- GET PFILTER : get_filter_heat_duty()
- GET NFILTER : get_nfilter()
- SET FILTER=[i | s] : set_filter(i | s)

- SET SHUTTER=i : set_shutter(i)
- SET SYNC=i : set_sync_shutter(i)

Про содержание упомянутых процедур смотрите их описание в пространстве **VAd**.
Работа с сервером описана в **SVServ**.

Также программа, кроме модулей vaphot и svserv, использует следующие:

- svcons: для консольного ввода команд через FIFO в программу
- configs: для разбора команд на слова (C)V.Kornilov
- astrotime: для точной фиксации времени начала экспозиции (C)V.Kornilov
- exchange: для работы с линией RS485 (C)V.Kornilov
- module: для работы с модулем RS485 (C)N.Shatsky
- fastmotor: для работы с модулем шагового двигателя (C)N.Shatsky

Эти модули в настоящем описании не документированы, так как являются универсальными и используемыми в других программах.