Прикладное программное обеспечение "ПАПИЛОН-ДС-4Х" SDK дактилоскопических сканеров ДС-ХХ Руководство программиста

Содержание

Состав библиотеки

Библиотека состоит из следующих компонентов:

- динамический модуль lsclient.dll, либо lsclient_rtm.dll, в зависимости от поддерживаемых устройств
- заголовочный файл **lsclient.h**
- пример использования библиотеки lsclient_test
- пакет DKMS для установки драйверов сканеров (ОС Linux)
- пакеты с драйверами для сканеров

Совместимость

Библиотека собрана на дистрибутивах

- Linux X86 RHEL5.5, gcc-4.1.2.
- Linux X64 RHEL6.3, gcc-4.4.6
- Windows X86 MSVC2010

Драйвера

Linux

Сам DKMS также поставляется в виде rpm-пакета. Порядок установки драйверов следующий:

- установить пакет dkms командой rpm -i dkms-2.0.13-1.noarch.rpm
- установить драйвер командой: rpm -i ds22_usb-4.2.2-3dkms.noarch.rpm (для сканеров DS21,DS22)

На компьютере должны быть установлены исходные тексты ядра либо заголовочные файлы ядра.

В результате этих действий DKMS скомпилирует драйвер и установит его в каталог /lib/modules/`uname -r`/kernel/drivers/misc.

Файл загрузки драйвера и создания специальных файлов в каталоге /dev/ ds22_usb будет установлен в каталоге /etc/rc.d/init.d/ и включен в 3,4,5 runlevel-ы.

При загрузке компьютеры сервис ds22_usb загрузит драйвер сканера и создаст необходимые файлы в каталоге /dev/usb.

Windows

После подключения сканера к компьютеру ОС запросит драйвера для сканера - необходимо указать каталог, в котором находятся драйвера для соответствующей модели сканера, например Drivers/DS22/. Если санер поставляется со встроенным LCD-экраном, то после установки драйверов для сканера будет выдан запрос на драйвера для LCD-экрана, необходимо указать каталог Drivers/LCD.

Поддерживаемые сканеры

Библиотекой поддерживаются следующие сканеры, выпускаемые фирмой "Папилон":

- Сканер для получения оттисков отпечатков DS21/DS22
- Сканер для получения оттисков отпечатков DS20/FX2000
- Сканер для получения прокатанных отпечатков и контрольных оттисков DS9.
- Сканер для получения прокатанных отпечатков и контрольных оттисков DS30.
- Сканер для получения прокатанных отпечатков и контрольных оттисков DS30M.
- Сканер для получения прокатанных отпечатков и контрольных оттисков DS31.
- Сканер для получения прокатанных отпечатков, контрольных оттисков и изображений ладоней DS7.
- Сканер для получения прокатанных отпечатков, контрольных оттисков и изображений ладоней DS10.
- Сканер для получения прокатанных отпечатков, контрольных оттисков и изображений ладоней DS14.
- Сканер для получения прокатанных отпечатков, контрольных оттисков и изображений ладоней DS40.
- Сканер для получения прокатанных отпечатков, контрольных оттисков и изображений ладоней DS40M.
- Сканер для получения прокатанных отпечатков, контрольных оттисков и изображений ладоней DS45.
- Сканер для получения прокатанных отпечатков, контрольных оттисков и изображений ладоней DS45M.
- Сканер для получения прокатанных отпечатков, контрольных оттисков DS30N.
- Сканер для получения прокатанных отпечатков, контрольных оттисков DS30NM.
- Сканер для получения прокатанных отпечатков DS22N.
- Сканер для получения прокатанных отпечатков DS21S (Orion) с возможностью определения муляжа пальца.

Порядок установки драйверов описан в разделе "Драйвера".

Порядок работы с библиотекой

Приложение, использующее данную библиотеку, должно:

1. Создать объект LSClient
2. Инициализировать следующие поля:
    - LSClient::Client
    - LSClient::debug_level
    - LSClient::GetCommand
    - LSClient::DrawPreview
    - LSClient::ClearPreview
    - LSClient::TakeImage
    - LSClient::GetWorkDir
    - LSClient::ProcessButton
    - LSClient::InitDone
    - LSClient::StateChanged
    - LSClient::DrawLCD

3. Вызывать функцию LSCInit().
4. Запустить поток прокатки функцией LSCStart().
5. Дождаться окончания инициализации по смене статуса с LSC_STATE::LSC_STAT_INIT на другой, либо после вызова callback-функции LSClient::InitDone().
6. Начать работу с потоком прокатки отдавая команды. через callback-функцию LSClient::GetCommand().
7. Для включения алгоритма слежения за отпечатком и прокатки необходимо подать команду LSC_COMMAND::LSC_CMD_ROLL.
8. После завершения работы с потоком необходимо либо остановить его командой LSC_COMMAND::LSC_CMD_STOP (если планируется дальнейшее использование сканера), либо завершить функцией LSCShutdown().
9. После остановки потока командой LSC_COMMAND::LSC_CMD_STOP можно возобновить его работу, подав команду LSC_COMMAND::LSC_CMD_ROLL.
10. После завершения функцией LSCShutdown() возобновить работу данного потока невозможно, в этом случае необходимо заново инициализировать объект LSClient и сам поток прокатки.

Группы

Полный список групп.

- Команды для потока прокатки
- Коды кнопок клавиатуры сканера
- Коды режимов управления светодиодами индикации сканера
- Номера диодов индикации сканера
- Ошибки процесса прокатки, допущенные оператором
- Типы сканеров
- Возвращаемые функциями ошибки
- Состояние потока прокатки
- Зоны прокатки
- Структура LSClient
- Callback-функции верхнего приложения
- Функции библиотеки
- Функции сжатия/разжатия WSQ

Команды для потока прокатки

Перечисления

enum LSC_COMMAND

Элементы перечислений:

- LSC_CMD_NOP - пустая команда
- LSC_CMD_EXIT - завершить поток прокатки
- LSC_CMD_STOP - остановить режим ожидания отпечатка и прокатки
- LSC_CMD_ROLL - запустить режим ожидания отпечатка и прокатки
- LSC_CMD_CLEAR - компенсировать грязь на призме
- LSC_CMD_TOUCH - захватить оттиск прямо сейчас

- LSC_CMD_START - не использовать
- LSC_CMD_STOP_ROLL - не использовать
- LSC_CMD_FINGERS - команды переключения текущего отпечатка. Для пальцев 0-9 -прокатка, для остальных - оттиск
- LSC_CMD_FINGER1 - Правый большой
- LSC_CMD_FINGER2 - Правый указательный
- LSC_CMD_FINGER3 - Правый средний
- LSC_CMD_FINGER4 - Правый безымянный
- LSC_CMD_FINGER5 - Правый мизинец
- LSC_CMD_FINGER6 - Левый большой
- LSC_CMD_FINGER7 - Левый указательный
- LSC_CMD_FINGER8 - Левый средний
- LSC_CMD_FINGER9 - Левый безымянный
- LSC_CMD_FINGER10 - Левый мизинец
- LSC_CMD_FINGER11 - Левые 4 оттиска
- LSC_CMD_FINGER12 - Левый большой оттиск
- LSC_CMD_FINGER13 - Правый большой оттиск
- LSC_CMD_FINGER14 - Правые 4 оттиска
- LSC_CMD_FINGER15 - Левая ладонь
- LSC_CMD_FINGER16 - Правая ладонь
- LSC_CMD_FINGER17 - Левая ладонь, ребро
- LSC_CMD_FINGER18 - Правая ладонь, ребро

См. определение в файле **lsclient.h** строка 150

```
00151 {
00152         LSC_CMD_NOP = 0, /*!< пустая команда  */
00153         LSC_CMD_EXIT = 1, /*!< завершить поток прокатки */
00154         LSC_CMD_STOP = 2, /*!< остановить режим ожидания отпечатка и
прокатки */
00155         LSC_CMD_ROLL = 3, /*!< запустить режим ожидания отпечатка и
прокатки */
00156         LSC_CMD_CLEAR = 4, /*!< компенсировать грязь на призме */
00157         LSC_CMD_TOUCH = 5, /*!< захватить оттиск прямо сейчас */
00158         LSC_CMD_START = 6, /*!< не использовать */
00159         LSC_CMD_STOP_ROLL = 7, /*!< не использовать */
00160
00161         /*! Команды переключения текущего отпечатка.
00162          * Для пальцев 0-9 -прокатка, для остальных - оттиск
00163          */
00164         LSC_CMD_FINGERS = 20, LSC_CMD_FINGER1 = 20, /*!< Правый
большой. */
00165         LSC_CMD_FINGER2 = 21, /*!< Правый указательный. */
00166         LSC_CMD_FINGER3 = 22, /*!< Правый средний. */
00167         LSC_CMD_FINGER4 = 23, /*!< Правый безымянный. */
00168         LSC_CMD_FINGER5 = 24, /*!< Правый мизинец. */
00169         LSC_CMD_FINGER6 = 25, /*!< Левый большой. */
00170         LSC_CMD_FINGER7 = 26, /*!< Левый указательный. */
00171         LSC_CMD_FINGER8 = 27, /*!< Левый средний. */
00172         LSC_CMD_FINGER9 = 28, /*!< Левый безымянный. */
00173         LSC_CMD_FINGER10 = 29, /*!< Левый мизинец. */
00174         LSC_CMD_FINGER11 = 30, /*!< Левые 4 оттиска. */
```

```
00175         LSC_CMD_FINGER12 = 31, /*!< Левый большой оттиск. */
00176         LSC_CMD_FINGER13 = 32, /*!< Правый большой оттиск. */
00177         LSC_CMD_FINGER14 = 33, /*!< Правые 4 оттиска. */
00178         LSC_CMD_FINGER15 = 34, /*!< Левая ладонь. */
00179         LSC_CMD_FINGER16 = 35, /*!< Правая ладонь. */
00180         LSC_CMD_FINGER17 = 36, /*!< Левая ладонь, ребро. */
00181         LSC_CMD_FINGER18 = 37, /*!< Правая ладонь, ребро. */
00182 } LSC_COMMAND;
```

Коды кнопок клавиатуры сканера

Перечисления

enum LSC_BUTTONS

Элементы перечислений:

- LSC_BTN_NONE - ничего не нажато
- LSC_BTN_LEFT - нажата стрелка влево
- LSC_BTN_RIGHT - нажата стрелка вправо
- LSC_BTN_OK - нажата кнопка Ок
- LSC_BTN_CANCEL - нажата кнопка Cancel

См. определение в файле **lsclient.h** строка 208

```
00209 {
00210         /**< ничего не нажато */
00211         LSC_BTN_NONE = 0,
00212
00213         /**< нажата стрелка влево */
00214         LSC_BTN_LEFT = 1,
00215
00216         /** нажата стрелка вправо */
00217         LSC_BTN_RIGHT = 2,
00218
00219         /** нажата кнопка Ок */
00220         LSC_BTN_OK = 4,
00221
00222         /** нажата кнопка Cancel */
00223         LSC_BTN_CANCEL = 8,
00224
00225 } LSC_BUTTONS;
```

Коды режимов управления светодиодами индикации сканера

Перечисления

enum LSC_LED_MODE

Режим индикации изменяется функцией LSC_SetLedMode().

Элементы перечислений:

- LSC_LED_AUTO - автоматический режим индикации
- LSC_LED_MANUAL - ручной режим, индикация управляется функцией LSC_SetLed()

См. определение в файле **lsclient.h** строка 234

```
00235 {
00236         /** Автоматический режим индикации. */
00237         LSC_LED_AUTO = 0,
00238
00239         /** Ручной режим, индикация управляется функцией LSC_SetLed()
*/
00240         LSC_LED_MANUAL = 1,
00241
00242 } LSC_LED_MODE;
```

Номера диодов индикации сканера

Перечисления

enum LSC_LED

Элементы перечислений:

- LSC_GREEN_LED - зеленый диод
- LSC_RED_LED - красный диод

См. определение в файле **lsclient.h** строка 250

```
00251 {
00252         /** Зеленый диод */
00253         LSC_GREEN_LED = 0,
00254
00255         /** Красный диод */
00256         LSC_RED_LED = 1,
00257
00258 } LSC_LED;
```

Типы сканеров

Перечисления

enum LSC_DEVICE

Элементы перечислений:

- UNKNOWN - Неизвестный сканер
- DS7 -Ладонный сканер ДС7

- DS9 - Пальцевый сканер ДС9 IEEE1394 или PCI
- DS21 - Пальцевый сканер ДС22 или ДС21
- FX2000 - Пальцевый сканер FX2000
- DS30 - Пальцевый сканер с контрольными оттисками ДС30
- DS31 - Пальцевый сканер с контрольными оттисками ДС31(2 призмы)
- DS14 - Ладонный сканер ДС14-USB
- DS40 - DS40 palm scanner
- FX2001 - Fingerprint FX2000 scanner with new sensor
- DS16 - Palm scanner DS16 + DS22 for rolling
- DS24 - Сканер отпечатков DS24, 1000дпи
- CM_VF - Cross Match Verifier.
- FDS7 - DS7 IEEE-1394.
- CM_VMW - Cross Match VMW сканер.
- DS45 - Сканер DS45, DS45M.
- DS22N - Сканер DS22N.
- DS30N - Сканер DS30N.
- DS30NM - Сканер DS30NM.
- DS21N - Сканер DS21N.
- CM_1000 - Сканер Cross Match 1000p с прокаткой Crossmatch.
- IDENTIX - Сканер Identix
- CM_1000P - Сканер DS21S(Orion) Сканер Cross Match 1000p с прокаткой Папилон.

См. определение в файле **lsclient.h** строка 298

```
00299 {
00300         UNKNOWN = 0, /*!< Неизвестный сканер */
00301         DS7 = 1, /*!< Ладонный сканер ДС7 */
00302         DS9 = 2, /*!< Пальцевый сканер ДС9 IEEE1394 или PCI */
00303         DS21 = 3, /*!< Пальцевый сканер ДС22 или ДС21 */
00304         FX2000 = 4, /*!< Пальцевый сканер FX2000 */
00305         DS30 = 5, /*!< Пальцевый сканер с контрольными оттисками ДС30
*/
00306         DS31 = 6, /*!< Пальцевый сканер с контрольными оттисками
ДС31(2 призмы) */
00307         DS14 = 7, /*!< Ладонный сканер ДС14-USB */
00308         DS40 = 8, /*!< DS40 palm scanner */
00309         FX2001 = 9, /*!< Fingerprint FX2000 scanner with new sensor */
00310         DS16 = 10, /*!< Palm scanner DS16 + DS22 for rolling */
00311         DS24 = 11, /*!< Сканер отпечатков DS24, 1000дпи */
00312         CM_VF = 12, /*!< Cross Match Verifier. */
00313         FDS7 = 13, /*!< DS7 IEEE-1394. */
00314         CM_VMW = 14, /*!<Cross Match VMW сканер. */
00315         DS45 = 15, /*!<Сканер DS45, DS45M. */
00316         DS22N = 16, /*!<Сканер DS22N. */
00317         DS30N = 17, /*!<Сканер DS30N. */
00318         DS30NM = 18, /*!<Сканер DS30NM. */
00319         DS21N = 19, /*!<Сканер DS21N. */
00320         CM_1000 = 21, /*!<Сканер Cross Match 1000p с прокаткой
Crossmatch.*/
00321         IDENTIX = 22, /*!<Сканер Identix*/
00322         DS21S = 23, /*!Сканер  DS21S(Orion)*/
```

```
00323         CM_1000P = 99, /*!<Сканер Cross Match 1000p с прокаткой
Папилон.*/
00324 } LSC_DEVICE;
```

Возвращаемые функциями ошибки

Перечисления

enum LSC_ ERROR

Элементы перечислений:

- LSC_CONNECT_ERROR - No errors
- LSC_MEMORY_ERROR - Ошибка соединения со сканером
- LSC_RESOURCES_ERROR - Мало памяти
- LSC_READ_CONFIG_ERROR - Ошибка выделения ресурсов IPC
- LSC_CALIBRATE_ERROR - Ошибка чтения конфигурации сканера
- LSC_ROLL_ERROR - Сканер не откалиброван
- LSC_EXIT_ERROR - Ошибка прокатки
- LSC_SLAP_IMAGE_ERROR - Поток прокатки закрыт Ошибка разделения контрольных оттисков
- LSC_SLAP_FINGER_NOT_FOUND - Ошибка разделения контрольных оттисков, не все отпечатки обнаружены
- LSC_ARG_ERROR - Неверные аргументы функции
- LSC_LICENSE_ERROR - Неверная лицензия

См. определение в файле **lsclient.h** строка 339

```
00340 {
00341         LSC_NO_ERROR = 0, /*! No errors */
00342         LSC_CONNECT_ERROR = 1, /*! Ошибка соединения со сканером */
00343         LSC_MEMORY_ERROR = 2, /*! Мало памяти */
00344         LSC_RESOURCES_ERROR = 3, /*! Ошибка выделения ресурсов IPC */
00345         LSC_READ_CONFIG_ERROR = 4, /*! Ошибка чтения конфигурации
сканера */
00346         LSC_CALIBRATE_ERROR = 5, /*! Сканер не откалиброван */
00347         LSC_ROLL_ERROR = 6, /*! Ошибка прокатки */
00348         LSC_EXIT_ERROR = 7, /*! Поток прокатки закрыт */
00349         LSC_SLAP_IMAGE_ERROR = 8, /*!< Ошибка разделения контрольных
оттисков.  */
00350         LSC_SLAP_FINGER_NOT_FOUND = 9, /*!< Ошибка разделения
контрольных оттисков, не все отпечатки обнаружены. */
00351         LSC_ARG_ERROR = 10, /*!< Неверные аргументы функции. */
00352         LSC_LICENSE_ERROR = 11, /*!< Неверная лицензия*/
00353 } LSC_ERROR;
```

Состояние потока прокатки

Перечисления

enum LSC_ STATE

Элементы перечислений:

- LSC_STAT_INIT - инициализация
- LSC_STAT_WAIT - ожидание команды без слежения за отпечатком и прокатки
- LSC_STAT_WAIT_FINGER - слежение за отпечатком
- LSC_STAT_ROLL - собственно процесс прокатки
- LSC_STAT_GLUE - обработка полученного после прокатки изображения
- LSC_STAT_CLEARING - компенсация грязи на призме
- LSC_STAT_EXITED - поток завершился
- LSC_STAT_ERROR - произошла какая-либо ошибка

См. определение в файле **lsclient.h** строка 363

```
00364 {
00365         LSC_STAT_INIT = 0, /*!< инициализация */
00366         LSC_STAT_WAIT = 1, /*!< ожидание команды без слежения за
отпечатком и прокатки */
00367         LSC_STAT_WAIT_FINGER = 2, /*!< слежение за отпечатком  */
00368         LSC_STAT_ROLL = 3, /*!< собственно процесс прокатки */
00369         LSC_STAT_GLUE = 4, /*!< обработка полученного после прокатки
изображения */
00370         LSC_STAT_CLEARING = 5, /*!< компенсация грязи на призме */
00371         LSC_STAT_EXITED = 6, /*!< поток завершился */
00372         LSC_STAT_ERROR = 7, /*!< произошла какая-либо ошибка */
00373 } LSC_STATE;
```

Зоны прокатки

Перечисления

enum LSC_ ROLL_PLACE

Определение зоны прокатки.

Элементы перечислений:

- LSC_RPLACE_AUTO - Автоматический выбор зоны прокатки.
- Правая рука прокатывается справа, левая рука прокатывается слева.
- LSC_RPLACE_LEFT - Прокатка выполняется в левой части призмы.
- LSC_RPLACE_RIGHT - Прокатка выполняется в правой части призмы.

См. определение в файле **lsclient.h** строка 386

```
00387 {
00388         /** Автоматический выбор зоны прокатки.
00389          *   Правая рука прокатывается справа, левая рука
прокатывается слева.
00390          */
00391         LSC_RPLACE_AUTO = 0,
00392
00393         /**Прокатка выполняется в левой части призмы*/
00394         LSC_RPLACE_LEFT = 1,
```

```
00395
00396         /**Прокатка выполняется в правой части призмы*/
00397         LSC_RPLACE_RIGHT = 2,
00398 } LSC_ROLL_PLACE;
```

Callback-функции верхнего приложения

Переменные

LSC_COMMAND(* LSClient::GetCommand)(LSC_HNDL Cl) [inherited]

Получение потоком прокатки команды от верхней программы. Функция вызывается внутри потока прокатки для получения команды от внешнего приложения.

Аргументы:

Cl - указатель на объект верхнего приложения.

Возвращает: команду для потока прокатки.

int(* LSClient::DrawPreview)(LSC_HNDL Cl, int finger_num, unsigned char *Buf, int W, int H) [inherited]

Вывод preview на экран. Когда отпечаток обнаружен на призме прибора, подготавливается изображение для показа оператору на экране. Эта функция вызывается из потока прокатки для передачи верхнему приложению подготовленного preview-изображения.

Аргументы:

Cl	- указатель на объект верхнего приложения.
finger_num	- номер текущего отпечатка, установленного одной из команд LSC_CMD_FINGERS.
Buf	- указатель на буфер с изображением, буфер глубиной 8 bpp (256 градаций серого), первый байт - левый верхний пиксель изображения.
W	- ширина изображения (количество колонок).
H	- высота изображения (количество строк).

Возвращает: 0 - в случае успеха.

int(* LSClient::ClearPreview)(LSC_HNDL Cl) [inherited]

Очистка окна preview. Вызывается, когда отпечаток убран с призмы либо процесс прокатки завершен.

Аргументы:

Cl - указатель на объект верхнего приложения.

Возвращает: 0 - в случае успеха.

int(* LSClient::TakeImage)(LSC_HNDL Cl, int finger_num, unsigned char *Buf, int W, int H, unsigned int roll_errors) [inherited]

Передача готового изображения отпечатка из потока прокатки в верхнее приложение. Вызывается когда процесс прокатки завершен.

Аргументы:

Cl	- указатель на объект верхнего приложения.
finger_num	- номер отпечатка
Buf	- указатель на буфер с изображением отпечатка, освобождается в потоке прокатки. Глубина буфера 8 bpp (256 градаций серого), разрешение 500 ppi, первый байт - верхний левый пиксел изображения. Может быть NULL, в этом случае будет установлена переменная roll_errors в соответствующее значение ошибки прокатки.
W	- ширина изображения(количество колонок).
H	- высота изображения(количество строк).
roll_errors	- маска ошибок прокатки. В случае критических ошибок параметр Buf возвращается как NULL.

Возвращает: 0 - в случае успеха.

char*(* LSClient::GetWorkDir)(LSC_HNDL Cl) [inherited]

Получить путь к рабочему каталогу. Вызывается из потока прокатки для получения каталога в котором можно сохранять временные файлы.

Аргументы:

Cl - указатель на объект верхнего приложения.

Возвращает: path - путь к каталогу.

int(* LSClient::ProcessButton)(LSC_HNDL Cl, LSC_BUTTONS button) [inherited]

Передача кода кнопки, нажатой на сканере. Вызывается из потока прокатки для передачи информации о нажатии какой-либо кнопки на клавиатуре сканера.

Аргументы:

Cl	- указатель на объект верхнего приложения.
button	- нажатая кнопка.

Возвращает: 0 в случае успеха.

void(* LSClient::InitDone)(LSC_HNDL Cl) [inherited]

Уведомление верхнего приложения о завершении инициализации потока прокатки. Вызывается из потока прокатки после завершения процедуры тестирования и инициализации сканера. После этого можно начинать работу с потоком прокатки - инициализация preview LSCSetPreview(), LSCGetPreviewSlap(), передавать команды в поток через callback LSClient::GetCommand().

Аргументы:

Cl - указатель на объект верхнего приложения.

void(* LSClient::StateChanged)(LSC_HNDL Cl, LSC_STATE state) [inherited]

Уведомление верхнего приложения о изменении состояния потока прокатки. Вызывается из потока прокатки после изменения состояния потока прокатки.

Аргументы:

Cl - указатель на объект верхнего приложения.
state - новое состояние потока прокатки.

void(* LSClient::DrawLCD)(LSC_HNDL Cl) [inherited]

Уведомление верхнего приложения о возможности отправить картинку на USB LCD. Вызывается из потока прокатки во время перерыва передачи данных со сканера по USB-шине. Используется только со сканерами, оборудованными LCD монитором(DS30M и DS15M). Если DrawLCD не инициализирован(равен NULL), то поток прокатки не будет вызывать этот callback.

Аргументы:

Cl - указатель на объект верхнего приложения.

void(* LSClient::RemoveFinger)(LSC_HNDL Cl) [inherited]

Уведомление верхнего приложения о том что накоплено достаточное количество кадров для построения финального изображения. Эта функция вызывается из потока прокатки, когда накоплено достаточное количество кадров для построения финального изображения, в случае прокатки отпечатка оператор должен сам решить, когда прокатка выполнена полностью. Если RemoveFinger не инициализирован (равен NULL), то поток прокатки не будет вызывать этот callback.

Аргументы:

Cl - указатель на объект верхнего приложения.

void(* LSClient::ScanComplete)(LSC_HNDL Cl) [inherited]

Сканирование всех отпечатков закончено. Вызывается из потока сканирования, когда отсканированы все отпечатки, используется только для сканера CorrMatch VMW.

void(* LSClient::AddScanStat)(LSC_HNDL Cl, char *data, int tag, int multiple) [inherited]

Передает в верхнее приложение статистику сканирования. Эти данные могут быть сохранены в отдельные текстовые записи для дальнейшего анализа ошибок прокатки оператора, ошибок работы сканера. Callback вызывается после LSC_CMD_START и LSC_CMD_ROLL команд для сохранения информации о модели, настройке сканера и версиях программного обеспечения.

TakeImage callback для сохранения информации о параметрах сканирования.

Аргументы:

Cl - указатель на объект верхнего приложения.
data - текстовая строка, заканчивающаяся нулем
tag - номер текстового тега записи
multiple - флаг множественного тега:

- 1 - множественный, данные должны быть добавлены к записям с таким же номером тега
- 0 - единичный, данные должны заместить записи с таким же номером тега

Функции библиотеки

Функции

LSCLIENT_API LSC_ERROR LSCInit	**(LSClient * cl, LSC_DEVICE device, int num)**

Инициализация библиотеки прокатки. Функция захватывает необходимые для работы ресурсы и создает соединение со сканером.

Аргументы:

cl - указатель на структуру LSClient.
device - тип сканера, с которым будет работать данный поток.
num - номер сканера, начиная с нуля.

Возвращает: код ошибки.

Примеры: mainform.cpp

```
LSCLIENT_API LSC_ERROR LSCSetPreview   ( LSClient *      cl,
                                         int *          width,
                                         int *          height
                                       )
```

Установка размеров preview-изображения при прокатке. Функция вычисляет параметры preview изображения и возвращает размеры preview в аргументах width и height. В дальнейшем именно с этими размерами будет вызываться callback-функция LSClient::DrawPreview(). Данную функцию можно вызывать только после окончания процесса инициализации. Окончание инициализации сигнализируется сменой статуса потока прокатки с LSC_STAT_INIT на любой другой и вызовом callback-функции LSClient::InitDone().

Аргументы:

cl - Указатель на структуру LSClient.
width - ширина изображения
height - высота изображения

Возвращает: код ошибки.

Примеры: mainform.cpp

```
LSCLIENT_API LSC_ERROR LSCGetPreviewSlap   ( LSClient *      cl,
                                             int *          width,
                                             int *          height
                                           )
```

Установка размеров preview-изображения при снятии оттисков. Функция вычисляет параметры preview изображения при снятии контрольных оттисков и возвращает размеры preview в аргументах width и height. В дальнейшем именно с этими размерами будет вызываться callback-функция LSClient::DrawPreview(). Данную функцию можно вызывать только после окончания процесса инициализации. Окончание инициализации сигнализируется сменой статуса потока прокатки с LSC_STAT_INIT на любой другой и вызовом callback-функции LSClient::InitDone().

Аргументы:

cl - Указатель на структуру LSClient.
width - ширина изображения
height - высота изображения

Возвращает: код ошибки.

Примеры: mainform.cpp

```
LSCLIENT_API LSC_ERROR LSCGetSlapSize   ( LSClient *      cl,
                                          int *           width,
                                          int *           height
                                        )
```

Не используется.

Аргументы:

cl - Указатель на структуру LSClient.
width - Указатель на ширину изображения.
height - Указатель на высоту изображения.

Возвращает: код ошибки.

```
LSCLIENT_API LSC_ERROR LSCGetFingerSize   ( LSClient *      cl,
                                            int *           width,
                                            int *           height
                                          )
```

Не используется.

Аргументы:

cl - Указатель на структуру LSClient.
width - Указатель на ширину изображения.
height - Указатель на высоту изображения.

Возвращает: код ошибки.

```
LSCLIENT_API LSC_ERROR LSCStart      ( LSClient *  cl )
```

Запуск потока прокатки. Функция порождает дополнительный поток, который выполняет процесс прокатки. Все callback-функции вызываются из этого потока. Поэтому любое обращение к графическому интерфейсу пользователя из них нежелательно. Лучше использовать систему сообщений между callback-функциями и главным потоком верхнего приложения.

Аргументы:

cl - Указатель на структуру LSClient.

Возвращает: код ошибки.

Примеры: mainform.cpp

```
LSCLIENT_ API void LSCShutDown      ( LSClient *  cl )
```

Завершение потока прокатки и освобождение всех ресурсов. Функция передает в поток прокатки команду на завершение и ожидает окончания этого потока, затем освобождает все занятые ресурсы и закрывает соединение со сканером.

Аргументы:

cl - Указатель на структуру LSClient.

Примеры: mainform.cpp

LSCLIENT_API LSC_STATE LSCGetState (LSClient * cl)

Получение текущего состояния потока прокатки. Функция возвращает статус потока прокатки. При изменении статуса так же вызывается callback-функция LSClient::StateChanged().

Аргументы:

cl - Указатель на структуру LSClient.

Возвращает: Состояние потока прокатки.

LSCLIENT_API LSC_ERROR LSCGetError (LSClient * cl)

Получение кода ошибки потока прокатки. Функция возвращает код ошибки потока прокатки. В случае какой-либо ошибки внутри потока статус потока меняется на LSC_STATE::LSC_STAT_ERROR.

Аргументы:

cl - Указатель на структуру LSClient.

Возвращает: код ошибки.

Примеры: mainform.cpp

LSCLIENT_API char* LSCGetVersion (LSClient * cl)

Получить версию библиотеки. Функция позволяет получить версию библиотеки в виде указателя на строку.

Аргументы:

cl - Указатель на структуру LSClient.

Возвращает: Строку с версией библиотеки.

```
LSCLIENT_API void LSCSetFingerArea      ( LSClient *      cl,
                                          int             left,
                                          int             right
                                        )
```

Не используется.

Аргументы:

cl - Указатель на структуру LSClient.
left - левая граница изображения.
right - правая граница изображения.

```
LSCLIENT_API LSC_ERROR LSCslapSegm      ( LSClient *      cl,
                                          unsigned char *  image,
                                          int             width,
                                          int             height,
                                          int             num,
                                          int *           left,
                                          int *           top,
                                          int *           fwidth,
                                          int *           fheight
                                        )
```

Сегментация контрольных оттисков. Находит отпечаток с заданным номером в изображении контрольных оттисков и возвращает его координаты.

Аргументы:

cl - Указатель на структуру LSClient
image - изображение контрольных оттисков
width - ширина изображения
height - высота изображения
num - номер отпечатка для выделения.
left - левая граница отпечатка в изображении
top - верхняя граница отпечатка в изображении
fwidth - ширина отпечатка
fheight - высота отпечатка

Возвращает: коды ошибок

```
LSCLIENT_API LSC_ERROR LSCslapSegmAll    ( LSClient *          cl,
                                           unsigned char *     image,
                                           int                 width,
                                           int                 height,
                                           int *               num,
                                           int *               left,
                                           int *               top,
                                           int *               fwidth,
                                           int *               fheight
                                         )
```

Сегментация контрольных оттисков. Разделяет контрольные оттиски на отдельные отпечатки и возвращает координаты и размеры найденных отпечатков.

Аргументы:

cl - указатель на структуру LSClient
image - изображение оттисков
width - ширина изображения
height - высота изображения
num - количество выделенных отпечатков
left - массив (int[4]) для сохранения левых границ отпечатков
top - массив (int[4]) для сохранения верхних границ отпечатков
fwidth - массив (int[4]) для сохранения ширины отпечатков
fheight - массив (int[4]) для сохранения высоты отпечатков

Возвращает: коды ошибок

Примеры: mainform.cpp.

```
LSCLIENT_API int LSC_GetScannersList      ( LSC_SCANNERS_LIST **   list  )
```

Формирует список подключенных сканеров.

Аргументы:

list - Указатель на указатель для списка сканеров.

Возвращает: длину выделенного списка подключенных сканеров

Примеры: mainform.cpp

```
LSCLIENT_API void LSC_SetLedMode          ( LSClient *          cl,
                                            LSC_LED_MODE        mode
                                          )
```

Установить режим индикации светодиодов сканера.

Аргументы:

cl - указатель на структуру верхнего приложения
mode - режим управления индикацией

Примеры: mainform.cpp

```
LSCLIENT_API LSC_ERROR LSC_SetLed      ( LSClient *      cl,
                                         LSC_LED         led,
                                         BOOL            on
                                       )
```

Выключение/выключение светодиодов индикации сканера в ручном режиме управления.

Аргументы:

cl - указатель на структуру верхнего приложения
led - номер светодиода, красный или зеленый.
on - состояние диода, TRUE - включить, FALSE - выключить.

Возвращает: коды ошибок

Примеры: mainform.cpp.

```
LSCLIENT_API LSC_ERROR LSC_SetUsbType  ( LSClient *      cl,
                                         BOOL            slow
                                       )
```

Переключение режима передачи данных сканером по USB-шине. Функция может применяться для уменьшения потока данных от сканера к компьютеру по USB-шине в случае медленной USB-шины.

Аргументы:

cl - указатель на структуру верхнего приложения
slow - TRUE - медленная USB-шина, FALSE - полноценная USB2 HIGH-SPEED 480mb/s шина.

Возвращает: коды ошибок

Примеры: mainform.cpp.

```
LSCLIENT_API void LSCSetCaptureMode    ( LSClient *                cl,
                                         LSC_CAPTURE_MODE          mode
                                       )
```

Установить режим захвата оттисков.

Аргументы:

cl - указатель на структуру верхнего приложения

mode - режим захвата оттисков

Примеры: mainform.cpp.

LSCLIENT_API LSC_ERROR LSCSetThreshold	**(LSClient ***	**cl,**
	int	**threshold**
	)	

Установить чувствительность алгоритма определения наличия отпечатков на призме. Чем больше число, те менее чувствителен алгоритм к отпечаткам и грязи соответственно. Рекомендуемое значение(оно же и по умолчанию) - 500.

Аргументы:

cl - указатель на структуру верхнего приложения
threshold - чувствительность, от LSC_MIN_THRESHOLD до LSC_MAX_THRESHOLD

LSCLIENT_API void LSCSetTouchMode	**(LSClient ***	**cl,**
	BOOL	**touch_mode**
	)	

Переключение режимов оттисков/прокатки для отпечатков.

Аргументы:

cl - указатель на структуру верхнего приложения
touch_mode - режим
- TRUE - берутся оттиски пальцев
- FALSE - пальца прокатываются

Примеры: mainform.cpp.

LSCLIENT_API LSC_ERROR LSC_SendJpeg2Device	**(LSClient ***	**cl,**
	unsigned char *	**buffer,**
	int	**length**
	)	

Отправить JPEG-изображение на монитор сканера. Данная функция работает только со сканером CrossMatch VMW.

Аргументы:

cl - указатель на структуру верхнего приложения
buffer - указатель на jpeg-изображения
length - длина буфера изображения

Возвращает: коды ошибок

Примеры: mainform.cpp.

```
LSCLIENT_API LSC_ERROR LSC_GetSizeLCD   ( LSClient *    cl,
                                          int *         width,
                                          int *         height
                                        )
```

Получить размеры экрана сканера. Работает только с интегрированным LCD-экраном сканера DS30NM, с остальными сканерами для вывода на экран необходимо использовать библиотеку liblcd.

Аргументы:

cl - указатель на структуру верхнего приложения
width - возвращается ширина экрана
height - возвращается высота экрана

Возвращает: возвращаемые функциями ошибки.

```
LSCLIENT_API LSC_ERROR LSC_ResetDataLCD   ( LSClient *    cl, )
```

Сброс USB-буферов для передачи на LCD экран сканера. Работает только с интегрированным LCD-экраном сканера DS30NM, с остальными сканерами для вывода на экран необходимо использовать библиотеку liblcd.

Аргументы:

cl - указатель на структуру верхнего приложения

Возвращает: возвращаемые функциями ошибки.

```
LSCLIENT_API LSC_ERROR LSC_WriteImageLCD   ( LSClient *    cl,
                                             long          size,
                                             void *        image
                                           )
```

Передать изображение на LCD экран сканера. Работает только с интегрированным LCD-экраном сканера DS30NM, с остальными сканерами для вывода на экран необходимо использовать библиотеку liblcd.

Аргументы:

cl - указатель на структуру верхнего приложения
size - размер буфера с изображением
image - буфер с изображением

Возвращает: возвращаемые функциями ошибки.

```
LSCLIENT_API LSC_ERROR LSC_WriteMaskLCD   ( LSClient *    cl,
                                            long          AMaskSize,
                                            void *        AMaskBuff
                                          )
```

Передать палитру на LCD экран сканера. Работает только с интегрированным LCD-экраном сканера DS30NM, с остальными сканерами для вывода на экран необходимо использовать библиотеку liblcd.

Аргументы:

cl - указатель на структуру верхнего приложения
AMaskSize - размер палитры
AMaskBuff - буфер с палитрой, по 3 байта на цвет, 256 цветов.

Возвращает: возвращаемые функциями ошибки.

```
LSCLIENT_API LSC_ERROR LSC_SetLightLCD   ( LSClient *    cl,
                                           int           light
                                         )
```

Установить яркость подсветки LCD экрана сканера. Работает только с интегрированным LCD-экраном сканера DS30NM, с остальными сканерами для вывода на экран необходимо использовать библиотеку liblcd.

Аргументы:

cl - указатель на структуру верхнего приложения
light - яркость подсветки, от 0 до 31.

Возвращает: возвращаемые функциями ошибки.

```
LSCLIENT_API LSC_ERROR LSC_SetLCDPower   ( LSClient *    cl,
                                           int           on
                                         )
```

Включение/выключение питания экрана сканера. Работает только с интегрированным LCD-экраном сканера DS30NM, с остальными сканерами для вывода на экран необходимо использовать библиотеку liblcd.

Аргументы:

cl - указатель на структуру верхнего приложения
on - 1 - включить питание, 0 - выключить.

Возвращает: возвращаемые функциями ошибки.

```
LSCLIENT_API BOOL LSC_IntegratedLCD      ( LSClient *    cl, )
```

Возвращает флаг наличия в сканере интегрированного LCD экрана.

Аргументы:

cl - указатель на структуру верхнего приложения

Возвращает:

- TRUE - экран интегрирован
- FALSE - экран не интегрирован

```
LSCLIENT_API LSC_ERROR LSC_GetFirmware   ( LSClient *    cl,
                                           char *       firmware
                                         )
```

Возвращается версию EEPROM firmware сканера.

Аргументы:

cl - указатель на структуру верхнего приложения
firmware - возвращается версия firmware

Возвращает: возвращаемые функциями ошибки.

```
LSCLIENT_API LSC_ERROR LSC_GetAfisQuality   ( LSClient *          cl,
                                              unsigned char *     image,
                                              int                 width,
                                              int                 height,
                                              int                 res,
                                              int *               aquality,
                                              int *               vquality,
                                              float *             pressure
                                            )
```

Расчет качества отпечатка по алгоритму Папилон.

Аргументы:

[in]	*cl*	- object of library.
[in]	*image*	- буфер с изображением. Изображение должно иметь глубину 8 bpp.
[in]	*width*	- количество колонок
[in]	*height*	- количество строк
[in]	*res*	- разрешение изображения в пикселах на дюйм, обычно 500.
[in]	*cl*	- указатель на структуру верхнего приложения.
[out]	*aquality*	- указатель для сохранения качества папиллярного узора. Изменяется от 0 (плохое) до 100 (отличное).
[out]	*vquality*	- указатель для сохранения качества изображения изменяется от 0 (худшее) до 100 (отличное). Характеризует динамический диапазон изображения.

[out] *pressure* - указатель для сохранения силы прижима пальца к призме. Изменяется 0.0 (слабое давление) до 2.0 (сильное давление). Нормальное значение 1.0.

Возвращает: возвращаемые функциями ошибки.

Примеры: mainform.cpp.

```
LSCLIENT_API LSC_ERROR LSC_GetNistQuality ( LSClient *          cl,
                                            unsigned char *     image,
                                            int                 width,
                                            int                 height,
                                            int                 res,
                                            int *               aquality,
                                            int *               vquality,
                                            float *             pressure
                                          )
```

Расчет качества отпечатка по алгоритму NIST.

Аргументы:

[in] *cl* - object of library.
[in] *image* - буфер с изображением. Изображение должно иметь глубину 8 bpp.
[in] *width* - количество колонок
[in] *height* - количество строк
[in] *res* - разрешение изображения в паикселах на дюйм, обычно 500.
[in] *cl* - указатель на структуру верхнего приложения.
[out] *aquality* - указатель для сохранения качества папиллярного узора. Изменяется от 5 (плохое) до 1 (отличное).
[out] *vquality* - указатель для сохранения качества изображения изменяется от 0(худшее) до 100(отличное). Характеризует динамический диапазон изображения.
[out] *pressure* - указатель для сохранения силы прижима пальца к призме. Изменяется 0.0 (слабое давление) до 2.0 (сильное давление). Нормальное значение 1.0.

Возвращает: возвращаемые функциями ошибки.

Примеры: mainform.cpp.

```
LSCLIENT_API BOOL LSC_ SupportRollPlace        ( LSClient *     cl, )
```

Проверить поддерживает ли сканер разные зоны прокатки.

Аргументы:

cl - указатель на структуру верхнего приложения.

Возвращает: true, если сканер поддерживает выбор различных зон прокатки.

```
LSCLIENT_API LSC_ERROR LSC_ SetRollPlace    ( LSClient *          cl,
                                              LSC_ROLL_PLACE      place
                                            )
```

Выбор зоны прокатки. Функцию можно использовать только для сканеров, которые поддерживают выбор зоны прокатки, в данный момент только для сканера ДС45 (М).

Аргументы:

cl - указатель на структуру верхнего приложения.
place - зона прокатки

Возвращает: возвращаемые функциями ошибки.

```
LSCLIENT_API BOOL LSC_EnableFakeDetection    ( LSClient *    cl,
                                               bool          _enable
                                             )
```

Включение/выключение режима определения муляжа отпечатка. При включении этой функции время захвата отпечатка увеличивается примерно в 2 раза. Данный режим работает только на некоторых сканерах.

Аргументы:

cl - указатель на структуру верхнего приложения.
_enable - true - включить определение муляжа, false - выключить.

Возвращает: false, если данная функция не поддерживается сканером.

```
LSCLIENT_API LSC_ERROR LSC_ CheckEncode    ( LSClient *          cl,
                                             unsigned char *     _image,
                                             int                 _width,
                                             int                 _height,
                                             BOOL *              _good
                                           )
```

Функция проверяет возможность извлечения мелких особенностей из изображения отпечатка. Эта функция анализирует качество изображения на предмет пригодности для кодирования и возвращает результат в параметре _good. Если изображение достаточного качества для последующего кодирования и сравнения, то выходной параметр _good будет содержать TRUE. Функция достаточно ресурсоемкая и занимает времени порядка 1 секунды на процессоре Intel Core i7 3ГГц для одного отпечатка. Использование данной функции защищено коммерческой лицензией, за получением лицензии обратитесь к разработчику.

Аргументы:

cl - указатель на структуру верхнего приложения.
_image - указатель на изображение отпечатка, 8bpp, 500ppi
_width - ширина изображения в пикселах, количество колонок
_height - высота изображения в пикселах, количество строк
[out] **_good** - на выходе содержит TRUE если изображение пригодно для обработки

Возвращает: возвращаемые функциями ошибки.

Примеры: mainform.cpp.

```
LSCLIENT_API LSC_ERROR LSC_ CheckQuality    ( LSClient *          cl,
                                              unsigned char *     _image,
                                              int                 _width,
                                              int                 _height,
                                              int *               _ quality
                                            )
```

Вычисление качества отпечатка применительно к алгоритму поиска компании Папилон. Эта функция рассчитывает качество отпечатка 0 (плохое) ... 3 (отличное). Функция может быть использована для оценки необходимого количества отпечатков для выполнения автоматического поиска по базе. Наилучшее качество поиска будет достигнуто при минимум двух отпечатках с качеством не ниже 2. Возможно подавать на поиски один отпечаток с качеством 3. В остальных случаях необходимо сканировать дополнительные отпечатки. Функция защищена коммерческой лицензией.

Аргументы:

cl - указатель на структуру верхнего приложения.
_image - указатель на изображение отпечатка, 8bpp, 500ppi
_width - ширина изображения в пикселах, количество колонок
_height - высота изображения в пикселах, количество строк
[out] *_quality* - выходное значение качества отпечатка.

Возвращает: возвращаемые функциями ошибки.

Примеры: mainform.cpp.

```
LSCLIENT_API LSC_ERROR LSC_SetLicense       ( LSClient *      cl,
                                              const char *    _path
                                            )
```

Установить файл лицензии LSSDK. Лицензия может быть привязана к серийному номеру сканера или к идентификатору оборудования на котором работает SDK. Лицензия проверяется в функциях

- LSC_CheckQuality()
- LSC_CheckEncode()
- LSC_CompressWsqPpln()
- LSC_DecompressWsqPpln()

Аргументы:

cl - указатель на структуру верхнего приложения.
_path -путь до файла лицензии, обычно называется lssdk.lic

Возвращает: возвращаемые функциями ошибки.

Примеры: mainform.cpp.

Функции сжатия/разжатия WSQ

Эти функции реализуют сжатие и разжатие изображений методом WSQ.

Функции

LSCLIENT_API LSC_ERROR LSC_CompressWsqPpln	**(LSClient ***	**cl,**
	unsigned char *	**image,**
	int	**width,**
	int	**height,**
	int	**bps,**
	int	**spp,**
	double	**compression,**
	unsigned char **	**wsq_rec,**
	int *	**wsq_len**
	)	

Сжатие изображения с использованием WSQ.

Аргументы:

[in]	*cl*	- объект библиотеки.
[in]	*image*	- указатель на буфер с изображением
[in]	*width*	- количество колонок изображения
[in]	*height*	- количество строк изображения
[in]	*bps*	- количество бит в одной компоненте, на данный момент поддерживатеся только 8.
[in]	*spp*	- количество цветовых компонент в пикселе, поддерживается только 1 для серых и 3 для RGB.
[in]	*compression*	- коэффициент сжатия, меняется от MIN_COPMPRESSION до MAX_COPMPRESSION.
[out]	*wsq_rec*	- указатель для сохранения указателя на выделенную память со сжатым изображением, память выделяется внутри функции и должна быть освобождена функцией LSC_FreeWsqPpln() после использования.
[out]	*wsq_len*	- указатель для сохранения длинны получившегося сжатого изображения.

Возвращает: возвращаемые функциями ошибки.

Примеры: mainform.cpp.

```
LSCLIENT_API LSC_ERROR LSC_ DecompressWsqPpln    ( LSClient *          cl,
                                                   unsigned char *     wsq_rec,
                                                   int                 wsq_len,
                                                   unsigned char *     image,
                                                   int *               width,
                                                   int *               height,
                                                   int                 bps,
                                                   int                 spp
                                                 )
```

Разжатие изображения WSQ.

Аргументы:

[in]	*cl*	- объект библиотеки.
[in]	*wsq_rec*	- указатель на сжатое изображение.
[in]	*wsq_len*	- длина сжатого изображения в байтах.
[out]	*image*	- указатель для сохранения указателя на выделенный буфер с разжатым изображением, память выделяется внутри функции и должна быть освобождена функцией LSC_FreeWsqPpln() после использования.
[out]	*width*	- указатель для сохранения количества колонок в изображении.
[out]	*height*	- указатель для сохранения количества строк в изображении.
[in]	*bps*	- количество бит в одной цветовой компоненте сжатого изображения, поддерживается только 8.
[in]	*spp*	- количество цветовых компонент в пикселе, 1 для серых и 3 для RGB изображений.

Возвращает: возвращаемые функциями ошибки.

Примеры: mainform.cpp.

```
LSCLIENT_API LSC_ERROR LSC_ FreeWsqPpln          ( LSClient *          cl,
                                                   unsigned char *     rec,
                                                 )
```

Освобождение памяти захваченной в функциях LSC_CompressWsqPpln() и LSC_DecompressWsqPpln().

Аргументы:

[in]	*cl*	- объект библиотеки.
[in]	*rec*	- указатель на память, которая должна быть освобождена.

Примеры: mainform.cpp.

Классы

Классы с их кратким описанием.

- **LSC_SCANNERS_LIST** - Структура для описания списка подключенных сканеров
- **LSClient** - Структура, передаваемая во все функции

Файлы

Полный список документированных файлов:

- **lsclient.h** - Заголовочный файл к библиотеке liblsclient.

Файл lsclient.h

Заголовочный файл к библиотеке **liblsclient.**

Классы

struct	LSC_SCANNERS_LIST Структура для описания списка подключенных сканеров.
struct	LSClient Структура, передаваемая во все функции.

Макросы

Ошибки процесса прокатки, допущенные оператором. Передаются в виде маски в качестве параметра в callback-функции LSClient::TakeImage().

#define	LSC_ROLL_SUCCESS 0 все хорошо.
#define	LSC_ROLL_LEFT_STRIP (1<<0) обрезан левый край отпечатка (вышел за границу призмы).
#define	LSC_ROLL_RIGHT_STRIP (1<<1) обрезан правый край отпечатка (вышел за границу призмы).
#define	LSC_ROLL_TOP_STRIP (1<<2) обрезан верхний край отпечатка (вышел за границу призмы).
#define	LSC_ROLL_WIDE_ERROR (1<<3) отпечаток прокатан не полностью.
#define	LSC_ROLL_SPEED_ERROR (1<<4) слишком быстрая прокатка.
#define	LSC_ROLL_SEQUENCE_ERROR (1<<5) неверная последовательность прокатки, рывки, скольжение...
#define	LSC_ROLL_PRINTS_ERROR (1<<6) обнаружено несколько отпечатков на призме.
#define	LSC_ROLL_FWD_SHIFT_ERROR (1<<7) проскальзывание отпечатка во время прокатки.
#define	LSC_ROLL_BCK_SHIFT_ERROR (1<<8) движение отпечатка в обратном направлении во время прокатки.
#define	LSC_ROLL_VERT_SHIFT_ERROR (1<<9)

	вертикальное смещение отпечатка во время прокатки.
#define	LSC_FSURFACE_DIRTY (1<<10) поверхность для прокатки загрязнилась, необходима очистка.
#define	LSC_PSURFACE_DIRTY (1<<11) поверхность для снятия оттисков загрязнилась, необходима очистка.
#define	LSC_SURFACE_DIRTY (1<<12) поверхность призмы сканера загрязнилась, необходима очистка.
#define	LSC_FAKE_FINGER (1<<13) Обнаружен муляж отпечатка пальца.

Определения типов

Определения типов указателей.

typedef void *	LSC_HNDL указатель на объект верхнего приложения, возвращается в callback-ах из потока прокатки.
typedef void *	ROL_HNDL указатель на внутренний объект потока прокатки, используется внутри потока.

Перечисления

enum	LSC_CAPTURE_MODE { LSC_CAPTURE_AUTO = 0, LSC_CAPTURE_CLEAN = 1,LSC_CAPTURE_AUTO_CLEAN = 2, LSC_CAPTURE_COMMAND = 3 } Режимы захвата оттисков, для функции LSCSetCaptureMode().

Команды для потока прокатки. Получаются потоком прокатки при вызове callback-функции верхнего приложения GetCommand().

enum	LSC_COMMAND {
	LSC_CMD_NOP = 0, LSC_CMD_EXIT = 1, LSC_CMD_STOP = 2,
	LSC_CMD_ROLL = 3, LSC_CMD_CLEAR = 4,
	LSC_CMD_TOUCH = 5, LSC_CMD_START = 6,
	LSC_CMD_STOP_ROLL = 7, LSC_CMD_FINGERS = 20,
	LSC_CMD_FINGER1 = 20, LSC_CMD_FINGER2 = 21,
	LSC_CMD_FINGER3= 22, LSC_CMD_FINGER4 = 23,
	LSC_CMD_FINGER5 = 24, LSC_CMD_FINGER6 = 25,
	LSC_CMD_FINGER7= 26, LSC_CMD_FINGER8 = 27,
	LSC_CMD_FINGER9 = 28, LSC_CMD_FINGER10 = 29,
	LSC_CMD_FINGER11 = 30, LSC_CMD_FINGER12 = 31,
	LSC_CMD_FINGER13 = 32, LSC_CMD_FINGER14 = 33,
	LSC_CMD_FINGER15 = 34, LSC_CMD_FINGER16 = 35,
	LSC_CMD_FINGER17 = 36, LSC_CMD_FINGER18 = 37
	}

Коды кнопок клавиатуры сканера.

enum	LSC_BUTTONS { LSC_BTN_NONE = 0 , LSC_BTN_RIGHT = 2, LSC_BTN_OK = 4, LSC_BTN_CANCEL= 8 }

Коды режимов управления светодиодами индикации сканера.

enum	LSC_LED_MODE { LSC_LED_AUTO = 0, LSC_LED_MANUAL = 1 } Режим индикации изменяется функцией LSC_SetLedMode().

Номера диодов индикации сканера.

enum	LSC_LED { LSC_GREEN_LED = 0, LSC_RED_LED = 1 }

Типы сканеров. Передается в качестве параметра в функцию LSCInit().

enum	LSC_DEVICE {
	UNKNOWN = 0, DS7 = 1, DS9 = 2, DS21 = 3,
	FX2000 = 4, DS30 = 5, DS31 = 6, DS14 = 7,
	DS40 = 8, FX2001 = 9, DS16 = 10, DS24 = 11,
	CM_VF = 12, FDS7 = 13, CM_VMW = 14, DS45 = 15,
	DS22N = 16, DS30N = 17, DS30NM = 18, DS21N = 19,
	CM_1000 = 21, IDENTIX = 22 , CM_1000P = 99
	}

Возвращаемые функциями ошибки.

enum	LSC_ERROR { , LSC_CONNECT_ERROR = 1, LSC_MEMORY_ERROR = 2, LSC_RESOURCES_ERROR = 3,LSC_READ_CONFIG_ERROR = 4, LSC_CALIBRATE_ERROR = 5, LSC_ROLL_ERROR = 6, LSC_EXIT_ERROR = 7,LSC_SLAP_IMAGE_ERROR = 8, LSC_SLAP_FINGER_NOT_FOUND = 9, LSC_ARG_ERROR = 10, LSC_LICENSE_ERROR = 11 }

Состояние потока. Возвращается в callback-е LSClient::StateChanged().

enum	LSC_STATE {
	LSC_STAT_INIT = 0, LSC_STAT_WAIT = 1,
	LSC_STAT_WAIT_FINGER = 2, LSC_STAT_ROLL = 3,
	LSC_STAT_GLUE = 4, LSC_STAT_CLEARING = 5,
	LSC_STAT_EXITED = 6, LSC_STAT_ERROR = 7
	}

Зоны прокатки

enum	LSC_ROLL_PLACE { LSC_RPLACE_AUTO = 0, LSC_RPLACE_LEFT = 1, LSC_RPLACE_RIGHT = 2 } Определение зоны прокатки.

Функции

Функции библиотеки.

LSCLIENT_API LSC_ERROR	LSCInit (LSClient *cl, LSC_DEVICE device, int num) Инициализация библиотеки прокатки.
LSCLIENT_API LSC_ERROR	LSCSetPreview (LSClient *cl, int *width, int *height) Установка размеров preview-изображения при

	прокатке.
LSCLIENT_API LSC_ERROR	LSCGetPreviewSlap (LSClient *cl, int *width, int *height) Установка размеров preview-изображения при снятии оттисков.
LSCLIENT_API LSC_ERROR	LSCGetSlapSize (LSClient *cl, int *width, int *height) Не используется.
LSCLIENT_API LSC_ERROR	LSCGetFingerSize (LSClient *cl, int *width, int *height) Не используется.
LSCLIENT_API LSC_ERROR	LSCStart (LSClient *cl) Запуск потока прокатки.
LSCLIENT_API void	LSCShutDown (LSClient *cl) Завершение потока прокатки и освобождение всех ресурсов.
LSCLIENT_API LSC_STATE	LSCGetState (LSClient *cl) Получение текущего состояния потока прокатки.
LSCLIENT_API LSC_ERROR	LSCGetError (LSClient *cl) Получение кода ошибки потока прокатки.
LSCLIENT_API char *	LSCGetVersion (LSClient *cl) Получить версию библиотеки.
LSCLIENT_API void	LSCSetFingerArea (LSClient *cl, int left, int right) Не используется.
LSCLIENT_API LSC_ERROR	LSCslapSegm (LSClient *cl, unsigned char *image, int width, int height, int num, int *left, int *top, int *fwidth, int *fheight) Сегментация контрольных оттисков.
LSCLIENT_API LSC_ERROR	LSCslapSegmAll (LSClient *cl, unsigned char *image, int width, int height, int *num, int *left, int *top, int *fwidth, int *fheight) Сегментация контрольных оттисков.
LSCLIENT_API int	LSC_GetScannersList (LSC_SCANNERS_LIST **list) Формирует список подключенных сканеров.
LSCLIENT_API void	LSC_FreeScannersList (LSC_SCANNERS_LIST *list) Освободить память выделенную в функции LSC_GetScannersList() под список сканеров.
LSCLIENT_API void	LSC_SetLedMode (LSClient *cl, LSC_LED_MODE mode) Установить режим индикации светодиодов сканера.
LSCLIENT_API LSC_ERROR	LSC_SetLed (LSClient *cl, LSC_LED led, BOOL on) Выключение/выключение светодиодов индикации сканера в ручном режиме управления.
LSCLIENT_API LSC_ERROR	LSC_SetUsbType (LSClient *cl, BOOL slow) Переключение режима передачи данных сканером по USB-шине.
LSCLIENT_API void	LSCSetCaptureMode (LSClient *cl, LSC_CAPTURE_MODE mode) Установить режим захвата оттисков.
LSCLIENT_API LSC_ERROR	LSCSetThreshold (LSClient *cl, int threshold) Установить чувствительность алгоритма определения

	наличия отпечатков на призме.
LSCLIENT_API void	LSCSetTouchMode (LSClient *cl, BOOL touch_mode) Переключение режимов оттисков/прокатки для отпечатков.
LSCLIENT_API LSC_ERROR	LSC_SendJpeg2Device (LSClient *cl, unsigned char *buffer, int length) Отправить JPEG-изображение на монитор сканера.
LSCLIENT_API LSC_ERROR	LSC_GetSizeLCD (LSClient *cl, int *width, int *height) Получить размеры экрана сканера.
LSCLIENT_API LSC_ERROR	LSC_ResetDataLCD (LSClient *cl) Сброс USB-буферов для передечи на LCD экран сканера.
LSCLIENT_API LSC_ERROR	LSC_WriteImageLCD (LSClient *cl, long size, void *image) Передать изображение на LCD экран сканера.
LSCLIENT_API LSC_ERROR	LSC_WriteMaskLCD (LSClient *cl, long AMaskSize, void *AMaskBuff) Передать палитру на LCD экран сканера.
LSCLIENT_API LSC_ERROR	LSC_SetLightLCD (LSClient *cl, int light) Установить яркость подсветки LCD экрана сканера.
LSCLIENT_API LSC_ERROR	LSC_SetLCDPower (LSClient *cl, int on) Включение/выключение питания экрана сканера.
LSCLIENT_API BOOL	LSC_IntegratedLCD (LSClient *cl) Возвращает флаг наличия в сканере интегрированного LCD экрана.
LSCLIENT_API LSC_ERROR	LSC_GetFirmware (LSClient *cl, char *firmware) Возвращается версию EEPROM firmware сканера.
LSCLIENT_API LSC_ERROR	LSC_GetAfisQuality (LSClient *cl, unsigned char *image, int width, int height, int res, int *aquality, int *vquality, float *pressure) Расчет качества отпечатка по алгоритму Папилон.
LSCLIENT_API LSC_ERROR	LSC_GetNistQuality (LSClient *cl, unsigned char *image, int width, int height, int res, int *aquality, int *vquality, float *pressure) Расчет качества отпечатка по алгоритму NIST.
LSCLIENT_API BOOL	LSC_SupportRollPlace (LSClient *cl) Проверить поддерживает ли сканер разные зоны прокатки.
LSCLIENT_API LSC_ERROR	LSC_SetRollPlace (LSClient *cl, LSC_ROLL_PLACE place) Выбор зоны прокатки.
LSCLIENT_API BOOL	LSC_EnableFakeDetection (LSClient *cl, bool _enable) Включение/выключение режима определения муляжа отпечатка.
LSCLIENT_API LSC_ERROR	LSC_CheckEncode (LSClient *cl, unsigned char *_image, int _width, int _height, BOOL *_good) Функция проверяет возможность извлечения мелких особенностей из изображения отпечатка.

LSCLIENT_API LSC_ERROR	LSC_CheckQuality (LSClient *cl, unsigned char *_image, int _width, int _height, int *_quality) Вычисление качества отпечатка применительно к алгоритму поиска компании Папилон.
LSCLIENT_API LSC_ERROR	LSC_SetLicense (LSClient *cl, const char *_path) Установить файл лицензии LSSDK.
LSCLIENT_API LSC_ERROR	LSC_CompressWsqPpln (LSClient *cl, unsigned char *image, int width, int height, int bps, int spp, double compression, unsigned char **wsq_rec, int *wsq_len) Сжатие изображения с сипользованием WSQ.
LSCLIENT_API LSC_ERROR	LSC_DecompressWsqPpln (LSClient *cl, unsigned char *wsq_rec, int wsq_len, unsigned char **image, int *width, int *height, int bps, int spp) Разжатие изображения WSQ.
LSCLIENT_API LSC_ERROR	LSC_FreeWsqPpln (LSClient *cl, unsigned char *rec) Освобождение памяти захваченной в функциях LSC_CompressWsqPpln() и LSC_DecompressWsqPpln().

Подробное описание

Заголовочный файл к библиотеке liblsclient. См. определение в файле lsclient.h

Перечисления

enum LSC_CAPTURE_MODE

Режимы захвата оттисков, для функции LSCSetCaptureMode().

Элементы перечислений:

LSC_CAPTURE_AUTO	Автоматический захват после стабилизации изображения.
LSC_CAPTURE_CLEAN	Изображение захватывается после того как пальцы будут убраны с призмы, выбирается последнее наиболее полное изображение.
LSC_CAPTURE_AUTO_CLEAN	Автоматический захват после стабилизации изображения, для возобновления сканирования пальцы необходимо будет убрать с призмы и приложить снова. В этом режиме не будет повторных захватов пока пальцы не уберут с призмы и не приложат снова
LSC_CAPTURE_COMMAND	Захват изображения отпечатка только после подачи команды LSC_COMMAND::LSC_CMD_TOUCH.

См. определение в файле lsclient.h строка 187.

```
00188 {
00189         /** Автоматический захват после стабилизации изображения. */
00190         LSC_CAPTURE_AUTO = 0,
```

```
00191
00192         /** Изображение захватывается после того как пальцы будут
                  убраны с призмы, выбирается последнее наиболее полное
                  изображение. */
00193         LSC_CAPTURE_CLEAN = 1,
00194
00195         /** Автоматический захват после стабилизации изображения,
                  для возобновления сканирования пальцы
00196          * необходимо будет убрать с призмы и приложить снова.
00197          * В этом режиме не будет повторных захватов пока пальцы
                не уберут с призмы и не приложат снова*/
00198         LSC_CAPTURE_AUTO_CLEAN = 2,
00199
00200         /** Захват изображения отпечатка только после подачи
                  команды LSC_COMMAND::LSC_CMD_TOUCH */
00201         LSC_CAPTURE_COMMAND = 3,
00202 } LSC_CAPTURE_MODE;
```

Исходный текст файла lsclient.h

```
00001 #ifndef lsclient_h
00002 #define lsclient_h
00003
00004 #if defined(__WIN32__) || defined(WIN32)
00005
00006 #ifdef LSCLIENT_EXPORTS
00007 #define LSCLIENT_API __declspec(dllexport)
00008 #else
00009 #define LSCLIENT_API __declspec(dllimport)
00010 #endif
00011 #include <windows.h>
00012 #else
00013 #define LSCLIENT_API
00014 #endif
00015
00016 #ifdef __cplusplus
00017 extern "C"
00018 {
00019 #endif
00020 /** \file lsclient.h
00021  *\brief Заголовочный файл к библиотеке liblsclient.
00022  */
00023
00024 /*! \mainpage Библиотека получения прокатанных отпечатков и оттисков.
00025  *
00026  * \author АО Папилон. 2015г.
00027  *
00028  * \section liblsclient_contetn Содержание.
00029  * - \ref cont
00030  * - \ref comp
00031  * - \ref scanners
00032  * - \ref drivers
```

```
00033  * - \ref working
00034  */
00035
00036 /**\page  cont Состав библиотеки.
00037  * Библиотека состоит из следующих компонентов:
00038  *      - динамический модуль lsclient.dll, либо lsclient_rtm.dll,
00039  *        в зависимости от поддерживаемых устройств.
00040  *      - заголовочный файл lsclient.h
00041  *      - пример использования библиотеки lsclient_test
00042  *      - пакет DKMS для установки драйверов сканеров(ОС Linux)
00043  *      - пакеты с драйверами для сканеров
00044  */
00045
00046 /** \page comp Совместимость.
00047  * Библиотека собрана на дистрибутивах
00048  * - Linux X86 RHEL5.5, gcc-4.1.2.
00049  * - Linux X64 RHEL6.3, gcc-4.4.6.
00050  * - Windows X86 MSVC2010
00051  */
00052
00053 /**\page drivers Драйвера.
00054  *
00055  *\section Linux Linux.
00056  *\n
00057  * Сам DKMS так же поставляется в виде rpm-пакета.
00058  * Порядок установки драйверов следующий:
00059  * - установить пакет dkms командой rpm -i dkms-2.0.13-1.noarch.rpm
00060  * - установить драйвер командой:
00061  *   rpm -i ds22_usb-4.2.2-3dkms.noarch.rpm (для сканеров DS21,DS22)
00062  *
00063  *\n
00064  * На компьютере должны быть установлены исходные тексты ядра либо
заголовочные файлы ядра.
00065  *\n
00066  *\n
00067  * В результате этих действий DKMS скомпилирует драйвер и установит
его в каталог
00068  * /lib/modules/`uname -r`/kernel/drivers/misc.
00069  *\n
00070  *\n
00071  * Файл загрузки драйвера и создания специальных файлов в каталоге
/dev/  ds22_usb будет установлен в
00072  * каталоге /etc/rc.d/init.d/ и включен в 3,4,5 runlevel-ы.
00073  *\n
00074  *\n
00075  * При загрузке компьютеры сервис ds22_usb загрузит драйвер сканера и
создаст необходимые файлы в каталоге /dev/usb.
00076  *
00077  *\n
00078  *\n
00079  *\section Windows Windows.
00080  *
```

```
00081  * После подключения сканера к компьютеру ОС запросит драйвера для
сканера - необходимо указать каталог,
00082  * в котором находятся драйвера для соответствующей модели сканера,
например Drivers/DS22/.
00083  * Если санер поставляется со встроенным LCD-экраном, то после
установки драйверов для сканера будет выдан запрос
00084  * на драйвера для LCD-экрана, необходимо указать  каталог
Drivers/LCD.
00085  */
00086
00087 /** \page scanners Поддерживаемые сканеры.
00088  * Библиотекой поддерживаются следующие сканеры, выпускаемые фирмой
Папилон:
00089  *- Сканер для получения оттисков отпечатков DS21/DS22.
00090  *- Сканер для получения оттисков отпечатков DS20/FX2000
00091  *- Сканер для получения прокатанных отпечатков и контрольных оттисков
DS9.
00092  *- Сканер для получения прокатанных отпечатков и контрольных оттисков
DS30.
00093  *- Сканер для получения прокатанных отпечатков и контрольных оттисков
DS30M.
00094  *- Сканер для получения прокатанных отпечатков и контрольных оттисков
DS31.
00095  *- Сканер для получения прокатанных отпечатков, контрольных оттисков
и изображений ладоней DS7.
00096  *- Сканер для получения прокатанных отпечатков, контрольных оттисков
и изображений ладоней DS10.
00097  *- Сканер для получения прокатанных отпечатков, контрольных оттисков
и изображений ладоней DS14.
00098  *- Сканер для получения прокатанных отпечатков, контрольных оттисков
и изображений ладоней DS40.
00099  *- Сканер для получения прокатанных отпечатков, контрольных оттисков
и изображений ладоней DS40M.
00100  *- Сканер для получения прокатанных отпечатков, контрольных оттисков
и изображений ладоней DS45.
00101  *- Сканер для получения прокатанных отпечатков, контрольных оттисков
и изображений ладоней DS45M.
00102  *- Сканер для получения прокатанных отпечатков, контрольных оттисков
DS30N.
00103  *- Сканер для получения прокатанных отпечатков, контрольных оттисков
DS30NM.
00104  *- Сканер для получения прокатанных отпечатков DS22N.
00105  *- Сканер для получения прокатанных отпечатков DS21S(Orion) с
возможностью определения муляжа пальца.
00106  *
00107  * Порядок установки драйверов описан на странице \ref drivers.
00108  */
00109
00110 /** \page working Порядок работы с библиотекой.
00111  * приложение, использующее данную библиотеку должно:
00112  * - создать объект LSClient
00113  * - инициализировать следующие поля:
00114  *      - LSClient::Client
00115  *      - LSClient::debug_level
00116  *      - LSClient::GetCommand
```

```
00117  *      - LSClient::DrawPreview
00118  *      - LSClient::ClearPreview
00119  *      - LSClient::TakeImage
00120  *      - LSClient::GetWorkDir
00121  *      - LSClient::ProcessButton
00122  *      - LSClient::InitDone
00123  *      - LSClient::StateChanged
00124  *      - LSClient::DrawLCD
00125  *
00126  * - вызывать функцию LSCInit().
00127  * - запустить поток прокатки функцией LSCStart().
00128  * - дождаться окончания инициализации по смене статуса с
LSC_STATE::LSC_STAT_INIT на другой,
00129  *   либо после вызова callback-функции LSClient::InitDone().
00130  * - начать работу с потоком прокатки отдавая \ref commands "команды".
00131  *   через callback-функцию LSClient::GetCommand().
00132  * - для включения алгоритма слежения за отпечатком и прокатки
необходимо подать
00133  *   команду LSC_COMMAND::LSC_CMD_ROLL.
00134  * - после завершения работы с потоком необходимо либо остановить его
командой
00135  *   LSC_COMMAND::LSC_CMD_STOP(если планируется дальнейшее
использование сканера),
00136  *   либо завершить функцией LSCShutdown().
00137  * - после остановки потока командой LSC_COMMAND::LSC_CMD_STOP
00138  *   можно возобновить его работу подав команду
LSC_COMMAND::LSC_CMD_ROLL.
00139  * - после завершения функцией LSCShutdown() возобновить работу
данного потока невозможно,
00140  *   в этом случае необходимо заново инициализировать объект LSClient
и сам поток прокатки.
00141  */
00142
00143 /** @name  Команды для потока прокатки.
00144  *
00145  * Получаются потоком прокатки при вызове callback-функции верхнего
приложения GetCommand().
00146  */
00147 /*@{*/
00148 /** \defgroup commands Команды для потока прокатки.*/
00149 /*@{*/
00150 typedef enum LSC_COMMAND
00151 {
00152         LSC_CMD_NOP = 0, /*!< пустая команда  */
00153         LSC_CMD_EXIT = 1, /*!< завершить поток прокатки */
00154         LSC_CMD_STOP = 2, /*!< остановить режим ожидания отпечатка и
прокатки */
00155         LSC_CMD_ROLL = 3, /*!< запустить режим ожидания отпечатка и
прокатки */
00156         LSC_CMD_CLEAR = 4, /*!< компенсировать грязь на призме */
00157         LSC_CMD_TOUCH = 5, /*!< захватить оттиск прямо сейчас */
00158         LSC_CMD_START = 6, /*!< не использовать */
00159         LSC_CMD_STOP_ROLL = 7, /*!< не использовать */
```

```
00160
00161         /*! Команды переключения текущего отпечатка.
00162          * Для пальцев 0-9 -прокатка, для остальных - оттиск
00163          */
00164         LSC_CMD_FINGERS = 20, LSC_CMD_FINGER1 = 20, /*!< Правый
большой. */
00165         LSC_CMD_FINGER2 = 21, /*!< Правый указательный. */
00166         LSC_CMD_FINGER3 = 22, /*!< Правый средний. */
00167         LSC_CMD_FINGER4 = 23, /*!< Правый безымянный. */
00168         LSC_CMD_FINGER5 = 24, /*!< Правый мизинец. */
00169         LSC_CMD_FINGER6 = 25, /*!< Левый большой. */
00170         LSC_CMD_FINGER7 = 26, /*!< Левый указательный. */
00171         LSC_CMD_FINGER8 = 27, /*!< Левый средний. */
00172         LSC_CMD_FINGER9 = 28, /*!< Левый безымянный. */
00173         LSC_CMD_FINGER10 = 29, /*!< Левый мизинец. */
00174         LSC_CMD_FINGER11 = 30, /*!< Левые 4 оттиска. */
00175         LSC_CMD_FINGER12 = 31, /*!< Левый большой оттиск. */
00176         LSC_CMD_FINGER13 = 32, /*!< Правый большой оттиск. */
00177         LSC_CMD_FINGER14 = 33, /*!< Правые 4 оттиска. */
00178         LSC_CMD_FINGER15 = 34, /*!< Левая ладонь. */
00179         LSC_CMD_FINGER16 = 35, /*!< Правая ладонь. */
00180         LSC_CMD_FINGER17 = 36, /*!< Левая ладонь, ребро. */
00181         LSC_CMD_FINGER18 = 37, /*!< Правая ладонь, ребро. */
00182 } LSC_COMMAND;
00183 /*@}*/
00184 /*@}*/
00185
00186 /**Режимы захвата оттисков, для функции LSCSetCaptureMode().*/
00187 typedef enum LSC_CAPTURE_MODE
00188 {
00189         /** Автоматический захват после стабилизации изображения. */
00190         LSC_CAPTURE_AUTO = 0,
00191
00192         /** Изображение захватывается после того как пальцы будут
убраны с призмы, выбирается последнее наибоолее полное изображение. */
00193         LSC_CAPTURE_CLEAN = 1,
00194
00195         /** Автоматический захват после стабилизации изображения, для
возобновления сканирования пальцы
00196          * необходимо будет убрать с призмы и приложить снова.
00197          * В этом режиме не будет повторных захватов пока пальцы не
уберут с призмы и не приложат снова*/
00198         LSC_CAPTURE_AUTO_CLEAN = 2,
00199
00200         /** Захват изображения отпечатка только после подачи команды
LSC_COMMAND::LSC_CMD_TOUCH */
00201         LSC_CAPTURE_COMMAND = 3,
00202 } LSC_CAPTURE_MODE;
00203
00204 /*! @name Коды кнопок клавиатуры сканера.  */
00205 /*@{*/
00206 /** \defgroup buttons Коды кнопок клавиатуры сканера.   */
```

```
00207 /*@{*/
00208 typedef enum LSC_BUTTONS
00209 {
00210         /**< ничего не нажато */
00211         LSC_BTN_NONE = 0,
00212
00213         /**< нажата стрелка влево */
00214         LSC_BTN_LEFT = 1,
00215
00216         /** нажата стрелка вправо */
00217         LSC_BTN_RIGHT = 2,
00218
00219         /** нажата кнопка Ок */
00220         LSC_BTN_OK = 4,
00221
00222         /** нажата кнопка Cancel */
00223         LSC_BTN_CANCEL = 8,
00224
00225 } LSC_BUTTONS;
00226 /*@}*/
00227 /*@}*/
00228
00229 /*! @name Коды режимов управления светодиодами индикации сканера. */
00230 /*@{*/
00231 /** \defgroup led_mode Коды режимов управления светодиодами индикации
сканера. */
00232 /*@{*/
00233 /** Режим индикации изменяется функцией LSC_SetLedMode() */
00234 typedef enum LSC_LED_MODE
00235 {
00236         /** Автоматический режим индикации. */
00237         LSC_LED_AUTO = 0,
00238
00239         /** Ручной режим, индикация управляется функцией LSC_SetLed()
*/
00240         LSC_LED_MANUAL = 1,
00241
00242 } LSC_LED_MODE;
00243 /*@}*/
00244 /*@}*/
00245
00246 /*! @name Номера диодов индикации сканера. */
00247 /*@{*/
00248 /** \defgroup led_dontrol Номера диодов индикации сканера. */
00249 /*@{*/
00250 typedef enum LSC_LED
00251 {
00252         /** Зеленый диод */
00253         LSC_GREEN_LED = 0,
00254
00255         /** Красный диод */
00256         LSC_RED_LED = 1,
```

```
00257
00258 } LSC_LED;
00259 /*@}*/
00260 /*@}*/
00261
00262 /*! @name Определения типов указателей.*/
00263 /*@{*/
00264 typedef void * LSC_HNDL;    //!< указатель на объект верхнего
приложения, возвращается в callback-ах из потока прокатки.
00265 typedef void * ROL_HNDL;    //!< указатель на внутренний объект потока
прокатки, используется внутри потока.
00266 /*@}*/
00267
00268 /*! @name Ошибки процесса прокатки, допущенные оператором.
00269  *
00270  * Передаются в виде маски в качестве параметра в callback-функции
LSClient::TakeImage().*/
00271 /*@{*/
00272 /** \defgroup roll_errors Ошибки процесса прокатки, допущенные
оператором. */
00273 /*@{*/
00274 #define LSC_ROLL_SUCCESS                0       /*!<\brief все хорошо
*/
00275 #define LSC_ROLL_LEFT_STRIP             (1<<0)  /*!<\brief обрезан
левый кай отпечатка(вышел за границу призмы) */
00276 #define LSC_ROLL_RIGHT_STRIP            (1<<1)  /*!<\brief обрезан
правый кай отпечатка(вышел за границу призмы) */
00277 #define LSC_ROLL_TOP_STRIP              (1<<2)  /*!<\brief обрезан
верхний кай отпечатка(вышел за границу призмы) */
00278 #define LSC_ROLL_WIDE_ERROR             (1<<3)  /*!<\brief отпечаток
прокатан не полностью */
00279 #define LSC_ROLL_SPEED_ERROR            (1<<4)  /*!<\brief слишком
быстрая прокатка */
00280 #define LSC_ROLL_SEQUENCE_ERROR         (1<<5)  /*!<\brief неверная
последовательность прокатки, рывки, скольжение... */
00281 #define LSC_ROLL_PRINTS_ERROR           (1<<6)  /*!<\brief обнаружено
несколько отпечатков на призме */
00282 #define LSC_ROLL_FWD_SHIFT_ERROR    (1<<7)  /*!<\brief проскальзывание
отпечатка во время прокатки */
00283 #define LSC_ROLL_BCK_SHIFT_ERROR    (1<<8)  /*!<\brief движение
отпечатка в обратном направлении во время прокатки */
00284 #define LSC_ROLL_VERT_SHIFT_ERROR   (1<<9)  /*!<\brief вертикальное
смещение отпечатка во время прокатки */
00285 #define LSC_FSURFACE_DIRTY          (1<<10) /*!<\brief поверхность для
прокатки загрязнилась, необходима очистка */
00286 #define LSC_PSURFACE_DIRTY          (1<<11) /*!<\brief поверхность для
снятия оттисков загрязнилась, необходима очистка */
00287 #define LSC_SURFACE_DIRTY           (1<<12) /*!<\brief поверхность
призмы сканера загрязнилась, необходима очистка */
00288 #define LSC_FAKE_FINGER                         (1<<13) /*!<\brief
Обнаружен муляж отпечатка пальца */
00289 /*@}*/
00290 /*@}*/
00291
00292 /*! @name Типы сканеров.
```

```
00293  *
00294  * Передается в качестве параметра в функцию LSCInit().*/
00295 /*@{*/
00296 /** \defgroup device Типы сканеров.  */
00297 /*@{*/
00298 typedef enum LSC_DEVICE
00299 {
00300         UNKNOWN = 0, /*!< Неизвестный сканер */
00301         DS7 = 1, /*!< Ладонный сканер ДС7 */
00302         DS9 = 2, /*!< Пальцевый сканер ДС9 IEEE1394 или PCI */
00303         DS21 = 3, /*!< Пальцевый сканер ДС22 или ДС21 */
00304         FX2000 = 4, /*!< Пальцевый сканер FX2000 */
00305         DS30 = 5, /*!< Пальцевый сканер с контрольными оттисками ДС30
*/
00306         DS31 = 6, /*!< Пальцевый сканер с контрольными оттисками
ДС31(2 призмы) */
00307         DS14 = 7, /*!< Ладонный сканер ДС14-USB */
00308         DS40 = 8, /*!< DS40 palm scanner */
00309         FX2001 = 9, /*!< Fingerprint FX2000 scanner with new sensor */
00310         DS16 = 10, /*!< Palm scanner DS16 + DS22 for rolling */
00311         DS24 = 11, /*!< Сканер отпечатков DS24, 1000дпи */
00312         CM_VF = 12, /*!< Cross Match Verifier. */
00313         FDS7 = 13, /*!< DS7 IEEE-1394. */
00314         CM_VMW = 14, /*!<Cross Match VMW сканер. */
00315         DS45 = 15, /*!<Сканер DS45, DS45M. */
00316         DS22N = 16, /*!<Сканер DS22N. */
00317         DS30N = 17, /*!<Сканер DS30N. */
00318         DS30NM = 18, /*!<Сканер DS30NM. */
00319         DS21N = 19, /*!<Сканер DS21N. */
00320         CM_1000 = 21, /*!<Сканер Cross Match 1000p с прокаткой
Crossmatch.*/
00321         IDENTIX = 22, /*!<Сканер Identix*/
00322         DS21S = 23, /*!Сканер  DS21S(Orion)*/
00323         CM_1000P = 99, /*!<Сканер Cross Match 1000p с прокаткой
Папилон.*/
00324 } LSC_DEVICE;
00325 /*@}*/
00326 /*@}*/
00327
00328 /** Структура для описания списка подключенных сканеров. */
00329 typedef struct LSC_SCANNERS_LIST
00330 {
00331                 LSC_DEVICE device; /*!< Модель сканера */
00332                 int num; /*!< Количество подключенных сканеров данной
модели. */
00333 } LSC_SCANNERS_LIST;
00334
00335 /*! @name Возвращаемые функциями ошибки.*/
00336 /*@{*/
00337 /** \defgroup returns Возвращаемые функциями ошибки. */
00338 /*@{*/
00339 typedef enum LSC_ERROR
```

```
00340 {
00341         LSC_NO_ERROR = 0, /*! No errors */
00342         LSC_CONNECT_ERROR = 1, /*! Ошибка соединения со сканером */
00343         LSC_MEMORY_ERROR = 2, /*! Мало памяти */
00344         LSC_RESOURCES_ERROR = 3, /*! Ошибка выделения ресурсов IPC */
00345         LSC_READ_CONFIG_ERROR = 4, /*! Ошибка чтения конфигурации
сканера */
00346         LSC_CALIBRATE_ERROR = 5, /*! Сканер не откалиброван */
00347         LSC_ROLL_ERROR = 6, /*! Ошибка прокатки */
00348         LSC_EXIT_ERROR = 7, /*! Поток прокатки закрыт */
00349         LSC_SLAP_IMAGE_ERROR = 8, /*!< Ошибка разделения контрольных
оттисков.  */
00350         LSC_SLAP_FINGER_NOT_FOUND = 9, /*!< Ошибка разделения
контрольных оттисков, не все отпечатки обнаружены. */
00351         LSC_ARG_ERROR = 10, /*!< Неверные аргументы функции. */
00352         LSC_LICENSE_ERROR = 11, /*!< Неверная лицензия*/
00353 } LSC_ERROR;
00354 /*@}*/
00355 /*@}*/
00356
00357 /*! @name Состояние потока.
00358  *
00359  * Возвращается в callback-е LSClient::StateChanged(). */
00360 /*@{*/
00361 /** \defgroup state  Состояние потока прокатки.  */
00362 /*@{*/
00363 typedef enum LSC_STATE
00364 {
00365         LSC_STAT_INIT = 0, /*!< инициализация */
00366         LSC_STAT_WAIT = 1, /*!< ожидание команды без слежения за
отпечатком и прокатки */
00367         LSC_STAT_WAIT_FINGER = 2, /*!< слежение за отпечатком  */
00368         LSC_STAT_ROLL = 3, /*!< собственно процесс прокатки */
00369         LSC_STAT_GLUE = 4, /*!< обработка полученного после прокатки
изображения */
00370         LSC_STAT_CLEARING = 5, /*!< компенсация грязи на призме */
00371         LSC_STAT_EXITED = 6, /*!< поток завершился */
00372         LSC_STAT_ERROR = 7, /*!< произошла какая-либо ошибка */
00373 } LSC_STATE;
00374 /*@}*/
00375 /*@}*/
00376
00377 /*! @name Зоны прокатки
00378  *
00379  */
00380 /*@{*/
00381 /** \defgroup roll_place Зоны прокатки.
00382  *
00383  */
00384 /*@{*/
00385 /**Определение зоны прокатки*/
00386 typedef enum LSC_ROLL_PLACE
```

```
00387 {
00388         /** Автоматический выбор зоны прокатки.
00389          *   Правая рука прокатывается справа, левая рука
прокатывается слева.
00390          */
00391         LSC_RPLACE_AUTO = 0,
00392
00393         /**Прокатка выполняется в левой части призмы*/
00394         LSC_RPLACE_LEFT = 1,
00395
00396         /**Прокатка выполняется в правой части призмы*/
00397         LSC_RPLACE_RIGHT = 2,
00398 } LSC_ROLL_PLACE;
00399 /*@}*/
00400 /*@}*/
00401
00402 /** \defgroup lsclient Структура LSClient.*/
00403 /*@{*/
00404 /** \brief Структура, передаваемая во все \ref func "функции".
00405  *
00406  *
00407  * Выделяется  и инициализируется в верхней программе,
00408  * затем передается для последующей инициализации в функцию ::LSCInit
.
00409  */
00410 typedef struct LSClient
00411 {
00412
00413                 /*!\brief указатель на какой-либо объект верхней
программы.
00414                  *
00415                  * Выделяется и инициализируется верхней программой,
потоком прокатки не модифицируется.
00416                  * Передается в качестве параметра во всех \ref
callback "callback-ax" из
00417                  * потока прокатки в верхнюю программу.
00418                  */
00419                 LSC_HNDL Client;
00420
00421                 /*!\brief указатель на объект потока прокатки.
00422                  *
00423                  * Инициализируется и используется внутри потока
прокатки.
00424                  */
00425                 ROL_HNDL Roll;
00426
00427                 /*!\brief Уровень отладочных сообщений.
00428                  *
00429                  * Задается в верхней программе 0 - нет сообщений,
максимум 9.
00430                  */
00431                 int debug_level;
00432
```

```
00433                 /** @name Callback-функции, верхнего приложения.
00434                  *
00435                  * Инициализируются в верхней программе. Вызываются из
потока прокатки.
00436                  * Должны выполняться быстро, все обращения из них к
UI делать через события.
00437                  */
00438                 /*@{*/
00439                 /** \defgroup callback Callback-функции верхнего
приложения. */
00440                 /*@{*/
00441
00442                 /*!\brief Получение потоком прокатки команды от
верхней программы.
00443                  *
00444                  * Функция вызывается внутри потока прокатки для
получения команды от внешнего приложения.
00445                  * \param Cl - указатель на объект верхнего
приложения.
00446                  * \return команду для потока прокатки.
00447                  */
00448                 LSC_COMMAND (*GetCommand)(LSC_HNDL Cl);
00449
00450                 /**\brief Вывод preview на экран.
00451                  *
00452                  * Когда отпечаток обнаружен на призме прибора,
подготавливается изображение для показа оператору
00453                  * на экране. Эта функция вызывается из потока
прокатки для передачи верхнему приложению подготовленного
00454                  * preview-изображения.
00455                  * \param Cl - указатель на объект верхнего
приложения.
00456                  * \param finger_num - номер текущего отпечатка,
установленного одной из комманд #LSC_CMD_FINGERS.
00457                  * \param Buf - указатель на буфер с изображением,
буфер глубиной 8bpp(256 градаций серого),
00458                  * первый байт - левый верхний пиксель изображения.
00459                  * \param W - ширина изображения(количество колонок).
00460                  * \param H - высота изображения(количество строк).
00461                  * \return 0 - в случае успеха.
00462                  */
00463                 int (*DrawPreview)(LSC_HNDL Cl, int finger_num,
unsigned char *Buf, int W, int H);
00464
00465                 /**\brief Очистка окна preview.
00466                  *
00467                  * Вызывается когда отпечаток убран с призмы, либо
процесс прокатки завершен.
00468                  * \param Cl - указатель на объект верхнего
приложения.
00469                  * \return 0 - в случае успеха.
00470                  */
00471                 int (*ClearPreview)(LSC_HNDL Cl);
00472
```

```
00473                  /**\brief Передача готового изображения отпечатка из
потока прокатки в верхнее приложение.
00474                   *
00475                   * Вызывается когда процесс прокатки завершен.
00476                   * \param Cl - указатель на объект верхнего
приложения.
00477                   * \param finger_num - номер отпечатка
00478                   * \param Buf - указатель на буфер с изображением
отпечатка, освобождается в потоке прокатки.
00479                   * Глубина буфера 8bpp(256 градаций серого),
разрешение 500ppi, первый байт - верхний левый пиксел изображения.
00480                   * Может быть NULL, в этом случае будет установлена
переменная roll_errors в соответствующее значение ошибки прокатки.
00481                   * \param W - ширина изображения(количество колонок).
00482                   * \param H - высота изображения(количество строк).
00483                   * \param roll_errors - маска \ref roll_errors "ошибок
прокатки".
00484                   * В случае критических ошибок параметр Buf
00485                   * возвращается как NULL.
00486                   * \return 0 - в случае успеха.
00487                   */
00488                  int (*TakeImage)(LSC_HNDL Cl, int finger_num, unsigned
char *Buf, int W, int H, unsigned int roll_errors);
00489
00490                  /**\brief Получить путь к рабочему каталогу.
00491                   *
00492                   * Вызывается из потока прокатки для получения
каталога
00493                   * в котором можно сохранять временные файлы.
00494                   * \param Cl - указатель на объект верхнего
приложения.
00495                   * \return path Путь к каталогу.
00496                   */
00497                  char *(*GetWorkDir)(LSC_HNDL Cl);
00498
00499                  /**\brief Передача кода кнопки, нажатой на сканере.
00500                   *
00501                   * Вызывается из потока прокатки для передачи
информации о нажатии какой-либо кнопки на клавиатуре сканера.
00502                   * \param Cl - указатель на объект верхнего
приложения.
00503                   * \param button - нажатая кнопка.
00504                   * \return 0 в случае успеха.
00505                   */
00506                  int (*ProcessButton)(LSC_HNDL Cl, LSC_BUTTONS button);
00507
00508                  /**\brief Уведомление верхнего приложения о завершении
инициализации потока прокатки.
00509                   *
00510                   * Вызывается из потока прокатки после завершения
процедуры
00511                   * тестирования и инициализации сканера.
00512                   * После этого можно начинать работу с потоком
```

```
00513                  * прокатки - инициализация preview LSCSetPreview(),
LSCGetPreviewSlap(),
00514                  * передавать \ref commands "команды" в поток через
callback LSClient::GetCommand().
00515                  * \param Cl - указатель на объект верхнего
приложения.
00516                  */
00517                 void (*InitDone)(LSC_HNDL Cl);
00518
00519                 /**\brief Уведомление верхнего приложения о изменении
состояния потока прокатки.
00520                  *
00521                  * Вызывается из потока прокатки после изменения
состояния потока прокатки.
00522                  * \param Cl - указатель на объект верхнего
приложения.
00523                  * \param state - новое состояние потока прокатки.
00524                  */
00525                 void (*StateChanged)(LSC_HNDL Cl, LSC_STATE state);
00526
00527                 /**\brief Уведомление верхнего приложения о
возможности отправить картинку на USB LCD.
00528                  *
00529                  * Вызывается из потока прокатки во время перерыва
передачи данных со сканера по USB-шине.
00530                  * Используется только со сканерами, оборудованными
LCD монитором(DS30M и DS15M).
00531                  * Если DrawLCD не инициализирован(равен NULL),
00532                  * то поток прокатки не будет вызывать этот callback.
00533                  * \param Cl - указатель на объект верхнего
приложения.
00534                  */
00535                 void (*DrawLCD)(LSC_HNDL Cl);
00536
00537                 /**\brief Уведомление верхнего приложения о том что
накоплено достаточное
00538                  * количество кадров для построения финального
изображения.
00539                  *
00540                  * Эта функция вызывается из потока прокатки, когда
накоплено достаточное
00541                  * количество кадров для построения финального
изображения, в случае прокатки отпечатка
00542                  * оператор должен сам решить, когда прокатка
выполнена полностью.
00543                  * Если RemoveFinger не инициализирован(равен NULL),
00544                  * то поток прокатки не будет вызывать этот callback.
00545                  *
00546                  * \param Cl - указатель на объект верхнего
приложения.
00547                  */
00548                 void (*RemoveFinger)(LSC_HNDL Cl);
00549
00550                 /**\brief Сканирование всех отпечатков закончено.
00551                  *
```

```
00552                  * Вызывается из потока сканирование когда
отсканированы все отпечатки,
00553                  * используется только для сканера CorrMatch VMW.
00554                  */
00555                 void (*ScanComplete)(LSC_HNDL Cl);
00556
00557                 /**\brief Передает в верхнее приложение статистику
сканирования.
00558                  *
00559                  *  Эти данные могут быть сохранены в отдельные
текстовые записи для
00560                  *  дальнейшего анализа ошибок прокатки оператора,
ошибок работы сканера.
00561                  *  Callback вызывается после
00562                  * - LSC_CMD_START и LSC_CMD_ROLL команд для
сохранения информации о
00563                  * медели, настройк есканера и версиях программного
обеспечения.
00564                  * - TakeImage callback для сохранения информации о
параметрах сканирования.
00565                  *
00566                  * \param Cl - указатель на объект верхнего
приложения.
00567                  * \param data - текстовая строка, заканчивающаяся
нулем
00568                  * \param tag - номер текстового тега записи
00569                  * \param multiple - флаг множественного тега:
00570                  *   - 1 - множественный, данные должны быть добавлены
к записям с таким же номером тега
00571                  *   - 0 - единичный, данные должны заместить записи с
таким же номером тега
00572                  */
00573                 void (*AddScanStat)(LSC_HNDL Cl, char* data, int tag,
int multiple);
00574                 /*@}*/
00575                 /*@}*/
00576 } LSClient;
00577 /*@}*/
00578
00579 /** @name Функции библиотеки.    */
00580 /*@{*/
00581 /** \defgroup func Функции библиотеки.*/
00582 /*@{*/
00583
00584 /** \brief Инициализация библиотеки прокатки.
00585  *
00586  * Функция захватывает необходимые для работы ресурсы и создает
соединение со сканером.
00587  * \param cl - Указатель на структуру LSClient.
00588  * \param device - Тип сканера, с которым будет работать данный поток.
00589  * \param num - номер сканера, начиная с нуля.
00590  * \return \ref returns "код ошибки".
00591  *
00592  */
```

```
00593 LSCLIENT_API LSC_ERROR LSCInit(LSClient *cl, LSC_DEVICE device, int
num);
00594
00595 /** \brief Установка размеров preview-изображения при прокатке.
00596  *
00597  * Функция вычисляет параметры preview изображения и возвращает
размеры preview
00598  * в аргументах width и height. В дальнейшем именно с этими размерами
будет вызываться
00599  * callback-функция LSClient::DrawPreview().
00600  * Данную функцию можно вызывать только после окончания процесса
инициализации.
00601  * Окончание инициализации сигнализируется сменой статуса потока
прокатки с
00602  * LSC_STAT_INIT на любой другой и вызовом callback-функции
LSClient::InitDone().
00603  * \param cl - Указатель на структуру LSClient.
00604  * \param width - ширина изображения
00605  * \param height - высота изображения
00606  * \return \ref returns "код ошибки".
00607  */
00608 LSCLIENT_API LSC_ERROR LSCSetPreview(LSClient *cl, int* width, int*
height);
00609
00610 /** \brief Установка размеров preview-изображения при снятии оттисков.
00611  *
00612  * Функция вычисляет параметры preview изображения при снятии
контрольных оттисков и
00613  * возвращает размеры preview в аргументах width и height.
00614  * В дальнейшем именно с этими размерами будет вызываться
00615  * callback-функция LSClient::DrawPreview().
00616  * Данную функцию можно вызывать только после окончания процесса
инициализации.
00617  * Окончание инициализации сигнализируется сменой статуса потока
прокатки с
00618  * LSC_STAT_INIT на любой другой и вызовом callback-функции
LSClient::InitDone().
00619  * \param cl - Указатель на структуру LSClient.
00620  * \param width - ширина изображения
00621  * \param height - высота изображения
00622  * \return \ref returns "код ошибки".
00623  */
00624 LSCLIENT_API LSC_ERROR
00625 LSCGetPreviewSlap(LSClient *cl, int* width, int* height);
00626
00627 /** \brief Не используется.
00628  *
00629  *
00630  * \param cl - Указатель на структуру LSClient.
00631  * \param width - Указатель на ширину изображения.
00632  * \param height - Указатель на высоту изображения.
00633  * \return \ref returns "код ошибки".
00634  */
```

```
00635 LSCLIENT_API LSC_ERROR LSCGetSlapSize(LSClient *cl, int* width, int*
height);
00636
00637 /** \brief Не используется.
00638  *
00639  *
00640  * \param cl - Указатель на структуру LSClient.
00641  * \param width - Указатель на ширину изображения.
00642  * \param height - Указатель на высоту изображения.
00643  * \return \ref returns "код ошибки".
00644  */
00645 LSCLIENT_API LSC_ERROR LSCGetFingerSize(LSClient *cl, int* width, int*
height);
00646
00647 /** \brief Запуск потока прокатки.
00648  *
00649  * Функция порождает дополнительный поток, который выполняет процесс
прокатки.
00650  * Все \ref callback "callback-функции" вызываются из этого потока.
Поэтому любое обращение
00651  * к графическому интерфейсу пользователя из них нежелательно.
00652  * Лучше использовать систему сообщений между callback-функциями и
00653  * главным потоком верхнего приложения.
00654  * \param cl - Указатель на структуру LSClient.
00655  * \return \ref returns "код ошибки".
00656  */
00657 LSCLIENT_API LSC_ERROR LSCStart(LSClient *cl);
00658
00659 /** \brief Завершение потока прокатки и освобождение всех ресурсов.
00660  *
00661  * Функция передает в поток прокатки команду на завершение и ожидает
окончания этого потока,
00662  * затем освобождает все занятые ресурсы и закрывает соединение со
сканером.
00663  * \param cl - Указатель на структуру LSClient.
00664  */
00665 LSCLIENT_API void LSCShutDown(LSClient* cl);
00666
00667 /** \brief Получение текущего состояния потока прокатки.
00668  *
00669  * Функция возвращает \ref state "статус" потока прокатки.
00670  * При изменении статуса так же вызывается callback-функция
LSClient::StateChanged().
00671  * \param cl - Указатель на структуру LSClient.
00672  * \return \ref state "Состояние потока прокатки".
00673  */
00674 LSCLIENT_API LSC_STATE LSCGetState(LSClient *cl);
00675
00676 /** \brief Получение кода ошибки потока прокатки.
00677  *
00678  * Функция возвращает код ошибки потока прокатки.
00679  * В случае какой-либо ошибки внутри потока статус потока меняется на
00680  * LSC_STATE::LSC_STAT_ERROR.
```

```
00681  * \param cl - Указатель на структуру LSClient.
00682  * \return \ref returns "код ошибки".
00683  */
00684 LSCLIENT_API LSC_ERROR LSCGetError(LSClient *cl);
00685
00686 /** \brief Получить версию библиотеки.
00687  *
00688  * Функция позволяет получить версию библиотеки в виде указателя на
строку
00689  *
00690  * \param cl - Указатель на структуру LSClient.
00691  * \return Строку с версией библиотеки.
00692  */
00693 LSCLIENT_API char* LSCGetVersion(LSClient* cl);
00694
00695 /** \brief Не используется.
00696  *
00697  *
00698  * \param cl - Указатель на структуру LSClient.
00699  * \param left - левая граница изображения.
00700  * \param right - правая граница изображения.
00701  */
00702 LSCLIENT_API void LSCSetFingerArea(LSClient* cl, int left, int right);
00703
00704 /** \brief Сегментация контрольных оттисков.
00705  *
00706  * Находит отпечаток с заданным номером в изображении контрольных
оттисков
00707  * и возвращает его координаты.
00708  * \param cl - Указатель на структуру LSClient
00709  * \param image - изображение контрольных оттисков
00710  * \param width - ширина изображения
00711  * \param height - высота изображения
00712  * \param num - номер отпечатка для выделения.
00713  * \param left - левая граница отпечатка в изображении
00714  * \param top - верхняя граница отпечатка в изображении
00715  * \param fwidth - ширина отпечатка
00716  * \param fheight - высота отпечатка
00717  * \return \ref returns "коды ошибок"
00718  */
00719 LSCLIENT_API LSC_ERROR LSCslapSegm(LSClient* cl, unsigned char* image,
int width, int height, int num, int* left,
00720                 int* top, int* fwidth, int* fheight);
00721
00722 /** \brief Сегментация контрольных оттисков.
00723  *
00724  * Разделяет контрольные оттиски на отдельные отпечатки и возвращает
00725  * координаты и размеры найденных отпечатков.
00726  * \param cl - указатель на структуру LSClient
00727  * \param image - изображение оттисков
00728  * \param width - ширина изображения
00729  * \param height - высота изображения
```

```
00730  * \param num - количество выделенных отпечатков
00731  * \param left - массив (int[4]) для сохранения левых границ
отпечатков
00732  * \param top - массив (int[4]) для сохранения верхних границ
отпечатков
00733  * \param fwidth - массив (int[4]) для сохранения ширины отпечатков
00734  * \param fheight - массив (int[4]) для сохранения высоты отпечатков
00735  * \return \ref returns "коды ошибок"
00736  */
00737 LSCLIENT_API LSC_ERROR LSCslapSegmAll(LSClient* cl, unsigned char*
image, int width, int height, int* num, int* left,
00738                 int* top, int* fwidth, int* fheight);
00739
00740 /** \brief Формирует список подключенных сканеров.
00741  *
00742  * \param list - Указатель на указатель для списка сканеров.
00743  * \return длинну выделенного списка подключенных сканеров
00744  */
00745 LSCLIENT_API int LSC_GetScannersList(LSC_SCANNERS_LIST** list);
00746
00747 /** \brief Освободить память выделенную в функции
LSC_GetScannersList() под список сканеров.
00748  */
00749 LSCLIENT_API void LSC_FreeScannersList(LSC_SCANNERS_LIST* list);
00750
00751 /** \brief Установить режим индикации светодиодов сканера.
00752  *
00753  *
00754  * \param cl - указатель на структуру верхнего приложения
00755  * \param mode - режим управления индикацией
00756  */
00757 LSCLIENT_API void LSC_SetLedMode(LSClient* cl, LSC_LED_MODE mode);
00758
00759 /** \brief Выключение/выключение светодиодов индикации сканера в
ручном режиме управления.
00760  *
00761  * \param cl - указатель на структуру верхнего приложения
00762  * \param led - номер светодиода, красный или зеленый.
00763  * \param on - состояние диода, TRUE - включить, FALSE - выключить.
00764  * \return \ref returns "коды ошибок"
00765  */
00766 LSCLIENT_API LSC_ERROR LSC_SetLed(LSClient* cl, LSC_LED led, BOOL on);
00767
00768 /** \brief Переключение режима передачи данных сканером по USB-шине.
00769  *
00770  * Функция может применяться для уменьшения потока данных от сканера к
компьютеру
00771  * по USB-шине в случае медленной USB-шины.
00772  * \param cl - указатель на структуру верхнего приложения
00773  * \param slow - TRUE - медленная USB-шина, FALSE - полноценная USB2
HIGH-SPEED 480mb/s шина.
00774  * \return \ref returns "коды ошибок"
00775  */
```

```
00776 LSCLIENT_API LSC_ERROR LSC_SetUsbType(LSClient* cl, BOOL slow);
00777
00778 /** \brief Установить режим захвата оттисков.
00779  *
00780  * \param cl - указатель на структуру верхнего приложения
00781  * \param mode - режим захвата оттисков
00782  *
00783  */
00784 LSCLIENT_API void LSCSetCaptureMode(LSClient* cl, LSC_CAPTURE_MODE
mode);
00785
00786 /** \brief Установить чувствительность алгоритма определения наличия
отпечатков на призме.
00787  *
00788  * Чем больше число, те менее чувствителен алгоритм к отпечаткам и
грязи соответственно.
00789  * Рекомендуемое значение(оно же и по умолчанию) - 500.
00790  * \param cl - указатель на структуру верхнего приложения
00791  * \param threshold - чувствительность, от LSC_MIN_THRESHOLD до
LSC_MAX_THRESHOLD
00792  */
00793 LSCLIENT_API LSC_ERROR LSCSetThreshold(LSClient* cl, int threshold);
00794
00795 /** Переключение режимов оттисков/прокатки для отпечатков.
00796  *
00797  * \param cl                    - указатель на структуру верхнего
приложения
00798  * \param touch_mode    - режим
00799  * - TRUE       - берутся оттиски пальцев
00800  * - FALSE      - пальца прокатываются
00801  */
00802 LSCLIENT_API void LSCSetTouchMode(LSClient* cl, BOOL touch_mode);
00803
00804 /** Отправить JPEG-изображение на монитор сканера.
00805  *
00806  *Данная функция работает только со сканером CrossMatch VMW.
00807  *
00808  * \param cl         - указатель на структуру верхнего приложения
00809  * \param buffer     - указатель на jpeg-изображения
00810  * \param length     - длина буфера изображения
00811  * \return \ref returns "коды ошибок"
00812  */
00813 LSCLIENT_API LSC_ERROR LSC_SendJpeg2Device(LSClient* cl, unsigned
char* buffer, int length);
00814
00815 /** Получить размеры экрана сканера.
00816  *
00817  *Работает только с интегрированным LCD-экраном сканера DS30NM,
00818  *с отстальными сканерами для вывода на экран необходимо использовать
00819  *библиотеку liblcd.
00820  * \param cl - указатель на структуру верхнего приложения
00821  * \param width - возвращается ширина экрана
00822  * \param height - возвращается высота экрана
```

```
00823  * \return                    - \ref returns
00824  */
00825 LSCLIENT_API LSC_ERROR LSC_GetSizeLCD(LSClient* cl, int *width,
int*height);
00826
00827 /** Сброс USB-буферов для передечи на LCD экран сканера.
00828  *
00829  *Работает только с интегрированным LCD-экраном сканера DS30NM,
00830  *с остальными сканерами для вывода на экран необходимо использовать
00831  *библиотеку liblcd.
00832  * \param cl - указатель на структуру верхнего приложения
00833  * \return                    - \ref returns
00834  */
00835 LSCLIENT_API LSC_ERROR LSC_ResetDataLCD(LSClient* cl);
00836
00837 /** Передать изображение на LCD экран сканера.
00838  *
00839  *Работает только с интегрированным LCD-экраном сканера DS30NM,
00840  *с остальными сканерами для вывода на экран необходимо использовать
00841  *библиотеку liblcd.
00842  * \param cl - указатель на структуру верхнего приложения
00843  * \param size - размер буфера с изображением
00844  * \param image - буфер с изображением
00845  * \return                    - \ref returns
00846  */
00847 LSCLIENT_API LSC_ERROR LSC_WriteImageLCD(LSClient* cl, long size, void
*image);
00848
00849 /** Передать палитру на LCD экран сканера.
00850  *
00851  *Работает только с интегрированным LCD-экраном сканера DS30NM,
00852  *с остальными сканерами для вывода на экран необходимо использовать
00853  *библиотеку liblcd.
00854  * \param cl - указатель на структуру верхнего приложения
00855  * \param AMaskSize - размер палитры
00856  * \param AMaskBuff - буфер с палитрой, по 3 байта на цвет, 256
цветов.
00857  * \return                    - \ref returns
00858  */
00859 LSCLIENT_API LSC_ERROR LSC_WriteMaskLCD(LSClient* cl, long AMaskSize,
void *AMaskBuff);
00860
00861 /** Установить яркость подсветки LCD экрана сканера.
00862  *
00863  *Работает только с интегрированным LCD-экраном сканера DS30NM,
00864  *с остальными сканерами для вывода на экран необходимо использовать
00865  *библиотеку liblcd.
00866  * \param cl - указатель на структуру верхнего приложения
00867  * \param light - яркость подсветки, от 0 до 31.
00868  * \return                    - \ref returns
00869  */
00870 LSCLIENT_API LSC_ERROR LSC_SetLightLCD(LSClient* cl, int light);
```

```
00871
00872 /** Включение/выключение питания экрана сканера.
00873  *
00874  *Работает только с интегрированным LCD-экраном сканера DS30NM,
00875  *с остальными сканерами для вывода на экран необходимо использовать
00876  *библиотеку liblcd.
00877  * \param cl - указатель на структуру верхнего приложения
00878  * \param on    - 1 - включить питание, 0 - выключить.
00879  * \return                  - \ref returns
00880  */
00881 LSCLIENT_API LSC_ERROR LSC_SetLCDPower(LSClient* cl, int on);
00882
00883 /** Возвращает флаг наличия в сканере интегрированного LCD экрана.
00884  *
00885  * \param cl - указатель на структуру верхнего приложения
00886  * \return TRUE  - экран интегрирован
00887  * \return FALSE  - экран не интегрирован
00888  */
00889 LSCLIENT_API BOOL LSC_IntegratedLCD(LSClient* cl);
00890
00891 /** Возвращается версию EEPROM firmware сканера.
00892  *
00893  * \param cl - указатель на структуру верхнего приложения
00894  * \param firmware - возвращается версия firmware
00895  * \return                  - \ref returns
00896  */
00897 LSCLIENT_API LSC_ERROR LSC_GetFirmware(LSClient* cl, char* firmware);
00898
00899 /** Расчет качества отпечатка по алгоритму Папилон.
00900  *
00901  * \param[in] cl - object of library.
00902  *
00903  * \param[in] image - буфер с изображением. Изображение должон иметь
глубину 8bpp.
00904  *
00905  * \param[in] width - количество колонок
00906  *
00907  * \param[in] height - количество строк
00908  *
00909  * \param[in] res - разрешение изображения в аикселах на дюйм, обычно
500.
00910  *
00911  * \param[in] cl - указатель на структуру верхнего приложения.
00912  *
00913  * \param[out] aquality - указатель для сохранения качества
папилярного узора.
00914  * Изменяется от  0(плохое) до 100(отличное).
00915  *
00916  * \param[out] vquality - указатель для сохранения качества
изображения
00917  * зменяется от 0(худшее) до 100(отличное).
00918  * Харакетризует динамический диапазон изображения.
```

```
00919  *
00920  * \param[out] pressure - указатель для сохранения силы прижима пальца
к призме.
00921  * Изменияется 0.0(слабое давление) до 2.0(сильное давление).
00922  * Нормальное значение 1.0.
00923  *
00924  * \return                                        - \ref returns
00925  */
00926 LSCLIENT_API LSC_ERROR LSC_GetAfisQuality(LSClient* cl, unsigned char*
image, int width, int height, int res,
00927                 int* aquality, int* vquality, float* pressure);
00928
00929 /** Расчет качества отпечатка по алгоритму NIST.
00930  *
00931  * \param[in] cl - object of library.
00932  *
00933  * \param[in] image - буфер с изображением. Изображение должон иметь
глубину 8bpp.
00934  *
00935  * \param[in] width - количество колонок
00936  *
00937  * \param[in] height - количество строк
00938  *
00939  * \param[in] res - разрешение изображения в аикселах на дюйм, обычно
500.
00940  *
00941  * \param[in] cl - указатель на структуру верхнего приложения.
00942  *
00943  * \param[out] aquality - указатель для сохранения качества
папилярного узора.
00944  * Изменяется от  5(плохое) до 1(отличное).
00945  *
00946  * \param[out] vquality - указатель для сохранения качества
изображения
00947  * зменяется от 0(худшее) до 100(отличное).
00948  * Харакетризует динамический диапазон изображения.
00949  *
00950  * \param[out] pressure - указатель для сохранения силы прижима пальца
к призме.
00951  * Изменияется 0.0(слабое давление) до 2.0(сильное давление).
00952  * Нормальное значение 1.0.
00953  *
00954  * \return                                        - \ref returns
00955  */
00956 LSCLIENT_API LSC_ERROR LSC_GetNistQuality(LSClient* cl, unsigned char*
image, int width, int height, int res,
00957                 int* aquality, int* vquality, float* pressure);
00958
00959 /** Проверить поддерживает ли сканер разные зоны прокатки.
00960  *
00961  * \param cl    - указатель на структуру верхнего приложения.
00962  * \return true если сканер поддерживает выбор различных зон прокатки.
00963  */
```

```
00964 LSCLIENT_API BOOL LSC_SupportRollPlace(LSClient* cl);
00965
00966 /** Выбор зоны прокатки
00967  *
00968  * Функцию можно использовать только для сканеров, которые
поддерживают
00969  * выбор зоны прокатки, в данный момент только для сканера ДС45(М).
00970  *
00971  * \param cl    - указатель на структуру верхнего приложения.
00972  * \param place - зона прокатки
00973  * \return - \ref returns
00974  */
00975 LSCLIENT_API LSC_ERROR LSC_SetRollPlace(LSClient* cl, LSC_ROLL_PLACE
place);
00976
00977 /**Включение/выключение режима определения муляжа отпечатка.
00978  *
00979  * При включении этой функции время захвта отпечатка увеличивается
примерно в 2 раза.
00980  * Данный режим работает только на некоторых сканерах.
00981  *
00982  * @param cl - указатель на структуру верхнего приложения.
00983  * @param _enable - true - включить определение муляжа, false -
выключить.
00984  * @return false если данная функция не поддерживается сканером.
00985  */
00986 LSCLIENT_API BOOL LSC_EnableFakeDetection(LSClient* cl, bool _enable);
00987
00988 /** Функция проверяет возможность извлечения мелких особенностей из
изображения отпечатка
00989  *
00990  * Эта функция нализирует качество изображения на предмет пригодности
для кодирования и возвращает результат
00991  * в параметре _good.
00992  * Если изображение достаточного качества для последующего кодирования
и сравнения, то выходной параметр _good
00993  * будет содержкать TRUE.
00994  * Функция достаточно ресурсоемкая и занимает времени порядка 1
секунды на процессоре Intel Core i7 3ГГц для одного отпечатка.
00995  * Использование данной функции защищено коммерческой лицензией, за
получением лицензии обратитесь к разработчику.
00996  *
00997  * @param cl  - указатель на структуру верхнего приложения.
00998  * @param _image - указатель на изображение отпечатка, 8bpp, 500ppi
00999  * @param _width - ширина изображения в пикселах, количество колонок
01000  * @param _height - высота изображения в пикселах, количество строк
01001  * @param[out] _good - на выходе содержит TRUE если изображение
пригодно для обработки
01002  * @return - \ref returns
01003  */
01004 LSCLIENT_API LSC_ERROR LSC_CheckEncode( LSClient *cl, unsigned char*
_image, int _width, int _height, BOOL* _good );
01005
```

```
01006 /** Вычисление качества отпечатка применительно к алгоритму посика
компании Папилон
01007  *
01008  * Эта функция рассчитывает качество отпечатка 0(плохое) ...
3(отличное).
01009  * Функция может быть использована для оценки необходимого количества
отпечатков для выполнения автоматического поика по базе.
01010  * Наилучшее качество поиска будет достигнуто при минимум двух
отпечатках с качеством не ниже 2.
01011  * Возможно подавать на поиски один отпечаток с качеством 3.
01012  * В остальных случаях необходимо сканировать дополнительные
отпечатки.
01013  * Функция защищена коммерческой лицензией.
01014  *
01015  * @param cl  - указатель на структуру верхнего приложения.
01016  * @param _image - указатель на изображение отпечатка, 8bpp, 500ppi
01017  * @param _width - ширина изображения в пикселах, количество колонок
01018  * @param _height - высота изображения в пикселах, количество строк
01019  * @param[out] _quality - выходное значение качества отпечатка.
01020  * @return - \ref returns
01021  */
01022 LSCLIENT_API LSC_ERROR LSC_CheckQuality( LSClient *cl, unsigned char*
_image, int _width, int _height, int* _quality );
01023
01024 /** Установить файл лицензии LSSDK
01025  *
01026  * Лицензия может быть привязана к серийному номеру сканера или к
идентификатору оборудования на котором работает SDK.
01027  * Лицензия проверяется в функциях
01028  * - LSC_CheckQuality()
01029  * - LSC_CheckEncode()
01030  * - LSC_CompressWsqPpln()
01031  * - LSC_DecompressWsqPpln()
01032  *
01033  * @param cl - указатель на структуру верхнего приложения.
01034  * @param _path -путь до файла лицензии, обычно называется lssdk.lic
01035  * @return - \ref returns
01036  */
01037 LSCLIENT_API LSC_ERROR LSC_SetLicense(LSClient *cl, const char*
_path);
01038
01039 /*@}*/
01040 /*@}*/
01041
01042 /**@{*/
01043 /** \defgroup wsq Функции сжатия/разжатия WSQ.
01044  *
01045  * Эти функции реализуют сжатие и разжатие изображений методом WSQ.
01046  */
01047 /*@{*/
01048
01049 /** Сжатие изображения с сипользованием WSQ
01050  *
```

```
01051  * \param[in] cl - объект библиотеки.
01052  * \param[in] image  - указатель на буфер с изображением
01053  * \param[in] width  - количество колонок изображения
01054  * \param[in] height  - количество строк изображения
01055  * \param[in] bps  - количество бит в одной компоненте, на данный
момент поддерживатеся только 8.
01056  * \param[in] spp  - количество цветовых компонент в пикселе,
01057  * поддерживается только 1 для серых и 3 для RGB.
01058  * \param[in] compression - коэффициент сжатия, меняется от
MIN_COPMPRESSION до MAX_COPMPRESSION.
01059  * \param[out] wsq_rec - указатель для сохранения указателя на
выделенную память со сжатым изображеним,
01060  *           память выделяется внутри функции и должна быть
освобождена функцией LSC_FreeWsqPpln() после использования.
01061  * \param[out] wsq_len  - указатель для сохранения длинны
получившегося сжатого изображения.
01062  * \return - \ref returns
01063  */
01064 LSCLIENT_API LSC_ERROR LSC_CompressWsqPpln(LSClient* cl, unsigned
char* image, int width, int height, int bps, int spp,
01065                 double compression, unsigned char**wsq_rec, int*
wsq_len);
01066
01067 /** Разжатие изображения WSQ
01068  *
01069  * \param[in] cl - объект библиотеки.
01070  * \param[in] wsq_rec - указатель на сжатое изображение.
01071  * \param[in] wsq_len  - длинна сжатого изображения в байтах.
01072  * \param[out] image  - указатель для сохранения указателя на
выделенный буфер с разжатым изображением,
01073  *           память выделяется внутри функции и должна быть
освобождена функцией LSC_FreeWsqPpln() после использования.
01074  * \param[out] width  - указатель для сохранения количества колонок в
изображении.
01075  * \param[out] height  - указатель для сохранения количества строк в
изображении.
01076  * \param[in] bps  - количество бит в одной цветовой компоненте
сжатого изображения, поддерживается только 8.
01077  * \param[in] spp  - количество цветовых компонент в пикселе, 1 для
серых и 3 для RGB изображений.
01078  * \return - \ref returns
01079  */
01080 LSCLIENT_API LSC_ERROR LSC_DecompressWsqPpln(LSClient* cl, unsigned
char*wsq_rec, int wsq_len, unsigned char** image,
01081                 int* width, int* height, int bps, int spp);
01082
01083 /** Освобождение памяти захваченной в функциях LSC_CompressWsqPpln() и
LSC_DecompressWsqPpln().
01084  *
01085  * \param[in] cl - объект библиотеки.
01086  * \param[in] rec - указатель на память, которая должна быть
освобождена.
01087  **/
01088 LSCLIENT_API LSC_ERROR LSC_FreeWsqPpln(LSClient* cl, unsigned char*
rec);
```

```
01089
01090 /*@}*/
01091 /*@}*/
01092
01093 /** \example main.cpp Пример использования библиотеки, функция main()
*/
01094 /** \example lschandle.h Пример использования библиотеки, описание
объекта приложения LSClient::Client
01095  *  и \ref callback "Callback-функций". */
01096 /** \example lschandle.cpp Пример использования библиотеки, \ref
callback "Callback-функции". */
01097 /** \example mainform.h Пример использования библиотеки, главная
форма. */
01098 /** \example mainform.cpp Пример использования библиотеки, главная
форма. */
01099 /** \example imgview.h Класс для вывода изображения 8bpp(256 градаций
серого). */
01100 /** \example imgview.cpp Класс для вывода изображения 8bpp(256
градаций серого), реализация. */
01101 /** \example messageform.cpp Класс для вывода различных сообщений. */
01102 /** \example messageform.h Класс для вывода различных сообщений. */
01103 /** \example previewform.cpp Класс для показа предварительного
изображения во время сканирования. */
01104 /** \example previewform.h Класс для показа предварительного
изображения во время сканирования. */
01105
01106 #ifdef __cplusplus
01107 }
01108 #endif
01109
01110 #endif
01111
```

Примеры

imgview.cpp

Класс для вывода изображения 8bpp (256 градаций серого), реализация.

```
/*!
 * \file imgview.cpp
 * \brief Функции класса  CImgView
 * \author vav
 * \date   created 12-Feb-2007
 */

#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include <qpainter.h>
#include <qapplication.h>
//Added by qt3to4:
```

```
#include <QPixmap>
#include <QMouseEvent>

#include "imgview.h"

#ifndef abs
/**Return absolute value of A*/
#define abs(a)   ((a>0) ? (a):(-(a)))
#endif

#ifndef M_PI
/**The PI*/
#define M_PI    3.14159265358979323846
#endif /* M_PI */

#ifndef min
/**Return minimum value*/
#define min(a,b)        (((a)<(b)) ? (a):(b))
#endif /* min */

#ifndef max
/**Return maximum value*/
#define max(a,b)        (((a)>(b)) ? (a):(b))
#endif /* min */

/*! Конструктор класса
 *
 * \param[in] parent    - родитель
 * \param[in] name      - имя объекта
 * \param[in] nCols     - количество пикселов в строке изображения
 * \param[in] nRows     - количество строк в изображении
 * \param[in] bViewAll  - установить размеры окна объекта равными
 * размерам изображения
 */
CImgView::CImgView( QWidget* parent, const char *name, int nCols, int nRows,
        bool bViewAll ) :
        Q3ScrollView( parent, name )
{
    color_mode = -1;
    m_bViewAll = bViewAll;

    num = -1;
    setFrameStyle( WinPanel | Sunken );
    viewport()->setBackgroundMode( Qt::NoBackground );

    nHLines = 10;
    nVLines = 10;
    draw_calibr = FALSE;
    zoom = FALSE;
    zoom_scale = 1.0;
    img = NULL;
```

```
    setImageSize( nCols, nRows );
    npoint = 0;
    draw_rect = draw_degree = draw_points = draw_mouse = FALSE;
    degree = 0.0;
    mouse_x = 0;
    rect_x1 = rect_y1 = rect_x2 = rect_y2 = 0;

    points[0][0] = QPoint( 557, 183 );
    points[0][1] = QPoint( 215, 193 );
    points[1][0] = QPoint( 557, 183 );
    points[1][1] = QPoint( 133, 197 );
}

/*!  Деструктор класса
 *
 */
CImgView::~CImgView()
{
    if( img )
    {
        delete img;
    }
}

/*! Установить размеры изображения.
 *
 * \param[in] nCols     - количество пикселов в строке
 * \param[in] nRows     - количество строк в изображении
 */
void CImgView::setImageSize( int nCols, int nRows )
{
    m_nCols = nCols;
    m_nRows = nRows;

    xy[0][0] = 10;
    xy[0][1] = 10;

    xy[1][0] = nCols / 2;
    xy[1][1] = 10;

    xy[2][0] = nCols - 10;
    xy[2][1] = 10;

    xy[3][0] = nCols - 10;
    xy[3][1] = nRows / 2;

    xy[4][0] = nCols - 10;
    xy[4][1] = nRows - 10;

    xy[5][0] = nCols / 2;
    xy[5][1] = nRows - 10;
```

```
    xy[6][0] = 10;
    xy[6][1] = nRows - 10;

    xy[7][0] = 10;
    xy[7][1] = nRows / 2;

    if( nRows == 0 || nCols == 0 )
        hide();
    else
        show();

    updateGeometry();

    if( zoom )
    {
        int w, h;

        w = visibleWidth();
        h = visibleHeight();

        m_pixmap = QPixmap( w, h );
    }
    else
    {
        m_pixmap = QPixmap( m_nCols, m_nRows );
    }

    m_pixmap.fill( QColor( 0, 0, 0 ) );

    if( img )
    {
        delete img;
        img = NULL;
    }

    makeImage( m_nCols, m_nRows );

    if( !zoom )
        resizeContents( m_nCols, m_nRows );

    setMaxSize();
}

/*! Задать режим окна объекта.
 *
 * Если TRUE, то размеры окна делаются равными размерам изображения.
 * Если FALSE, то  окно может быть меньше размеров изображения и появятся
 * слайдеры для перемещения изображения.
 *
 * \param[in] bViewAll  - режим окна объекта
```

```
 */
void CImgView::setViewAll( bool bViewAll )
{
    m_bViewAll = bViewAll;
    setMaxSize();
}

/*! Установить режим масштабирования изображения под размер окна объекта.
 *
 * \param[in] aZoom     - если TRUE, то изображение будет
 * автоматически вписано в окно объекта.
 */
void CImgView::setZoom( bool aZoom )
{
    zoom = aZoom;
    m_bViewAll = FALSE;

    if( !img )
        return;

    viewport()->setBackgroundMode( Qt::PaletteBackground );

    if( zoom )
    {
        int w, h;

        w = visibleWidth();
        h = visibleHeight();
        resizeContents( w, h );
    }
    else
    {
        resizeContents( m_nCols, m_nRows );
    }

    makePixmap();

    resizeContents( m_pixmap.width(), m_pixmap.height() );

    updateFrame();

    setMaxSize();
    viewport()->setBackgroundMode( Qt::NoBackground );
}

/*! Ограничить минимальные и максимальные размеры окна объекта в
 *    зависимости от режима.
 *
 */
void CImgView::setMaxSize()
{
```

```
    QSize s;

    s = QSize( m_nCols, m_nRows ) + QSize( 2 * frameWidth(), 2 *
frameWidth() );

    setMaximumSize( s );

    if( m_bViewAll )
        setMinimumSize( s );
    else
        setMinimumSize( QSize( 20 * frameWidth(), 20 * frameWidth() ) );

    resize( s );
    updateGeometry();
}

/*! Return size of object
 *
 * \return size of object
 */
QSize CImgView::sizeHint()
{
    QSize s;

    s = QSize( m_nCols, m_nRows ) + QSize( 2 * frameWidth(), 2 *
frameWidth() );

    return s;
}

/*! Обработчик события перерисовки содержимого объекта.
 *
 * \param[in] paint     - указатель на объект QPainter
 * \param[in] clipx     - смещение обновляемой области по X
 * \param[in] clipy     - смещение обновляемой области по Y
 * \param[in] clipw     - ширина обновляемой области
 * \param[in] cliph     - высота обновляемой области
 */
void CImgView::drawContents( QPainter * paint, int clipx, int clipy, int
clipw,
        int cliph )
{
    bitBlt( paint->device(), clipx - contentsX(), clipy - contentsY(),
            &m_pixmap, clipx, clipy, clipw, cliph, 0 );
}

/*! Создание объекта QPixmap из объекта с изображением QImage.
 *
 */
void CImgView::makePixmap()
{
    int h, w;
```

```
    if( zoom )
    {
        QImage zi;

        w = visibleWidth();
        h = visibleHeight();

        if( (w == 0) || (h == 0) )
            return;

        zi = img->smoothScale( w, h, Qt::KeepAspectRatio );
        zoom_scale = ((double) m_nCols) / (double) zi.width();
        m_pixmap = QPixmap::fromImage( zi );
    }
    else
    {
        zoom_scale = 1.0;
        m_pixmap = QPixmap::fromImage( *img );
    }

    drawMouseLine();
    drawDegreeLine();
    updateRect();
}

/*! Отрисовка вертикальной линии, следующей за указателем мыши.
 *
 */
void CImgView::drawMouseLine( void )
{
    if( draw_mouse )
    {
        QPainter qp( &m_pixmap );
        qp.setPen( QColor( 0, 150, 0 ) );
        qp.drawLine( mouse_img_x, 0, mouse_img_x, m_pixmap.height() - 1 );
        qp.end();
        updateFrame();
    }
}

/*! Отрисовка линии для оценки угла наклона.
 *
 */
void CImgView::drawDegreeLine( void )
{
    if( draw_degree )
    {
        QPainter qp( &m_pixmap );
        qp.setPen( QColor( 150, 0, 0 ) );
        qp.drawLine( degree_x1, degree_y1, degree_x2, degree_y2 );
```

```
        qp.end();
        updateFrame();
    }
}

/*! Функция обновления изображения на экране.
 *
 */
void CImgView::updateFrame()
{
    viewport()->repaint( FALSE );
}

/*! Передать новое изображение для отрисовки.
 *
 *  Исходное изображение подается в raw-формате, градации серого, 8 бит на
 *  пиксел.
 *
 * \param[in] pData     - буфер с изображением
 * \param[in] cx        - ширина изображения
 * \param[in] cy        - высота изображения
 */
void CImgView::makePixmap( unsigned char * pData, int cx, int cy )
{
    unsigned char* pDst;

    if( (cx != m_nCols) || (cy != m_nRows) || (!img) )
    {
        if( img )
        {
            delete img;
            img = NULL;
        }

        makeImage( cx, cy );

    }

    for( int y = 0; y < cy; y++ )
    {
        pDst = img->scanLine( y );
        memcpy( pDst, pData, cx );
        pData += cx;
    }

    makePixmap();

    if( zoom )
    {
        resizeContents( m_pixmap.width(), m_pixmap.height() );
        setFixedSize(
```

```
                QSize( m_pixmap.width(), m_pixmap.height() )
                        + QSize( 2 * frameWidth(), 2 * frameWidth() ) );
    }
}

/*! Отрисовка части изображения
 *
 *  Исходное изображение подается в raw-формате, градации серого, 8 бит на
 *  пиксел.
 *
 * \param[in] pData     - буфер с изображением
 * \param[in] left      - смещение области для отрисовки от левого
 * края изображения
 * \param[in] top       - смещение области для отрисовки от верхнего
 * края изображения
 * \param[in] cx        - ширина области отрисовки
 * \param[in] cy        - высота области отрисовки
 * \param[in] buf_cx    - ширина всего изображения
 */
void CImgView::makePixmap( unsigned char * pData, int left, int top, int cx,
        int cy, int buf_cx )
{
    unsigned char* pDst;

    if( (left > m_nCols) || ((left + cx) > m_nCols) || (cx > m_nCols)
            || (top > m_nRows) || ((top + cy) > m_nRows) || (cy > m_nRows) )
        return;

    if( (!img) )
    {
        makeImage( cx, cy );
    }

    for( int y = top; y < (top + cy); y++ )
    {
        pDst = img->scanLine( y ) + left;
        memcpy( pDst, pData, cx );
        pData += buf_cx;
    }

    makePixmap();
}

/*! Установить координаты прямоугольника выбранной
 *    области изображения.
 *
 * Выбрать область можно используя мышь при включенной опции draw_rect.
 *
 * \param[in] left      - смещение прямоугольника от левого края изображения
 * \param[in] top       - смещение прямоугольника от верхнего края
изображения
 * \param[in] width     - ширина прямоугольника
```

```
 * \param[in] height    - высота прямоугольника
 */
void CImgView::setSelectedRect( int left, int top, int width, int height )
{
    rect_x1 = (int) (((double) left) / zoom_scale + 0.5);
    rect_y1 = (int) (((double) top) / zoom_scale + 0.5);
    rect_x2 = (int) (((double) (left + width - 1)) / zoom_scale + 0.5);
    rect_y2 = (int) (((double) (top + height - 1)) / zoom_scale + 0.5);
}

/*! Получить координаты выбранной области.
 *
 * Выбрать область можно используя мышь при включенной опции draw_rect.
 *
 * \param[out] left     - возвращается смещение выбранной области от
 * левого края изображения
 * \param[out] top      - возвращается смещение выбранной области
 * от верхнего края изображения
 * \param[out] width    - возвращается ширина выбранной области
 * \param[out] height   - возвращается высота выбранной области
 */
void CImgView::getSelectedRect( int* left, int* top, int* width, int* height
)
{
    int l, t, r, b;

    l = (int) (((double) min(rect_x2,rect_x1)) * zoom_scale + 0.5);
    t = (int) (((double) min(rect_y2,rect_y1)) * zoom_scale + 0.5);
    r = (int) (((double) max(rect_x2,rect_x1)) * zoom_scale + 0.5);
    b = (int) (((double) max(rect_y2,rect_y1)) * zoom_scale + 0.5);

    l = (l / 8) * 8;
    t = (t / 8) * 8;
    r = (((r - l) + 7) / 8) * 8 + l;
    b = (((b - t) + 7) / 8) * 8 + t;

    if( l > m_nCols )
        l = 0;

    if( r >= m_nCols )
        r = m_nCols - 1;

    if( b >= m_nRows )
        b = m_nRows - 1;

    if( left )
        *left = l;

    if( top )
        *top = t;

    if( width )
```

```
        *width = r - l + 1;

    if( height )
        *height = b - t + 1;
}

/*! Обновить  изображение без изменения его размеров.
 *
 * \param[in] pData     - изображение
 * \param[in] buf_cx    - ширина исходного изображения
 */
void CImgView::makePixmap( unsigned char * pData, int buf_cx )
{
    unsigned char* pDst;
    int y;

    if( (!img) )
    {
        makeImage( m_nCols, m_nRows );
    }

    for( y = 0; y < m_nRows; y++ )
    {
        pDst = img->scanLine( y );
        memcpy( pDst, pData, m_nCols );
        pData += buf_cx;
    }

    makePixmap();
}

/*! Обработчик нажатия мышки
 *
 * \param[in] e - указатель на объект события
 */
void CImgView::contentsMousePressEvent( QMouseEvent* e )
{
    m_bMoving = true;
    m_posMovingStart = e->pos();

    if( draw_points && (npoint == 1) )
    {
        points[npoint][1] = e->pos();
    }

    if( draw_calibr )
    {
        int x, y, i, d;
        int min = 100000;
        int p = 0;
```

```
        x = e->x();
        x -= contentsRect().x();

        y = e->y();
        y -= contentsRect().y();

        if( x < 0 )
            x = 0;

        if( y < 0 )
            y = 0;

        if( x >= this->m_nCols )
            x = m_nCols - 1;

        if( y >= this->m_nRows )
            y = m_nRows - 1;

        if( num < 0 )
        {
            for( i = 0; i < 8; i++ )
            {
                d = (int) ((xy[i][0] - x) * (xy[i][0] - x)
                        + (xy[i][1] - y) * (xy[i][1] - y));
                if( d < min )
                {
                    min = d;
                    p = i;
                }
            }
            num = p;
        }
    }

    if( draw_rect )
    {
        rect_x2 = rect_x1 = (e->pos().x() / 8) * 8;
        rect_y2 = rect_y1 = (e->pos().y() / 8) * 8;
    }

    if( draw_mouse )
    {
        mouse_img_x = e->pos().x();
        mouse_x = (int) (((double) mouse_img_x) * zoom_scale);
        emit mouseChanged( mouse_x );
    }

    if( draw_degree )
    {
        degree_x1 = e->pos().x();
        degree_y1 = e->pos().y();
```

```
    }
}

/*! Обработчик отпускания мышки
 *
 * \param[in] e - указатель на объект события
 */
void CImgView::contentsMouseReleaseEvent( QMouseEvent* e )
{
    (void)e;
    m_bMoving = false;
    num = -1;
}

/*! Обработчик перемещения мышки
 *
 * \param[in] e - указатель на объект события
 */
void CImgView::contentsMouseMoveEvent( QMouseEvent* e )
{
    if( draw_calibr )
    {
        int x, y;

        if( (e->state() & Qt::RightButton) )
        {
            x = e->x() - contentsRect().x();
            y = e->y() - contentsRect().y();

            if( x < 0 )
                x = 0;

            if( y < 0 )
                y = 0;

            if( x >= this->m_nCols )
                x = m_nCols - 1;

            if( y >= this->m_nRows )
                y = m_nRows - 1;

            if( num >= 0 )
            {
                xy[num][0] = x;
                xy[num][1] = y;
            }
        }

        makePixmap();
        DrawCPoints();
        updateFrame();
```

```
    }

    if( draw_rect )
    {
        if( e->state() == Qt::LeftButton )
        {
            int l, t, w, h;

            rect_x2 = e->pos().x();
            rect_y2 = e->pos().y();
            getSelectedRect( &l, &t, &w, &h );
            rect_x1 = (int) (l / zoom_scale + 0.5);
            rect_x2 = (int) ((l + w - 1) / zoom_scale + 0.5);
            rect_y1 = (int) (t / zoom_scale + 0.5);
            rect_y2 = (int) ((t + h - 1) / zoom_scale + 0.5);

            makePixmap();
            emit
            rectChanged( w, h );
            emit
            rectChangedWidth( w );
            emit rectChangedHeight( h );
        }
    }

    if( draw_points )
    {
        int pw, ph, i;

        if( npoint == 1 )
            points[0][0] = points[1][0] = e->pos();
        else
            points[0][1] = e->pos();

        makePixmap();

        QPainter qp( &m_pixmap );
        qp.setPen( QColor( 0, 150, 0 ) );

        for( i = 0; i < 2; i++ )
        {
            ph = (points[i][1].y() - points[i][0].y());
            pw = (points[i][1].x() - points[i][0].x());
            pw = (int) (sqrt( (double) (pw * pw + ph * ph) )*2);
            if( pw < ph )
                pw = ph;

            qp.drawEllipse( points[i][0].x() - pw / 2,
                            points[i][0].y() - pw / 2, pw, pw );
        }
```

```
        qp.end();
        updateFrame();
        return;
    }

    if( draw_degree )
    {
        degree_x2 = e->pos().x();
        degree_y2 = e->pos().y();
        if( abs(degree_x2-degree_x1) < 1 )
            degree = 90.0;
        else
            degree = (180.0
                    * atan( ((double) abs(degree_y2-degree_y1))
                            / ((double) abs(degree_x2-degree_x1)) )) / M_PI;
        makePixmap();
        emit degreeChanged( degree );
    }

    if( !(e->state() & Qt::RightButton) )
    {
        m_bMoving = true;
        if( m_bMoving )
        {
            setContentsPos( m_posMovingStart.x() - e->pos().x() +
contentsX(),
                            m_posMovingStart.y() - e->pos().y() +
contentsY() );
        }
    }
}

/*! Установить количество отображаемых окружностей.
 *
 * Всего можно рисовать 2 окружности, при включенном режиме
 * draw_points. Центр окружности задается указателем мышки по нажатию
 * левой кнопки мыши, дальше не отпуская левой кнопки задается радиус
окружности.
 *
 * \param[in] num       -  количество отображаемых окружностей - 1
 */
void CImgView::setPoint( int num )
{
    npoint = num;
}

/*! Переключение режима отрисовки вертикальной линии следующей за указателем
мыши.
 *
 * \param[in] en        - TRUE - включить отрисовку линии, FALSE -
выключить.
 */
```

```
void CImgView::drawMouse( bool en )
{
    draw_mouse = en;
    mouse_img_x = mouse_x = 0;
}

/*! Переключение режима отрисовки линии для оценки угла наклона
 объекта изображения.
 *
 * \param[in] en        - TRUE - включить отрисовку, FALSE - выключить.
 */
void CImgView::drawDegree( bool en )
{
    draw_degree = en;
    degree_x1 = degree_y1 = degree_x2 = degree_y2 = 0;
}

/*! Переключение режима отрисовки выбранной области на изображении.
 *
 * \param[in] en        - TRUE - включить отрисовку, FALSE - выключить.
 */
void CImgView::drawRect( bool en )
{
    draw_rect = en;
}

/*! Переключение режима отрисовки окружностей.
 *
 * \param[in] en        - TRUE - включить отрисовку, FALSE - выключить.
 */
void CImgView::drawPoints( bool en )
{
    draw_points = en;

    if( draw_points )
    {
        int pw, ph, i;

        makePixmap();

        QPainter qp( &m_pixmap );
        qp.setPen( QColor( 0, 150, 0 ) );

        for( i = 0; i < 2; i++ )
        {
            ph = (points[i][1].y() - points[i][0].y());
            pw = (points[i][1].x() - points[i][0].x());
            pw = (int) (sqrt( (double) (pw * pw + ph * ph) )*2);
            if( pw < ph )
                pw = ph;
```

```
            qp.drawEllipse( points[i][0].x() - pw / 2,
                            points[i][0].y() - pw / 2, pw, pw );
        }
        qp.end();
        updateFrame();
    }
}

/*! Перерисовать прямоугольник выбранной области изображения.
 *
 * Отображается в случае включенной опции draw_rect.
 */
void CImgView::updateRect()
{
    if( draw_rect )
    {
        QPainter qp( &m_pixmap );
        qp.setPen( QColor( 0, 64, 0 ) );
        qp.drawRect( min(rect_x1,rect_x2), min(rect_y1,rect_y2),
        abs(rect_x2-rect_x1), abs(rect_y1-rect_y2) );

        qp.setPen( QColor( 255, 255, 128 ) );
        qp.drawRect( min(rect_x1,rect_x2) + 1, min(rect_y1,rect_y2) + 1,
                     abs(rect_x2-rect_x1) - 2, abs(rect_y1-rect_y2) - 2 );

        qp.setPen( QColor( 0, 64, 0 ) );
        qp.drawRect( min(rect_x1,rect_x2) + 2, min(rect_y1,rect_y2) + 2,
                     abs(rect_x2-rect_x1) - 4, abs(rect_y1-rect_y2) - 4 );

        qp.end();
        updateFrame();
    }
}

/*! Создание и инициализация объекта img.
 *
 *
 * \param[in] cx        - ширина изображения
 * \param[in] cy        - высота изображения
 */
void CImgView::makeImage( int cx, int cy )
{
    if( !cx || !cy )
        return;

    if( !img )
    {
        img = new QImage( cx, cy, 8, 256 );
        img->setNumColors( 256 );
    }
```

```
    switch( color_mode )
    {
        case 0:
        {
            for( int n = 0; n < 256; n++ )
                img->setColor( n, QColor( n, 0, 0 ).rgb() );

            break;
        }
        case 1:
        {
            for( int n = 0; n < 256; n++ )
                img->setColor( n, QColor( 0, n, 0 ).rgb() );

            break;
        }
        case 2:
        {
            for( int n = 0; n < 256; n++ )
                img->setColor( n, QColor( 0, 0, n ).rgb() );

            break;
        }
        default:
        {
            for( int n = 0; n < 256; n++ )
                img->setColor( n, QColor( n, n, n ).rgb() );

            break;
        }
    }

}

/*! Установить цвет отрисовки изображения.
 *
 * В зависимости от режима изображение отображается либо
 * - красным цветом, режим 0
 * - зеленым цветом, режим 1
 * - синим цветом, режим 2
 * - градациями серого, режим -1
 *
 * \param[in] color     - цвет отрисовки изображения.
 */
void CImgView::setColorMode( int color )
{
    color_mode = color;

}

/*! Функция рисует точку на изображении в указанных координатах.
```

```
 *
 * \param[in] x - координата X точки
 * \param[in] y - координата Y точки
 */
void CImgView::drawImagePoint( int x, int y )
{
    QPainter qp( &m_pixmap );
    qp.setPen( QColor( 0, 255, 0 ) );
    qp.drawPoint(
            (int) (((double) x) / zoom_scale + 0.5),
(int)(((double)y)/zoom_scale+0.5) );
    qp.end();
}

/*! Переключение режима отрисовки калибровочной сетки.
 *
 * Калибровочная сетка генерируется по 8-ми точкам, устанавливаемым
 * по периметру изображения мышкой.
 *
 * \param[in] en        - TRUE - включение отрисовки, FALSE - выключение.
 */
void CImgView::drawCalibr( bool en )
{
    draw_calibr = en;
}

/*! Расчет коэффициентов функций описывающей калибровочную сетку.
 *
 * \param[in] xy        - координаты заданных 8-ми точек
 * \param[out] abc      - возвращаются коэффициенты функции изменения
 * плотности установленных точек вдоль четырех сторон изображения.
 * \param[out] ab       - возвращаются коэффициенты функции четырех линий
вдоль
 * которых установлены точки.
 * \param[in] width     - ширина изображения
 * \param[in] height    - высота изображения
 */
void CImgView::CalculateFunctions( double xy[8][2], double abc[4][3],
        double ab[4][2], int width, int height )
{
    double X[3], Y[3];

    X[0] = xy[0][0];
    Y[0] = 1;
    X[2] = xy[2][0];
    Y[2] = width - 1;
    ab[0][1] = (xy[0][1] - xy[2][1]) / (xy[0][0] - xy[2][0] + 0.0001);
    ab[0][0] = xy[0][1] - ab[0][1] * xy[0][0];
    xy[1][1] = ab[0][0] + ab[0][1] * xy[1][0];
    X[1] = xy[1][0];
    Y[1] = width / 2;
    least_squares2( Y, X, 3, &abc[0][0], &abc[0][1], &abc[0][2] );
```

```
    X[0] = xy[2][1];
    Y[0] = 1;
    X[2] = xy[4][1];
    Y[2] = height - 1;
    ab[1][1] = (xy[2][1] - xy[4][1]) / (xy[2][0] - xy[4][0] + 0.0001);
    ab[1][0] = xy[2][1] - ab[1][1] * xy[2][0];
    xy[3][0] = (xy[3][1] - ab[1][0]) / ab[1][1];
    X[1] = xy[3][1];
    Y[1] = height / 2;
    least_squares2( Y, X, 3, &abc[1][0], &abc[1][1], &abc[1][2] );

    X[0] = xy[6][0];
    Y[0] = 1;
    X[2] = xy[4][0];
    Y[2] = width - 1;

    ab[2][1] = (xy[6][1] - xy[4][1]) / (xy[6][0] - xy[4][0] + 0.0001);
    ab[2][0] = xy[6][1] - ab[2][1] * xy[6][0];
    xy[5][1] = ab[2][0] + ab[2][1] * xy[5][0];
    X[1] = xy[5][0];
    Y[1] = width / 2;
    least_squares2( Y, X, 3, &abc[2][0], &abc[2][1], &abc[2][2] );

    X[0] = xy[0][1];
    Y[0] = 1;
    X[2] = xy[6][1];
    Y[2] = height - 1;
    ab[3][1] = (xy[0][1] - xy[6][1]) / (xy[0][0] - xy[6][0] + 0.0001);
    ab[3][0] = xy[0][1] - ab[3][1] * xy[0][0];
    xy[7][0] = (xy[7][1] - ab[3][0]) / ab[3][1];
    X[1] = xy[7][1];
    Y[1] = height / 2;
    least_squares2( Y, X, 3, &abc[3][0], &abc[3][1], &abc[3][2] );
}

/*! Расчет коэффициентов функций для отрисовки линий калибровочной сетки.
 *
 * \param[in] abc       - коэффициенты функции плотности линий
 * \param[out] ab_vert  - возвращаются коэффициенты функций для
 * построения вертикальных линий сетки
 * \param[out] ab_hor   - возвращаются коэффициенты функций для
 * построения горизонтальных линий сетки
 * \param[in] ab        - коэффициенты функций линий вдоль которых
 * расположены точки
 * \param[in] width     - ширина изображения
 * \param[in] height    - высота изображения
 */
void CImgView::CalculateLines( double abc[4][3], double
ab_vert[MAX_LINES][2],
        double ab_hor[MAX_LINES][2], double ab[4][2], int width, int height
)
```

```
{
    double a, b;
    double x, y, x1, y1, x2, y2;
    double dx, dy;
    int i;

    dx = ((double) width) / (double) (nVLines - 1);
    dy = ((double) height) / (double) (nHLines - 1);

    /* vertical lines */
    for( i = 0; i < nVLines; i++ )
    {
        x = dx * i;
        x1 = abc[0][0] + x * abc[0][1] + x * x * abc[0][2];
        y1 = ab[0][0] + ab[0][1] * x1;

        x2 = abc[2][0] + x * abc[2][1] + x * x * abc[2][2];
        y2 = ab[2][0] + ab[2][1] * x2;

        b = (x1 - x2) / (y1 - y2);
        a = x1 - b * y1;
        ab_vert[i][0] = a;
        ab_vert[i][1] = b;
    }

    /* horizontal lines */
    for( i = 0; i < nHLines; i++ )
    {
        y = dy * i;
        y1 = abc[3][0] + y * abc[3][1] + y * y * abc[3][2];
        x1 = (y1 - ab[3][0]) / ab[3][1];

        y2 = abc[1][0] + y * abc[1][1] + y * y * abc[1][2];
        x2 = (y2 - ab[1][0]) / ab[1][1];

        b = (y1 - y2) / (x1 - x2);
        a = y1 - b * x1;
        ab_hor[i][0] = a;
        ab_hor[i][1] = b;
    }
}

/*! Установить количество горизонтальных линий калибровочной сетки.
 *
 * \param[in] n - количество линий
 */
void CImgView::setHLines( int n )
{
    nHLines = n;
    makePixmap();
    DrawCPoints();
```

```
    updateFrame();
}

/*! Установить количество вертикальных линий калибровочной сетки.
 *
 * \param[in] n - количество линий
 */
void CImgView::setVLines( int n )
{
    nVLines = n;
    makePixmap();
    DrawCPoints();
    updateFrame();
}

/*! Функция отрисовки калибровочной сетки на изображении.
 *
 * \param[in] qp        - QPainter
 * \param[in] ab_vert   - коэффициенты функций для вертикальных линий
 * \param[in] ab_hor    - коэффициенты функций для горизонтальных линий
 */
void CImgView::DrawCLines( QPainter* qp, double ab_vert[MAX_LINES][2],
        double ab_hor[MAX_LINES][2] )
{
    int i, j, k;

    qp->setPen( QColor( 0, 196, 0 ) );
    qp->setBrush( QColor( 0, 196, 0 ) );

    for( k = 0; k < nHLines; k++ )
    {
        /* horizontal line */
        for( i = 0; i < m_nCols; i++ )
        {
            j = (int) (ab_hor[k][0] + i * ab_hor[k][1] + 0.5);
            if( (j > 0) && (j < m_nRows) )
            {
                qp->drawPoint( i, j );
                if( j < (m_nRows - 1) )
                    qp->drawPoint( i, j + 1 );
            }
        }
    }

    for( k = 0; k < nVLines; k++ )
    {
        /* vertical line */
        for( i = 0; i < m_nRows; i++ )
        {
            j = (int) (ab_vert[k][0] + i * ab_vert[k][1] + 0.5);
            if( (j > 0) && (j < m_nCols) )
```

```
            {
                qp->drawPoint( j, i );
                if( j < (m_nCols - 1) )
                    qp->drawPoint( j + 1, i );
            }
        }
    }
}

/*! Функция отрисовки точек(узлов) для задания калибровочной сетки.
 *
 */
void CImgView::DrawCPoints( void )
{
    int i;

    QPainter qp( &m_pixmap );

    CalculateFunctions( xy, abc, ab, m_nCols, m_nRows );
    CalculateLines( abc, ab_vert, ab_hor, ab, m_nCols, m_nRows );
    DrawCLines( &qp, ab_vert, ab_hor );
    qp.setPen( QColor( 196, 0, 0 ) );
    qp.setBrush( QColor( 196, 0, 0 ) );
    for( i = 0; i < 8; i++ )
        qp.drawPie(
                (int) (xy[i][0] - m_nCols / 100 / 2), (int)(xy[i][1]-
m_nCols/100/2),
                m_nCols/100, m_nCols/100, 0, 16*360 )
;
    qp.end();
}

/*! Расчет коэффициентов функции второго порядка описывающей кривую
 по N точкам.
 *
 * \param[in] x         - массив X координат точек
 * \param[in] y         - массив Y координат точек
 * \param[in] n         - количество точек, описывающих кривую
 * \param[in] a0        - возвращается коэффициент рассчитанной функции
 * \param[in] a1        - возвращается коэффициент рассчитанной функции
 * \param[in] a2        - возвращается коэффициент рассчитанной функции
 */
void CImgView::least_squares2( double *x, double *y, int n, double *a0,
        double *a1, double *a2 )
{
    double c0, c1, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12;
    double x1, x2;
    int i;

    c0 = 0.0;
    c1 = 0.0;
    c3 = 0.0;
```

```
    c4 = 0.0;
    c5 = 0.0;
    c6 = 0.0;
    c7 = 0.0;

    for( i = 0; i < n; ++i )
    {
        x1 = x[i];
        x2 = x[i] * x[i];
        c0 += x1;
        c1 += x2;
        c3 += x1 * x2;
        c4 += x1 * y[i];
        c5 += x2 * x2;
        c6 += x2 * y[i];
        c7 += y[i];
    }

    c8 = c1 - c0 * c0 / n;
    c9 = c3 - c0 * c1 / n;
    c10 = (c0 * c1 / n - c3) / c8;
    c11 = c9 * c10 - c1 * c1 / n + c5;
    c12 = c6 - c1 * c7 / n - c4 * c9 / c8 + c0 * c7 * c9 / (n * c8);

    *a2 = c12 / c11;
    *a1 = (c4 - c0 * c7 / n + *a2 * c0 * c1 / n - *a2 * c3) / c8;
    *a0 = c7 / n - *a1 * c0 / n - *a2 * c1 / n;
}

/*! Отрисовка линий калибровочной сетки в заданном буфере с изображением.
 *
 * \param[in] buf       - буфер с изображением
 * \param[in] width     - ширина изображения
 * \param[in] height    - высота изображения
 * \param[in] color     - значение градации серого для отрисовки линий
 */
void CImgView::DrawBufLines( unsigned char* buf, int width, int height,
        unsigned char color )
{
    int i, j, k;

    for( k = 0; k < nHLines; k++ )
    {
        /* horizontal line */
        for( i = 0; i < width; i++ )
        {
            j = (int) (ab_hor[k][0] + i * ab_hor[k][1] + 0.5);
            if( (j > 0) && (j < height) )
            {
                buf[j * width + i] = color;
                if( j < (height - 1) )
```

```
                    buf[(j + 1) * width + i] = color;
                }
            }
        }

        for( k = 0; k < nVLines; k++ )
        {
            /* vertical line */
            for( i = 0; i < height; i++ )
            {
                j = (int) (ab_vert[k][0] + i * ab_vert[k][1] + 0.5);
                if( (j > 0) && (j < width) )
                {
                    buf[i * width + j] = color;
                    if( j < (width - 1) )
                        buf[i * width + j + 1] = color;
                }
            }
        }
}
```

imgview.h

Класс для вывода изображения 8 bpp (256 градаций серого).

```
/*!
 * \file imgview.h
 * \brief Отрисовка изображений, класс CImgView
 * \author vav
 * \date   created 12-Feb-2007
 */


#ifndef __IMGVIEW_H__
#define __IMGVIEW_H__

#include <q3scrollview.h>
#include <qimage.h>
#include <qpixmap.h>
//Added by qt3to4:
#include <QMouseEvent>

/**Максимальное количество линий калибровочной сетки*/
#define MAX_LINES       100

/*!\class QScrollView
 * QT Scrolling viewclass.
 */

/*!\class CImgView
 * Отрисовка изображений.
 *
```

```
 * Данный класс используется для
 *
 * - отрисовки изображений в формате 8bpp, 256 градаций серого
 * - выбора прямоугольной области на изображении
 * - отрисовки окружностей на изображении
 * - задания отрисовки калибровочной сетки на изображении
 * - оценки угла наклона деталей изображения относительно горизонтали
 * - оценки смещения деталей изображения относительно верхнего левого
 * угла изображения
 *
 */
class CImgView : public Q3ScrollView
{
      Q_OBJECT          /*!<QObject base object*/

      public:
      CImgView(QWidget *parent=0, const char *name=0, int nCols=0, int
nRows=0, bool bViewAll=true);
      virtual ~CImgView();

      void setColorMode( int color );
      void makeImage( int cx, int cy );

      void makePixmap();
      void makePixmap(unsigned char * pData, int buf_cx );
      void makePixmap(unsigned char * pData, int cx, int cy);
      void makePixmap(unsigned char * pData, int left, int top, int cx, int
cy, int buf_cx );
      void updateFrame();

      void DrawBufLines( unsigned char* buf, int width, int height, unsigned
char color );
      void getSelectedRect( int* left, int* top, int* width, int* height );
      void setSelectedRect( int left, int top, int width, int height );
      void setViewAll(bool bViewAll=true);
      void setImageSize( int nCols, int nRows);
      void setZoom( bool aZoom );
      void setMaxSize();
      QSize sizeHint();

      void DrawCPoints( void );
      void DrawCLines( QPainter* qp, double ab_vert[MAX_LINES][2], double
ab_hor[MAX_LINES][2] );
      void CalculateLines( double abc[4][3], double ab_vert[MAX_LINES][2],
                           double ab_hor[MAX_LINES][2], double ab[4][2],
                           int width, int height );
      void CalculateFunctions( double xy[8][2], double abc[4][3], double
ab[4][2],
                               int width, int height );
      void least_squares2( double *x, double *y, int n,
                           double *a0, double *a1, double *a2 );
```

```
void drawCalibr( bool en );
void drawPoints( bool en );
void drawMouse( bool en );
void drawRect( bool en );
void drawMouseLine( void );
void drawDegree( bool en );
void drawDegreeLine( void );
void setPoint( int num );
void drawImagePoint( int x, int y );
void setHLines( int n );
void setVLines( int n );

/**Объект для непосредственной отрисовки на экране*/
QPixmap m_pixmap;

/**Координаты точек для отрисовки двух окружностей*/
QPoint points[2][2];

/**Координаты указателя мыши относительно окна объекта*/
int mouse_x;

/**Координаты указателя мыши относительно изображения*/
int mouse_img_x;

/** Координата X первой точки для оценки угла наклона объекта
 * относительно горизонтали*/
int degree_x1;

/** Координата Y первой точки для оценки угла наклона объекта
 * относительно горизонтали*/
int degree_y1;

/** Координата X второй точки для оценки угла наклона объекта
 * относительно горизонтали*/
int degree_x2;

/** Координата Y второй точки для оценки угла наклона объекта
 * относительно горизонтали*/
int degree_y2;

/**Угол наклона объекта относительно горизонтали*/
double degree;

/**Коэффициент масштабирования изображения для того что бы
 * вписать его в окно*/
double zoom_scale;

/**Координата X верхнего левого угла выбранной области изображения*/
int rect_x1;

/**Координата Y верхнего левого угла выбранной области изображения*/
```

```
      int rect_y1;

      /**Координата X нижнего правого угла выбранной области изображения*/
      int rect_x2;

      /**Координата Y нижнего правого угла выбранной области изображения*/
      int rect_y2;

      /**Цвет отрисовки изображения на экране
       * - -1 - градации серого
       * - 0 - градации красного
       * - 1 - градации зеленого
       * - 2 - градации синего
       */
      int color_mode;

      /**Номер устанавливаемого узла калибровочной сетки*/
      int num;

      /**Коэффициенты для функций распределения плотности
       * калибровочных линий вдоль четырех сторон изображения*/
      double abc[4][3];

      /**Коэффициенты функций для построения четырех линий вдоль
       * которых расположены задаваемые узлы калибровочной сетки*/
      double ab[4][2];

      /** Координаты задаваемых узлов калибровочной сетки*/
      double xy[8][2];

      /**Коэффициенты функций описывающих вертикальные линии калибровочной
сетки*/
      double ab_vert[MAX_LINES][2];

      /**Коэффициенты функций описывающих горизонтальные линии калибровочной
сетки*/
      double ab_hor[MAX_LINES][2];

      /**Количество вертикальных линий калибровочной сетки*/
      int nVLines;

      /**Количество горизонтальных линий калибровочной сетки*/
      int nHLines;


   protected:
      void drawContents(QPainter * paint, int clipx, int  clipy, int clipw,
int cliph );
      virtual void contentsMousePressEvent(QMouseEvent*);
      virtual void contentsMouseReleaseEvent(QMouseEvent*);
      virtual void contentsMouseMoveEvent(QMouseEvent*);
      void updateRect( );
```

```
    /**Количество колонок в изображении*/
    int m_nCols;

    /**Количество строк в изображении*/
    int m_nRows;

    /**Объект для манипуляции с изображением*/
    QImage* img;

    /**Режим масштабирования изображения*/
    bool zoom;

    /**количество отображаемых окружностей на изображении*/
    int npoint;

    /**Режим отрисовки окружностей на изображении*/
    bool draw_points;

    /**Режим отрисовки вертикальной линии в месте курсора мыши*/
    bool draw_mouse;

    /**Режим отрисовки линии для оценки угла наклона объекта на
     * изображении относительно горизонтали*/
    bool draw_degree;

    /**Режим отрисовки выбранной прямоугольной области на изображении*/
    bool draw_rect;

    /**Режим задания, расчета и отрисовки калибровочной сетки*/
    bool draw_calibr;
signals:

    /*! Mouse moved signal
     *
     * \param[in]     x       - mouse X-coordinate
     */
    void mouseChanged( int x );

    /*! Rectangle changed signal
     *
     * \param[in] x   - rectangle X-coordinate
     * \param[in] y   - rectangle Y-coordinate
     */
    void rectChanged( int x, int y );

    /*! Rectangle width changed signal
     *
     * \param[in] w   - rectangle width
     */
    void rectChangedWidth( int w );
```

```
      /*! Rectangle height changed signal
       *
       * \param[in] h   - rectangle height
       */
      void rectChangedHeight( int h );


      /*! Degree changed signal
       *
       * \param[in] x   - the new degree
       */
      void degreeChanged( double x );

   private:

      /**Режим отображения всего изображения в окне или только его
       * части с перемещением области отображения с помощью слайдеров*/
      bool m_bViewAll;

      /**Переменная отображающая состояние перемещение мыши с нажатой левой
кнопкой*/
      bool m_bMoving;

      /**Начальная точка перемещения мыши с нажатой левой кнопкой*/
      QPoint m_posMovingStart;
};

#endif // __IMGVIEW_H__
```

lschandle.cpp

Пример использования библиотеки, Callback-функции.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#if !defined(__WIN32__) && !defined(WIN32)
#include <unistd.h>
#endif

#include <qapplication.h>
#include <qevent.h>

#include <lsclient.h>

#include "lschandle.h"

/*
 Clearing live preview window callback
 It is called from lsclient for clearing preview window
```

```
 */
int clearPreview( LSC_HNDL handle )
{
        LSC_HANDLE* hndl = (LSC_HANDLE*) handle;
        QCustomEvent* event = new QCustomEvent( (QEvent::Type) (QEvent::User
+ CLEARPREVIEW_EVENT) );

        /*Clearing live preview window*/
        QApplication::postEvent( (QObject*) hndl->visual_ptr, event );

        return 0;
}

/*
 Draw live preview (slaps/touch or roll)  window
 It's called from lsclient for drawing live preview image
 Image is 8 bits per pixel color depth, 256 gray
 Begins with the top left corner
 finger_num - number of fingerprint
 */
int drawPreview( LSC_HNDL handle, int finger_num, unsigned char *buf, int
width, int height, int error_code )
{
        LSC_HANDLE* hndl = (LSC_HANDLE*) handle;
        QCustomEvent* event = new QCustomEvent( (QEvent::Type) (QEvent::User
+ DRAWPREVIEW_EVENT) );

        /*Copy live preview image for drawing*/
        (void) finger_num;

        if( hndl->preview_ready )
                return 0;

        if( !hndl->preview_img )
                return 0;

        hndl->preview_lock->lock();
        hndl->preview_error = error_code;
        hndl->preview_ready = 1;

        /**Store the current size of the preview*/
        hndl->preview_width = width;
        hndl->preview_height = height;

        /**Store a preview image*/
        memcpy( hndl->preview_img, buf, width * height );

        hndl->preview_lock->unlock();
        QApplication::postEvent( (QObject*) hndl->visual_ptr, event );

        return 0;
}
```

```
/*
 send command to lsclient
 It's calledfrom lsclient
 */
LSC_COMMAND getCommand( LSC_HNDL handle )
{
        LSC_HANDLE* hndl = (LSC_HANDLE*) handle;
        LSC_COMMAND cmd;

        cmd = hndl->command;
        hndl->command = LSC_CMD_NOP;

        return cmd;
}

/*
 Send work directory to lsclient
 It's called from lsclient and used for reading ini-files
 */
char* getWorkDir( LSC_HNDL handle )
{
        LSC_HANDLE* hndl = (LSC_HANDLE*) handle;

        return hndl->work_dir;
}

/*
 Initialization done
 It's caled from lsclient and designates end of initialization of the device
 */
void initDone( LSC_HNDL handle )
{
        LSC_HANDLE* hndl = (LSC_HANDLE*) handle;
        QCustomEvent* event = new QCustomEvent( (QEvent::Type) (QEvent::User
+ INITDONE_EVENT) );

        QApplication::postEvent( (QObject*) hndl->visual_ptr, event );
}

/*
 send pressed scanner keypad from lsclient to upper software
 It's called from lsclient
 */
int processButton( LSC_HNDL handle, LSC_BUTTONS button )
{
        LSC_HANDLE* hndl = (LSC_HANDLE*) handle;
        QCustomEvent* event = new QCustomEvent( (QEvent::Type) (QEvent::User
+ PROCESSBUTTON_EVENT) );

        printf("button=%d\n", button);
        hndl->buttons = button;
```

```
        QApplication::postEvent( (QObject*) hndl->visual_ptr, event );
        return 0;
}

/*
 Indicate change of lsclient state
 It's called from lsclient when state is changed
 */
void stateChanged( LSC_HNDL handle, LSC_STATE state )
{
        LSC_HANDLE* hndl = (LSC_HANDLE*) handle;
        QCustomEvent* event = new QCustomEvent( (QEvent::Type) (QEvent::User
+ STATECHANGED_EVENT) );

        if( state == LSC_STAT_ERROR )
                printf( "Error!\n" );

        hndl->state = state;
        QApplication::postEvent( (QObject*) hndl->visual_ptr, event );
}

/*
 Send ready rolled image from lsclient to upper software
 Image is 8 bits per pixel, 256 gray, 500ppi
 finger_num - number of finger
 0 - right thum
 1 - right index
 ...
 5 - left thumb
 6- left index
 ....
 9 - left little
 10 - left four plains
 11 - left thumb plain
 12 - right thumb plain
 13 - right four plains
 14 - left palm
 15 - right palm
 */
int takeImage( LSC_HNDL handle, int finger_num, unsigned char *buf, int
width, int height, unsigned int roll_errors )
{
        LSC_HANDLE* hndl = (LSC_HANDLE*) handle;
        QCustomEvent* event = new QCustomEvent( (QEvent::Type) (QEvent::User
+ TAKEIMAGE_EVENT) );
        QMutexLocker _lock( hndl->image_lock );

        hndl->roll_errors = roll_errors;
        hndl->finger_num = finger_num;
        if( hndl->finger_image )
        {
                free( hndl->finger_image );
```

```
                hndl->finger_image = NULL;
        }
        hndl->finger_width = width;
        hndl->finger_height = height;

        if( buf != NULL )
        {
                hndl->finger_image = (unsigned char*) malloc( width * height
);
                if( !hndl->finger_image )
                {
                        printf( "Error allocating %d bytes!\n", width *
height );
                        return -1;
                }

                memcpy( hndl->finger_image, buf, width * height );
        }

        _lock.unlock();
        QApplication::postEvent( (QObject*) hndl->visual_ptr, event );
        return 0;
}

void drawLCD( LSC_HNDL handle )
{
        LSC_HANDLE* hndl = (LSC_HANDLE*) handle;
        int ret;

        if( !hndl->lcd )
                return;

        if( hndl->lcd_ready )
        {
                ret = PLCD_WriteImage( hndl->lcd, hndl->lcd_image );
                if( ret != 0 )
                {
                        exit( -1 );
                }
                hndl->lcd_ready = 0;
        }
}

void removeFinger( LSC_HNDL handle )
{
        LSC_HANDLE* hndl = (LSC_HANDLE*) handle;
        QCustomEvent* event = new QCustomEvent( (QEvent::Type) (QEvent::User
+ REMOVE_FINGER_EVENT) );
        QApplication::postEvent( (QObject*) hndl->visual_ptr, event );
}

void sendCommand( LSC_HNDL handle, LSC_COMMAND command )
```

```
{
        LSC_HANDLE* hndl = (LSC_HANDLE*) handle;

        while( hndl->command != LSC_CMD_NOP )
        {
#ifdef _WIN32
                SleepEx( 10, TRUE );
#else
                usleep( 10000 );
#endif
        }

        hndl->command = command;
}

void ScanComplete( LSC_HNDL handle )
{
        LSC_HANDLE* hndl = (LSC_HANDLE*) handle;
        QCustomEvent* event = new QCustomEvent( (QEvent::Type) (QEvent::User
+ SCAN_COMPLETE_EVENT) );
        QApplication::postEvent( (QObject*) hndl->visual_ptr, event );
}

void AddStat( LSC_HNDL handle, char* data, int tag, int mult )
{
        fprintf( stdout, "stat: handle=%p\ttag=%d\tmult=%d\t%s\n", handle,
tag, mult, data );
}
```

lschandle.h

Пример использования библиотеки, описание объекта приложения LSClient::Client и Callback-функций.

```
#ifndef _lschandle_h_
#define _lschandle_h_

#define CLEARPREVIEW_EVENT      0
#define DRAWPREVIEW_EVENT       1
#define TAKEIMAGE_EVENT         2
#define PROCESSBUTTON_EVENT     3
#define INITDONE_EVENT          4
#define STATECHANGED_EVENT      5
#define REMOVE_FINGER_EVENT     6
#define SCAN_COMPLETE_EVENT     7

#include <QMutex>

#include "libplcd.h"
#include "lsclient.h"

/* Any structure with a set of parameters */
```

```
typedef struct LSC_HANDLE
{
  LSC_COMMAND command;
  LSC_STATE state;
  LSC_BUTTONS buttons;
  int preview_width;
  int preview_height;
  unsigned char* preview_img;
  int preview_ready;
  int finger_num;
  int finger_width;
  int finger_height;
  unsigned char* finger_image;
  int roll_errors;
  char work_dir[1024];
  void* visual_ptr;
  unsigned char* lcd_image;
  int lcd_ready;
  PLCD_handle lcd;
  int lcd_width;
  int lcd_height;
  int preview_error;
  BOOL touch_mode;
  QMutex* image_lock;
  QMutex* preview_lock;
}LSC_HANDLE;


/* callback functions */

/*Clear preview window */
int clearPreview( LSC_HNDL handle );

/*Draw preview image*/
int drawPreview( LSC_HNDL handle, int img_class, unsigned char *buf, int
width, int height, int error );

/*Get command to lsclient thread*/
LSC_COMMAND getCommand(LSC_HNDL handle);

/*Get work dir.*/
char* getWorkDir(LSC_HNDL handle);

/*Initialization done*/
void initDone(LSC_HNDL handle);

/*Processing buttons*/
int processButton(LSC_HNDL handle, LSC_BUTTONS button);

/*Chenge state of lsclient thread*/
void stateChanged(LSC_HNDL handle, LSC_STATE state );
```

```
/*Take final image*/
int takeImage(LSC_HNDL handle, int finger_num, unsigned char *buf, int
width, int height, unsigned int roll_errors );

/* You can clean finger from baseline */
void removeFinger( LSC_HNDL handle );

/* send any data to scanner`s LCD in this function */
void drawLCD( LSC_HNDL handle );

/*Send command to the lsclient library thread.*/
void sendCommand(LSC_HNDL handle, LSC_COMMAND command );

/*Scan Complete Callback, called from the lsclinet thread after all of
fingerprints is captured and
*       transfered from the CrossMatch wireless device.
*/
void ScanComplete( LSC_HNDL handle );

void AddStat( LSC_HNDL handle, char* data, int tag, int mult );

#endif /* _lschandle_h_ */
```

main.cpp

Пример использования библиотеки, функция **main()**

```
#include <stdio.h>
#include <stdlib.h>
#include <qapplication.h>
#include <qlabel.h>
#include <errno.h>
#include "mainform.h"

int main(int argc, char ** argv)
{
        QApplication a( argc, argv );
        a.setStyle( "windows" );
        int scanner_number = 0;
        /**The first argument might be nuber of scanner.*/
        if( argc > 1 )
        {
                scanner_number = atoi( argv[1] );
        }

        std::string qt_path = "QTDIR";
        char* p = getenv(qt_path.c_str());
        if(p && p[0])
        {
                QString addPath = p;
                if( !addPath.isEmpty() ) {
                        QApplication::addLibraryPath( p );
```

```
                }
        }

        mainForm form( scanner_number );

        /** Main form */
        form.reparent( NULL,
                        Qt::WStyle_Customize | Qt::WStyle_NormalBorder |
Qt::WStyle_SysMenu | Qt::WStyle_Title | Qt::WStyle_Maximize
                                        | Qt::WStyle_Minimize,
form.geometry().topLeft() );

        form.showMaximized();

        a.connect( &a, SIGNAL( lastWindowClosed() ), &a, SLOT( quit() ) );

        return a.exec();
}
```

mainform.cpp

Пример использования библиотеки, главная форма.

```
/****************************************************************************
*
 ** ui.h extension file, included from the uic-generated form
implementation.
 **
 ** If you want to add, delete, or rename functions or slots, use
 ** Qt Designer to update this file, preserving your code.
 **
 ** You should not define a constructor or destructor in this file.
 ** Instead, write your code in functions called init() and destroy().
 ** These will automatically be called by the form's constructor and
 ** destructor.

*****************************************************************************
*/
#include <QMutexLocker>
#include <QMessageBox>
#include <QFileDialog>
#include <QImage>

#include "mainform.h"
#include "select_model.h"
#include "lcd.h"

mainForm::mainForm(int _number, QWidget *parent, Qt::WindowFlags f) :
                QDialog( parent, f )
{
        mNumber = _number;
        fake_supported = false;
        lssdk_started = false;
```

```
        message = NULL;
        preview = NULL;
        lcd_lut = NULL;

        setupUi( this );

        startButton->setEnabled( false );
        stopButton->setEnabled( false );
        shutdownButton->setEnabled( false );
        initButton->setEnabled( false );
        saveButton->setEnabled( false );
        clearButton->setEnabled( false );
        licenseButton->setEnabled( false );
        compressButton->setEnabled( false );
        compressionBox->setEnabled( false );
        decompressButton->setEnabled( false );
        checkEncodeButton->setEnabled( false );
        checkQualityButton->setEnabled( false );
        modeBox->setEnabled( false );
        greenBox->setEnabled( false );
        redBox->setEnabled( false );
        fingerBox->setEnabled( false );

        init();
}

mainForm::~mainForm()
{
        destroy();
}

/*Initialize the LSSDK*/
void mainForm::initSlot()
{
        LSC_ERROR ret;

        /*Check if initialized already*/
        if( lssdk_started )
                return;

        if( !message )
                message = new messageForm( this, "message", FALSE );

        message->textLabel->setText( "Scanner initialization in progress..."
);
        message->show();
        qApp->processEvents();

        /* initialization of LSClient thread. */
        ret = LSCInit( &cl, device, mNumber );
        if( ret != LSC_NO_ERROR )
        {
```

```
                QMessageBox::critical( this, "Error", "Error in LSCInit " +
QString::number( ret ) );
                exit( -1 );
        }

        if( lsc_handle.state == LSC_STAT_ERROR )
        {
                QMessageBox::critical( this, "Error", "Error after LSCInit "
+ QString::number( LSCGetError( &cl ) ) );
                LSCShutDown( &cl );
                exit( -1 );
        }

        // For enable the fake finger detection uncomment the next string
        //fake_supported = LSC_EnableFakeDetection( &cl, false );

        LSCSetTouchMode( &cl, lsc_handle.touch_mode );
        LSC_SetUsbType( &cl, FALSE );

        /*You could test a few modes of capture and choose more suitable for
you.*/
//      LSCSetCaptureMode( &cl, LSC_CAPTURE_CLEAN );
//      LSCSetCaptureMode( &cl, LSC_CAPTURE_AUTO );
        LSCSetCaptureMode( &cl, LSC_CAPTURE_AUTO_CLEAN );
//      LSCSetCaptureMode( &cl, LSC_CAPTURE_COMMAND );

        /* Open connection to LCD */
        if( LCD_Open( &lsc_handle.lcd, lcd_lut, &cl ) != 0 )
        {
                lsc_handle.lcd = NULL;
                lcdPowerBox->hide();
                lcdLightSlider->hide();
                lcdLightLabel->hide();
        }

        if( lsc_handle.lcd )
        {
                LCD_GetSize( lsc_handle.lcd, &lsc_handle.lcd_width,
&lsc_handle.lcd_height );

                if( lsc_handle.lcd_image != NULL )
                        free( lsc_handle.lcd_image );

                /* Allocating lcd image buffer */
                lsc_handle.lcd_image = (unsigned char*) malloc(
lsc_handle.lcd_width * lsc_handle.lcd_height );
                if( !lsc_handle.lcd_image )
                {
                        QMessageBox::critical( this, "Error", "Out of
memory!" );
                        exit( -1 );
                }
```

```
                memset( lsc_handle.lcd_image, 0, lsc_handle.lcd_width *
lsc_handle.lcd_height );

                /* Make LCD splash image */
                LCD_MakeSplash( lsc_handle.lcd, lsc_handle.lcd_image );

                /* Set LCD image ready flag */
                lsc_handle.lcd_ready = 1;

                /* send splash image to LCD */
                drawLCD( &lsc_handle );

                /* LCD power on */
                lcdPowerBox->setChecked( TRUE );

                /* Set maximum LCD light */
                lcdLightSlider->setValue( LCD_MAX_LIGHT );
        }

        /* start lsclient thread */
        ret = LSCStart( &cl );
        if( ret != LSC_NO_ERROR )
        {
                QMessageBox::critical( this, "Error", "Error in LSCStart " +
QString::number( ret ) );
                LSCShutDown( &cl );
                free( lsc_handle.preview_img );
                exit( -1 );
        }

        if( lsc_handle.state == LSC_STAT_ERROR )
        {
                QMessageBox::critical( this, "Error", "Error after LSCStart
" + QString::number( LSCGetError( &cl ) ) );
                LSCShutDown( &cl );
                exit( -1 );
        }

        lssdk_started = true;
}

/*Shutdown the LSSDK*/
void mainForm::shutDownSlot()
{
        /*Check if doesn't initialized*/
        if( !lssdk_started )
                return;

        LSCShutDown( &cl );

        if( lsc_handle.lcd )
                LCD_Close( lsc_handle.lcd );
```

```
        lsc_handle.lcd = NULL;

        lssdk_started = false;

        if( message )
                message->hide();

        startButton->setEnabled( false );
        stopButton->setEnabled( false );
        shutdownButton->setEnabled( false );
        initButton->setEnabled( true );
        saveButton->setEnabled( false );
        clearButton->setEnabled( false );
        licenseButton->setEnabled( false );
        compressButton->setEnabled( false );
        compressionBox->setEnabled( false );
        decompressButton->setEnabled( false );
        checkEncodeButton->setEnabled( false );
        checkQualityButton->setEnabled( false );
        modeBox->setEnabled( false );
        greenBox->setEnabled( false );
        redBox->setEnabled( false );
        fingerBox->setEnabled( false );
}

/* Send start roll command to lsclient */
void mainForm::startSlot()
{
        sendCommand( &lsc_handle, LSC_CMD_ROLL );
}

/* Send stop  command to lsclient */
void mainForm::stopSlot()
{
        sendCommand( &lsc_handle, LSC_CMD_STOP );
}

/* Change the finger */
void mainForm::fingerSlot(int finger)
{
        sendCommand( &lsc_handle, (LSC_COMMAND) ( LSC_CMD_FINGERS + finger -
1 ) );
        if( finger < 11 )
                fingerLabel->setText( "Finger " + QString::number( finger )
+ " Roll mode" );
        else
                fingerLabel->setText( "Finger " + QString::number( finger )
+ " Touch mode" );

        LCD_MakeSplash( lsc_handle.lcd, lsc_handle.lcd_image );
        lsc_handle.lcd_ready = 1;
```

```
}

/* Send the clear baseline command to lsclient */
void mainForm::clearSlot()
{
        sendCommand( &lsc_handle, LSC_CMD_CLEAR );
}

/* Send the Exit command to lsclient */
void mainForm::exitSlot()
{
        sendCommand( &lsc_handle, LSC_CMD_EXIT );
}

/* init the main form */
void mainForm::init()
{
        selecScannerSlot();

        message = NULL;
        lcd_lut = NULL;

        preview = new previewForm( this );

        memset( &lsc_handle, 0, sizeof( lsc_handle ) );
        memset( &cl, 0, sizeof( cl ) );

        lsc_handle.image_lock = new QMutex();
        lsc_handle.preview_lock = new QMutex();

        lsc_handle.lcd = NULL;

        /* Allocatind LCD lut array */
        lcd_lut = (unsigned char*) malloc( 256 * 3 );
        if( !lcd_lut )
        {
                QMessageBox::critical( this, "Error", "Out of memory!" );
                exit( -1 );
        }
        memset( lcd_lut, 0, 256 * 3 );

        /* setup  LSClient structure */
        cl.ClearPreview = clearPreview; /* clear preview window callback */
        cl.Client = &lsc_handle; /* pointer to our structure */
        cl.debug_level = 0; /* setup debug information level from 0(no
debug) to 9 - maximum debug level */
        cl.DrawPreview = drawPreview; /* draw preview image callback */
        cl.GetCommand = getCommand; /* get command callback */
        cl.GetWorkDir = getWorkDir; /* get work directory callback */
        cl.InitDone = initDone; /* initialization done callback */
        cl.ProcessButton = processButton; /* processing buttons callback */
        cl.StateChanged = stateChanged; /* state change callback */
```

```
        cl.TakeImage = takeImage; /* take final image callback */
        cl.RemoveFinger = removeFinger; /* clear finger from baseline
callback */
        cl.DrawLCD = drawLCD; /* draw lcd image callback */
        cl.ScanComplete = ScanComplete; /*Scan complete callback*/
        cl.AddScanStat = AddStat; /*Scan statistic callback*/

        strcpy( lsc_handle.work_dir, "./" );

        lsc_handle.visual_ptr = (void*) this;

        initSlot();
}

void mainForm::destroy()
{

        if( message )
        {
                delete message;
                message = NULL;
        }

        shutDownSlot();

        if( lcd_lut )
        {
                free( lcd_lut );
                lcd_lut = NULL;
        }

        if( lsc_handle.lcd_image )
        {
                free( lsc_handle.lcd_image );
                lsc_handle.lcd_image = NULL;
        }

        if( lsc_handle.preview_img )
        {
                free( lsc_handle.preview_img );
                lsc_handle.preview_img = NULL;
        }

        if( lsc_handle.finger_image )
        {
                free( lsc_handle.finger_image );
                lsc_handle.finger_image = NULL;
        }

        if( lsc_handle.preview_lock )
                delete lsc_handle.preview_lock;
```

```
        if( lsc_handle.image_lock )
                delete lsc_handle.image_lock;

        if( preview )
                delete preview;

}

void mainForm::drawPreviewSlot()
{
        if( lsc_handle.preview_ready ) /* Check - if we've finger preview
image */
        {
                if( !preview )
                        return;

                /*Clear fake label*/
                fakeLabel->setText( tr( "" ) );
                fakeLabel->setStyleSheet( "" );

                preview->show();

                preview->cimage->setImageSize( lsc_handle.preview_width,
lsc_handle.preview_height );
                preview->cimage->makePixmap( lsc_handle.preview_img,
lsc_handle.preview_width, lsc_handle.preview_height );
                if( lsc_handle.lcd )
                {
                        if( !lsc_handle.lcd_ready ) /* Check - if ready flag
is clear */
                        {
                                /*Prepeare preview image for LCD */
                                LCD_MakeLCDImageGray( lsc_handle.lcd,
lsc_handle.lcd_image, lsc_handle.preview_img,
                                                lsc_handle.preview_width,
lsc_handle.preview_height );

                                /* Set lcd image ready flag */
                                lsc_handle.lcd_ready = 1;
                        }
                }

                lsc_handle.preview_ready = 0;
                preview->cimage->updateFrame();
        }
}

/* Close preview windows */
void mainForm::clearPreviewSlot()
{
        if( preview )
                preview->hide();
}
```

```
/* Initialization done */
void mainForm::initDoneSlot()
{
        int ret;

        if( lsc_handle.state == LSC_STAT_ERROR )
        {
                QMessageBox::critical( this, "Error", "Error after init
done" + QString::number( LSCGetError( &cl ) ) );
                LSCShutDown( &cl );
                exit( -1 );
        }

        /* setup preview image size in roll(fingerprint) mode */
        int _width = width();
        int _height = height();
        ret = LSCSetPreview( &cl, &_width, &_height );
        if( ret != 0 )
        {
                QMessageBox::critical( this, "Error", "Error in
LSCSetPreview" + QString::number( ret ) );
                LSCShutDown( &cl );
                exit( -1 );
        }

        /* setup preview image size in touch(slaps and palms) mode */
        int _swidth = width();
        int _sheight = height();
        ret = LSCGetPreviewSlap( &cl, &_swidth, &_sheight );
        if( ret != 0 )
        {
                QMessageBox::critical( this, "Error", "Error in
LSCSetPreview" + QString::number( ret ) );
                LSCShutDown( &cl );
                exit( -1 );
        }

        /*Select size of the biggest preview image*/
        if( ( _width * _height ) > ( _swidth * _sheight ) )
        {
                lsc_handle.preview_width = _width;
                lsc_handle.preview_height = _height;
        }
        else
        {
                lsc_handle.preview_width = _swidth;
                lsc_handle.preview_height = _sheight;
        }

        if( lsc_handle.preview_img != NULL )
                free( lsc_handle.preview_img );
```

```
        lsc_handle.preview_img = (unsigned char*) malloc(
lsc_handle.preview_width * lsc_handle.preview_height );
        if( lsc_handle.preview_img == NULL )
        {
                QMessageBox::critical( this, "Error", "Out of memory!" );
                LSCShutDown( &cl );
                exit( -1 );
        }

        preview->cimage->setImageSize( lsc_handle.preview_width,
lsc_handle.preview_height );
        preview->cimage->setZoom( FALSE );
        preview->cimage->setViewAll( true );
        preview->cimage->updateGeometry();

        if( message )
        {
                message->hide();
        }

        startButton->setEnabled( true );
        stopButton->setEnabled( true );
        shutdownButton->setEnabled( true );
        initButton->setEnabled( false );
        saveButton->setEnabled( true );
        clearButton->setEnabled( true );
        licenseButton->setEnabled( true );
        compressButton->setEnabled( false );
        compressionBox->setEnabled( false );
        decompressButton->setEnabled( false );
        checkEncodeButton->setEnabled( false );
        checkQualityButton->setEnabled( false );
        modeBox->setEnabled( true );
        greenBox->setEnabled( true );
        redBox->setEnabled( true );
        fingerBox->setEnabled( true );

        startSlot();
        fingerBox->setValue( 1 );
}

void mainForm::processButtonSlot()
{
        if( lsc_handle.buttons == LSC_BTN_OK )
                sendCommand( &lsc_handle, LSC_CMD_TOUCH );
}

/* processing final image */
void mainForm::takeImageSlot()
{
        /**AFIS quality*/
```

```
        int aq = 0;

        /**Visual quality*/
        int vq = 0;

        /**Pressure*/
        float press = 0.0;

        removeFingerLabel->setText( "" );

        if( lsc_handle.finger_image == NULL )
        {
                QMessageBox::critical( this, "Error",
                                "Error in ROLL roll_errors=" +
QString::number( lsc_handle.roll_errors ) );
        }
        else
        {
                if( ( lsc_handle.finger_num == 10 ) || (
lsc_handle.finger_num == 13 ) )
                {
                        segmSlapSlot();
                }

                if( fake_supported )
                {
                        if( lsc_handle.roll_errors & LSC_FAKE_FINGER )
                        {
                                fakeLabel->setText( tr( "FAKE" ) );
                                fakeLabel->setStyleSheet( "background-
color:rgb(255, 128, 128);" );
                        }
                        else
                        {
                                fakeLabel->setText( tr( "LIVE" ) );
                                fakeLabel->setStyleSheet( "background-
color:rgb(128, 255, 128);" );
                        }
                }
                else
                {
                        fakeLabel->setText( tr( "" ) );
                        fakeLabel->setStyleSheet( "" );
                }

                cimage->setZoom( FALSE );
                cimage->setViewAll( false );
                cimage->setImageSize( lsc_handle.finger_width,
lsc_handle.finger_height );
                cimage->makePixmap( lsc_handle.finger_image,
lsc_handle.finger_width, lsc_handle.finger_height );
                cimage->updateFrame();
                if( ( lsc_handle.finger_num ) < 10 )
```

```
                        fingerLabel->setText( "Finger " + QString::number(
lsc_handle.finger_num + 1 ) + " Roll mode" );
                else
                        fingerLabel->setText( "Finger " + QString::number(
lsc_handle.finger_num + 1 ) + " Touch mode" );
                errorsLabel->setText( "Roll errors: " + QString::number(
lsc_handle.roll_errors ) );

                if( LSC_GetAfisQuality( &cl, lsc_handle.finger_image,
lsc_handle.finger_width, lsc_handle.finger_height, 500,
                                &aq, &vq, &press ) != LSC_NO_ERROR )
                {
                        QMessageBox::critical( this, "Error", "Error in
LSC_GetAfisQuality()!" );
                }
                else
                {
                        pqualityLabel->setText( QString( "Papillon quality "
) + QString::number( aq ) );
                        pressureLabel->setText( QString( "Pressure " ) +
QString::number( press ) );
                }

                if( LSC_GetNistQuality( &cl, lsc_handle.finger_image,
lsc_handle.finger_width, lsc_handle.finger_height, 500,
                                &aq, &vq, &press ) != LSC_NO_ERROR )
                {
                        QMessageBox::critical( this, "Error", "Error in
LSC_GetNistQuality()!" );
                }
                else
                {
                        nqualityLabel->setText( QString( "NIST quality " ) +
QString::number( aq ) );
                }
        }
}

/* state change */
void mainForm::stateChangedSlot()
{
        switch( lsc_handle.state )
        {
                case LSC_STAT_INIT:
                {
                        message->textLabel->setText( "Scanner initialization
in progress..." );
                        message->show();
                        stateLabel->setText( "State: LSC_STAT_INIT" );
                        break;
                }
                case LSC_STAT_WAIT:
                {
                        message->hide();
```

```
                        stateLabel->setText( "State: LSC_STAT_WAIT" );
                        break;
                }
                case LSC_STAT_WAIT_FINGER:
                {
                        message->hide();
                        stateLabel->setText( "State: LSC_STAT_WAIT_FINGER"
);
                        break;
                }
                case LSC_STAT_WAIT_REMOVE:
                {
                        message->hide();
                        stateLabel->setText( "State: LSC_STAT_WAIT_REMOVE"
);
                        break;
                }
                case LSC_STAT_ROLL:
                {
                        message->hide();
                        stateLabel->setText( "State: LSC_STAT_ROLL" );
                        break;
                }
                case LSC_STAT_GLUE:
                {
                        message->textLabel->setText( "Preparing image..." );
                        message->show();
                        stateLabel->setText( "State: LSC_STAT_GLUE" );
                        LCD_MakeSplash( lsc_handle.lcd, lsc_handle.lcd_image
);
                        lsc_handle.lcd_ready = 1;
                        break;
                }
                case LSC_STAT_CLEARING:
                {
                        message->textLabel->setText( "Clearing scanner
platen..." );
                        stateLabel->setText( "State: LSC_STAT_CLEARING" );
                        break;
                }
                case LSC_STAT_EXITED:
                {
                        message->hide();
                        stateLabel->setText( "State: LSC_STAT_WAIT_EXITED"
);
                        break;
                }
                case LSC_STAT_ERROR:
                {
                        message->hide();
                        stateLabel->setText( "State: LSC_STAT_ERROR" );
                        QMessageBox::critical( this, "Error", "Error state
error=" + QString::number( LSCGetError( &cl ) ) );
```

```
                        break;
                }
        }
}

/* custom events(from lsclient callbacks) dispatcher */
void mainForm::customEvent(QEvent * e)
{
        switch( (int) e->type() )
        {
                case ( QEvent::User + CLEARPREVIEW_EVENT ):
                {
                        clearPreviewSlot();
                        LCD_MakeSplash( lsc_handle.lcd, lsc_handle.lcd_image
);
                        lsc_handle.lcd_ready = 1;
                        break;
                }
                case ( QEvent::User + DRAWPREVIEW_EVENT ):
                {
                        QMutexLocker _lock( lsc_handle.preview_lock );
                        drawPreviewSlot();
                        break;
                }
                case ( QEvent::User + TAKEIMAGE_EVENT ):
                {
                        QMutexLocker _lock( lsc_handle.image_lock );
                        takeImageSlot();
                        break;
                }
                case ( QEvent::User + PROCESSBUTTON_EVENT ):
                {
                        processButtonSlot();
                        break;
                }
                case ( QEvent::User + INITDONE_EVENT ):
                {
                        initDoneSlot();
                        break;
                }
                case ( QEvent::User + STATECHANGED_EVENT ):
                {
                        stateChangedSlot();
                        break;
                }
                case ( QEvent::User + REMOVE_FINGER_EVENT ):
                {
                        removeFingerSlot();
                        break;
                }
                case ( QEvent::User + SCAN_COMPLETE_EVENT ):
                {
```

```
                        sendResult();
                        break;
                }
        }
}

void mainForm::lcdLightSlot(int light)
{
        if( lsc_handle.lcd )
                PLCD_Light( lsc_handle.lcd, light );
}

void mainForm::lcdPowerSlot(bool power)
{
        if( lsc_handle.lcd )
        {
                if( power )
                        PLCD_PowerOn( lsc_handle.lcd );
                else
                        PLCD_PowerOff( lsc_handle.lcd );
        }
}

void mainForm::removeFingerSlot()
{
        removeFingerLabel->setText( "You can clean a finger." );
}

void mainForm::ledModeSlot(int mode)
{
        if( mode == 0 )
        {
                LSC_SetLedMode( &cl, LSC_LED_AUTO );
        }
        else
        {
                LSC_SetLedMode( &cl, LSC_LED_MANUAL );
        }
}

void mainForm::greenSlot(bool on)
{
        int ret;

        if( ( ret = LSC_SetLed( &cl, LSC_GREEN_LED, on ) ) != LSC_NO_ERROR )
        {
                QMessageBox::critical( this, "Error", "Error in
LSC_SetLed()" + QString::number( ret ) );
                exit( -1 );
        }
}
```

```
void mainForm::redSlot(bool on)
{
        int ret;

        if( ( ret = LSC_SetLed( &cl, LSC_RED_LED, on ) ) != LSC_NO_ERROR )
        {
                QMessageBox::critical( this, "Error", "Error in
LSC_SetLed()" + QString::number( ret ) );
                exit( -1 );
        }
}

/** Slaps segmntation demonstration slot.
 *
 */
void mainForm::segmSlapSlot()
{
        int i, k;
        int l[10];
        int t[10];
        int w[10];
        int h[10];
        int fnum;

        if( ( i = LSCslapSegmAll( &cl, lsc_handle.finger_image,
lsc_handle.finger_width, lsc_handle.finger_height, &fnum, l,
                        t, w, h ) ) != LSC_NO_ERROR )
        {
                QMessageBox::critical( this, "Error", "Error in
LSCslapSegmAll() " + QString::number( i ) );
                return;
        }

        for( k = 0; k < fnum; k++ )
        {
                memset( lsc_handle.finger_image + t[k] *
lsc_handle.finger_width + l[k], 0, w[k] );
                memset( lsc_handle.finger_image + ( t[k] + h[k] ) *
lsc_handle.finger_width + l[k], 0, w[k] );
                for( i = 0; i < h[k]; i++ )
                {
                        lsc_handle.finger_image[( t[k] + i ) *
lsc_handle.finger_width + l[k]] = 0;
                        lsc_handle.finger_image[( t[k] + i ) *
lsc_handle.finger_width + l[k] + w[k]] = 0;
                }
        }

}

void mainForm::sendResult()
{
        FILE* fd;
```

```
        unsigned char* buffer;
        int length;

        fd = fopen( "result.jpg", "r+b" );
        if( fd )
        {
                if( fseek( fd, 0, SEEK_END ) != 0 )
                {
                        fclose( fd );
                        return;
                }

                length = ftell( fd );

                if( fseek( fd, 0, SEEK_SET ) != 0 )
                {
                        fclose( fd );
                        return;
                }

                buffer = (unsigned char*) malloc( length );
                if( buffer == NULL )
                {
                        QMessageBox::critical( this, "Error", "Out of
memory!" );
                        fclose( fd );
                        return;
                }

                if( fread( buffer, length, 1, fd ) != 1 )
                {
                        QMessageBox::critical( this, "Error", "Error read
file!" );
                        fclose( fd );
                        return;
                }

                fclose( fd );

                LSC_SendJpeg2Device( &cl, buffer, length );

                free( buffer );
        }
}

void mainForm::selecScannerSlot()
{
        selectDialog dlg;
        int ret;
        int i;
        LSC_SCANNERS_LIST* list = NULL;
```

```
        ret = LSC_GetScannersList( &list );
        if( ret <= 0 )
        {
                QMessageBox::critical( this, "Error", "There are no
scanners!" );
                LSC_FreeScannersList( list );
                exit( -1 );
        }

        if( ret == 1 )
        {
                this->device = list[0].device;
                LSC_FreeScannersList( list );
                return;
        }

        for( i = 0; i < ret; i++ )
        {
                switch( list[i].device )
                {
#ifdef _WIN32
                        case DS7:
                        {
                                dlg.listBox->insertItem( "DS7", i );
                                break;
                        }

                        case DS9:
                        {
                                dlg.listBox->insertItem( "DS9", i );
                                break;
                        }
                        case FX2000:
                        {
                                dlg.listBox->insertItem( "FX2000", i );
                                break;
                        }

                        case DS30NM:
                        {
                                dlg.listBox->insertItem( "DS30NM", i );
                                break;
                        }

                        case DS31:
                        {
                                dlg.listBox->insertItem( "DS31", i );
                                break;
                        }

                        case FX2001:
                        {
```

```
                        dlg.listBox->insertItem( "FX2001", i );
                        break;
                }

                case DS16:
                {
                        dlg.listBox->insertItem( "DS16", i );
                        break;
                }

                case DS24:
                {
                        dlg.listBox->insertItem( "DS24", i );
                        break;
                }

                case CM_VF:
                {
                        dlg.listBox->insertItem( "CrossMatch
Verifier USB", i );
                        break;
                }

                case FDS7:
                {
                        dlg.listBox->insertItem( "DS7 FireWare", i
);
                        break;
                }

                case CM_VMW:
                {
                        dlg.listBox->insertItem( "CrossMatch
Verifier WiFi", i );
                        break;
                }
                case CM_1000P:
                {
                        dlg.listBox->insertItem( "CrossMatch 1000
Pap", i );
                        break;
                }
                case CM_1000:
                {
                        dlg.listBox->insertItem( "CrossMatch 1000",
i );
                        break;
                }
                case IDENTIX:
                {
                        dlg.listBox->insertItem( "Identix", i );
                        break;
```

```
                        }
#endif
                        case DS27:
                        {
                                dlg.listBox->insertItem( "DS27", i );
                                break;
                        }

                        case DS21:
                        {
                                dlg.listBox->insertItem( "DS21, DS22", i );
                                break;
                        }

                        case DS21N:
                        {
                                dlg.listBox->insertItem( "DS21N", i );
                                break;
                        }

                        case DS30:
                        {
                                dlg.listBox->insertItem( "DS30", i );
                                break;
                        }

                        case DS30N:
                        {
                                dlg.listBox->insertItem( "DS30N", i );
                                break;
                        }

                        case DS14:
                        {
                                dlg.listBox->insertItem( "DS14", i );
                                break;
                        }

                        case DS40:
                        {
                                dlg.listBox->insertItem( "DS40", i );
                                break;
                        }

                        case DS45:
                        {
                                dlg.listBox->insertItem( "DS45", i );
                                break;
                        }

                        case DS22N:
```

```
                        {
                                dlg.listBox->insertItem( "DS22N", i );
                                break;
                        }
                        case DS21S:
                        {
                                dlg.listBox->insertItem( "DS21S", i );
                                break;
                        }
                        default:
                        {
                                break;
                        }
                }
        }

        dlg.listBox->setCurrentItem( 0 );
        if( dlg.exec() != QDialog::Accepted )
        {
                exit( 0 );
        }

        device = list[dlg.listBox->currentItem()].device;
        LSC_FreeScannersList( list );
}

/**Compress image using WSQ algorithm and save to file.*/
void mainForm::compressSlot()
{
        unsigned char* wsq_rec = NULL;
        int wsq_len = 0;
        QString _f;
        FILE* fd;
        QMutexLocker _lock( lsc_handle.image_lock );

        if( lsc_handle.finger_image != NULL )
        {
                if( LSC_CompressWsqPpln( &cl, lsc_handle.finger_image,
lsc_handle.finger_width, lsc_handle.finger_height, 8, 1,
                                compressionBox->value(), &wsq_rec, &wsq_len
) != 0 )
                {
                        QMessageBox::critical( this, "Error", "Error in WSQ
compression!" );
                }
                else
                {
                        _f = QFileDialog::getSaveFileName( "image.wsq",
"*.wsq", this );
                        if( !_f.isEmpty() )
                        {
                                fd = fopen( _f.ascii(), "w+b" );
```

```
                                if( fd )
                                {
                                        fwrite( wsq_rec, wsq_len, 1, fd );
                                        fclose( fd );
                                }
                        }
                        LSC_FreeWsqPpln( &cl, wsq_rec );
                }
        }
}

void mainForm::decompressSlot()
{
        unsigned char* wsq_rec = NULL;
        unsigned char* image = NULL;
        int wsq_len = 0;
        QString _f;
        FILE* fd;

        _f = QFileDialog::getOpenFileName( "image.wsq", "*.wsq", this );
        if( _f.isEmpty() )
                return;

        fd = fopen( _f.ascii(), "rb" );
        if( !fd )
        {
                QMessageBox::critical( this, "Error", "Error to open file "
+ _f );
                return;
        }

        if( fseek( fd, 0, SEEK_END ) != 0 )
        {
                QMessageBox::critical( this, "Error", "Error to read file "
+ _f );
                fclose( fd );
                return;
        }

        wsq_len = ftell( fd );
        if( wsq_len <= 0 )
        {
                QMessageBox::critical( this, "Error", "Error to read file "
+ _f );
                fclose( fd );
                return;
        }

        if( fseek( fd, 0, SEEK_SET ) != 0 )
        {
                QMessageBox::critical( this, "Error", "Error to read file "
+ _f );
```

```
                fclose( fd );
                return;
        }

        wsq_rec = (unsigned char*) malloc( wsq_len );
        if( !wsq_rec )
        {
                QMessageBox::critical( this, "Error", "Out of memory!" );
                fclose( fd );
                return;
        }

        if( fread( wsq_rec, wsq_len, 1, fd ) != 1 )
        {
                QMessageBox::critical( this, "Error", "Error to read file "
+ _f );
                fclose( fd );
                free( wsq_rec );
                return;
        }
        fclose( fd );

        QMutexLocker _lock( lsc_handle.image_lock );
        if( lsc_handle.finger_image != NULL )
        {
                free( lsc_handle.finger_image );
                lsc_handle.finger_image = NULL;
        }

        if( LSC_DecompressWsqPpln( &cl, wsq_rec, wsq_len, &image,
&lsc_handle.finger_width, &lsc_handle.finger_height, 8,
                        1 ) != 0 )
        {
                QMessageBox::critical( this, "Error", "Error in WSQ
decompression!" );
                free( wsq_rec );
                return;
        }

        free( wsq_rec );
        lsc_handle.finger_image = (unsigned char*) malloc(
lsc_handle.finger_height * lsc_handle.finger_width );
        if( !lsc_handle.finger_image )
        {
                QMessageBox::critical( this, "Error", "Out of memory!" );
                return;
        }

        memcpy( lsc_handle.finger_image, image, lsc_handle.finger_height *
lsc_handle.finger_width );

        LSC_FreeWsqPpln( &cl, image );
```

```
        /**Draw the image*/
        takeImageSlot();
}

void mainForm::saveSlot()
{
        if( lsc_handle.finger_image == NULL )
        {
                QMessageBox::critical( this, "Warning", "There is no image
captured!" );
                return;
        }

        QString _f = QFileDialog::getSaveFileName( this, QString( "Save
Image" ), QString( "image.tif" ),
                        QString( "*.tif *.png *.bmp" ) );
        if( _f.isEmpty() )
        {
                return;
        }

        QImage _image( lsc_handle.finger_width, lsc_handle.finger_height,
QImage::Format_Indexed8 );
        int i;

        _image.setNumColors( 256 );
        for( i = 0; i < 256; i++ )
        {
                _image.setColor( i, QColor( i, i, i ).rgb() );
        }

        for( i = 0; i < lsc_handle.finger_height; i++ )
        {
                memcpy( _image.scanLine( i ), lsc_handle.finger_image + i *
lsc_handle.finger_width, lsc_handle.finger_width );
        }

        _image.save( _f );
}

/*Check fingerprint quality*/
void mainForm::checkQualitySlot()
{
        int _quality = 0;
        LSC_ERROR _error;

        if( lsc_handle.finger_image == NULL )
        {
                QMessageBox::warning( this, tr( "Warning" ), tr( "You should
captuer any fingerprint image before!" ) );
                return;
```

```
        }

        QApplication::setOverrideCursor( Qt::WaitCursor );
        qApp->processEvents();

        _error = LSC_CheckQuality( &cl, lsc_handle.finger_image,
lsc_handle.finger_width, lsc_handle.finger_height,
                        &_quality );
        QApplication::restoreOverrideCursor();
        if( _error != LSC_NO_ERROR )
        {
                QApplication::restoreOverrideCursor();
                QMessageBox::critical( this, tr( "Error" ),
                                tr( "Error inLSC_CheckEncode()!\nCode " ) +
QString::number( _error ) );
                return;
        }

        QApplication::restoreOverrideCursor();

        QMessageBox::information( this, tr( "Information" ), tr( "Matching
quality " ) + QString::number( _quality ) );
}

/**Load license file*/
void mainForm::loadLicenseSlot()
{
        QString _f = QFileDialog::getOpenFileName( "lssdk.lic", "*.lic",
this );
        if( _f.isEmpty() )
                return;

        LSC_ERROR _err = LSC_SetLicense( &cl, _f.ascii() );
        if( _err != LSC_NO_ERROR )
        {
                QMessageBox::critical( this, tr( "Error" ),
                                tr( "Error in LSC_SetLicense()!\nCode " ) +
QString::number( _err ) );
        }
        else
        {
                compressButton->setEnabled( true );
                decompressButton->setEnabled( true );
                checkEncodeButton->setEnabled( true );
                checkQualityButton->setEnabled( true );
                compressionBox->setEnabled( true );
        }
}

void mainForm::checkEncodeSlot()
{
        BOOL _good = FALSE;
```

```
        LSC_ERROR _error;

        if( lsc_handle.finger_image == NULL )
        {
                QMessageBox::warning( this, tr( "Warning" ), tr( "You should
captuer any fingerprint image before!" ) );
                return;
        }

        QApplication::setOverrideCursor( Qt::WaitCursor );
        qApp->processEvents();

        _error = LSC_CheckEncode( &cl, lsc_handle.finger_image,
lsc_handle.finger_width, lsc_handle.finger_height, &_good );
        if( _error != LSC_NO_ERROR )
        {
                QApplication::restoreOverrideCursor();
                QMessageBox::critical( this, tr( "Error" ),
                                tr( "Error inLSC_CheckEncode()!\nCode " ) +
QString::number( _error ) );
                return;
        }

        QApplication::restoreOverrideCursor();

        if( _good )
        {
                QMessageBox::information( this, tr( "Information" ), tr(
"Fingerprint image has a good quality for encode" ) );
        }
        else
        {
                QMessageBox::warning( this, tr( "Warning" ),
                                tr( "Fingerprint image has a poor quality
and not suitable for encode." ) );
        }
}
```

mainform.h

Пример использования библиотеки, главная форма.

```
/**
 * \file mainform.h
 * \brief
 * \date 20 июня 2014 г.
 * \author vav
 */
#ifndef MAINFORM_H_
#define MAINFORM_H_

#include <lsclient.h>
```

```
#include "ui_mainform.h"

/**An application main window class*/
class mainForm: public QDialog, private Ui::mainForm
{
                /**QT object*/
                Q_OBJECT

        public:
                /**A constructor
                 *
                 * @param _number - a number of scanner on the bus,
beginning from 0.
                 * @param parent - a parent
                 * @param f - flags
                 */
                mainForm( int _number = 0, QWidget *parent = 0,
Qt::WindowFlags f = 0 );

                /**A destructor*/
                ~mainForm();

                protected slots:

                void init();
                void destroy();
                void initSlot();
                void shutDownSlot();
                void startSlot();
                void stopSlot();
                void fingerSlot( int finger );
                void clearSlot();
                void exitSlot();
                void drawPreviewSlot();
                void clearPreviewSlot();
                void initDoneSlot();
                void processButtonSlot();
                void takeImageSlot();
                void stateChangedSlot();
                void lcdLightSlot( int light );
                void lcdPowerSlot( bool power );
                void removeFingerSlot();
                void ledModeSlot( int mode );
                void greenSlot( bool on );
                void redSlot( bool on );
                void segmSlapSlot();
                void sendResult();
                void selecScannerSlot();
                void compressSlot();
                void decompressSlot();
                void saveSlot();
```

```
                void checkEncodeSlot();
                void checkQualitySlot();
                void loadLicenseSlot();

        protected:
                void customEvent( QEvent * e );

            messageForm* message;
            LSClient cl;
            previewForm* preview;
            LSC_HANDLE lsc_handle;
            unsigned char* lcd_lut;
            LSC_DEVICE device;

            /**Scanner supports fake detection*/
            bool fake_supported;

            bool lssdk_started;

            /**Number of scanner of the bus, initialized from the
constructor.*/
            int mNumber;
};

#endif /* MAINFORM_H_ */
```

messageform.cpp

Класс для вывода различных сообщений.

```
#include "messageform.h"

/*
 *  Constructs a messageForm as a child of 'parent', with the
 *  name 'name' and widget flags set to 'f'.
 *
 *  The dialog will by default be modeless, unless you set 'modal' to
 *  true to construct a modal dialog.
 */
messageForm::messageForm(QWidget* parent, const char* name, bool modal,
Qt::WindowFlags fl)
    : QDialog(parent, name, modal, fl)
{
    setupUi(this);

}

/*
 *  Destroys the object and frees any allocated resources
 */
messageForm::~messageForm()
{
```

```
    // no need to delete child widgets, Qt does it all for us
}

/*
 *  Sets the strings of the subwidgets using the current
 *  language.
 */
void messageForm::languageChange()
{
    retranslateUi(this);
}
```

messageform.h

Класс для вывода различных сообщений.

```
#ifndef MESSAGEFORM_H
#define MESSAGEFORM_H

#include <ui_messageform.h>


class messageForm : public QDialog, public Ui::messageForm
{
    Q_OBJECT

public:
    messageForm(QWidget* parent = 0, const char* name = 0, bool modal =
false, Qt::WindowFlags fl = 0);
    ~messageForm();

protected slots:
    virtual void languageChange();

};

#endif // MESSAGEFORM_H
```

previewform.cpp

Класс для показа предварительного изображения во время сканирования.

```
#include "previewform.h"

#include "imgview.h"

/*
 *  Constructs a previewForm as a child of 'parent', with the
 *  name 'name' and widget flags set to 'f'.
 *
 *  The dialog will by default be modeless, unless you set 'modal' to
 *  true to construct a modal dialog.
```

```
 */
previewForm::previewForm(QWidget* parent, const char* name, bool modal,
Qt::WindowFlags fl)
    : QDialog(parent, name, modal, fl)
{
    setupUi(this);
}

/*
 *  Destroys the object and frees any allocated resources
 */
previewForm::~previewForm()
{
    // no need to delete child widgets, Qt does it all for us
}

/*
 *  Sets the strings of the subwidgets using the current
 *  language.
 */
void previewForm::languageChange()
{
    retranslateUi(this);
}
```

previewform.h

Класс для показа предварительного изображения во время сканирования.

```
#ifndef PREVIEWFORM_H
#define PREVIEWFORM_H

#include <ui_previewform.h>

class previewForm : public QDialog, public Ui::previewForm
{
    Q_OBJECT

public:
    previewForm(QWidget* parent = 0, const char* name = 0, bool modal =
false, Qt::WindowFlags fl = 0);
    ~previewForm();

protected slots:
    virtual void languageChange();

};

#endif // PREVIEWFORM_H
```