

Акционерное общество
«Научно-производственное объединение
Русские Базовые Информационные Технологии»

РУКОВОДЯЩИЕ УКАЗАНИЯ ПО КОНСТРУИРОВАНИЮ
прикладного программного обеспечения
для операционной системы специального назначения
«Astra Linux Special Edition» РУСБ.10015-01
Листов 78

Москва
2018

АННОТАЦИЯ

Настоящие руководящие указания по конструированию (РУК) предназначены для разработчиков прикладных программ на платформе операционной системы специального назначения «Astra Linux Special Edition» (далее по тексту — ОС СН), разработчиков автоматизированных систем (АС) управления в защищенном исполнении и аппаратно-программных средств.

В документе приводится краткий обзор архитектурных особенностей ОС СН и принципов ее версионности. Описаны применение программного интерфейса средств защиты информации (СЗИ) и основы разработки программ для многофункционального защищенного оконного менеджера Fly. Объясняются способы использования интегрированных в ОС СН прикладных программных средств. Настоящие руководящие указания (РУК) распространяются на версию 1.6 ОС СН.

СОДЕРЖАНИЕ

1. Состав дистрибутива ОС СН	6
1.1. Основные компоненты	7
1.2. Рекомендации по использованию средств для разработки ПО	7
1.2.1. Базовые библиотеки и утилиты	7
1.2.2. Графическая система и многофункциональный рабочий стол	8
1.2.3. Средства разработки и отладки	8
1.3. Общее программное обеспечение	8
1.4. Релизы дистрибутива и их версии	9
2. Особенности разработки приложений с графическим интерфейсом, взаимодействующих с рабочим столом Fly	11
2.1. Среда сборки	11
2.2. Группы и имена приложений	11
2.3. Сборка	12
2.4. Указание секции дистрибутива в пакетах	12
2.5. Основные файлы приложения Fly	13
2.5.1. Файл .pro	13
2.5.2. Файл desktop entry	15
2.5.3. Файл перевода	17
2.5.4. Файл для определения иконок приложений	17
2.5.5. Дополнительные параметры	18
2.6. Разработка приложения	18
2.6.1. Общие требования	18
2.6.2. Абсолютные и относительные пути	20
2.6.3. Средства разработки	22
2.6.4. Локализация	23
2.6.5. Диалоги	23
2.6.6. Меню	24
2.6.7. Сохранение настроек	24
2.6.8. Использование тем иконок	25
2.6.9. MIME-типы	26
2.6.10. Использование звуковой темы	26

2.7. Пример простейшего приложения с использованием flybuild	28
2.7.1. Перевод на русский язык	29
2.8. Взаимодействие с рабочим столом	29
2.8.1. Иконки и заголовки окон	29
2.8.2. Системный трей	30
2.8.3. Автозапуск	30
2.8.4. Меню «Пуск» и рабочий стол	30
2.8.5. Корзина	33
2.8.6. Перетаскивание объектов на рабочем столе	34
2.8.7. Подсистема помощи	35
2.8.8. Синхронизация с менеджером окон fly-wm	35
2.8.9. Полноэкранный режим	36
2.8.10. Смена раскладки клавиатуры	36
2.8.11. Дополнительные панели	36
2.9. Плагины для менеджера файлов Fly-fm	37
2.10. Настройки планшетного режима	37
2.11. Приложения и права администратора	37
2.12. Межмандатный буфер обмена	37
2.13. Приложения и мандатная политика	38
3. Программный интерфейс мандатного разграничения доступа	45
3.1. Общие сведения	45
3.2. Сетевое взаимодействие	45
3.2.1. Передача мандатных меток по сети	45
3.2.2. Разработка ПО для обработки информации с различными уровнями конфиденциальности	47
4. Описание использования API системы расширенного аудита	58
5. Разработка ПО для взаимодействия с СУБД PostgreSQL	59
5.1. Мандатное разграничение доступа в СУБД PostgreSQL	59
6. Разработка ПО для WEB-сервера	60
7. Интеграция операционных систем семейства Linux с ОС CH	61
7.1. Интеграция с ЕПП	61
7.1.1. Служба каталогов	62
7.1.2. NSS	63

7.1.3. Аутентификация	64
7.1.4. Домашние каталоги пользователей	65
7.1.5. Администрирование	66
7.2. Интеграция с системой централизованного протоколирования ОС СН	67
8. Рекомендации по способам миграции приложений в среду ОС СН с других платформ	68
8.1. Особенности переноса программ с 32-разрядных платформ на 64-разрядную . .	68
8.2. Настройка среды исполнения 32-разрядных приложений	69
8.3. Перенос программ из среды Windows	69
8.4. Технология компиляции при переносе ПО	70
9. Мобильная сессия	71
9.1. Пакет fly-qml-components	71
9.2. Структура приложения	71
9.3. Клавиша «Назад»	73
9.4. Виртуальная клавиатура	73
9.5. Основные компоненты fly-qml-components	74
9.6. Добавление пользовательского виджета в fly-launcher	74
9.7. Пакет fly-phone-dbus	75
Список литературы	76

1. СОСТАВ ДИСТРИБУТИВА ОС СН

ОС СН относится к семейству дистрибутивов Linux, и соответствует следующим требованиям руководящих документов:

- ГОСТ Р 58256-2018;
- по 3 классу защищенности информации от несанкционированного доступа согласно руководящего документа «Средства вычислительной техники. Защита от несанкционированного доступа к информации. Показатели защищенности от несанкционированного доступа к информации» (Гостехкомиссия России, 1992 г.);
- по 2 уровню контроля отсутствия недеklarированных возможностей согласно руководящего документа «Защита от несанкционированного доступа к информации. Часть 1. Программное обеспечение средств защиты информации. Классификация по уровню контроля отсутствия недеklarированных возможностей» (Гостехкомиссия России, 1999 г.);
- требованиям по безопасности информации ФСТЭК России к операционным системам (приказ ФСТЭК России от 19 августа 2016 года номер 119);
- требованиям ФСБ России по защите информации в автоматизированных системах 1-го класса, и требованиями ФСБ России к программному обеспечению информационных и телекоммуникационных систем специального назначения;
- требованиям безопасности информации Министерства Обороны России (приказ министра обороны 1996 года номер 058);

Это позволяет применять ОС СН при построении АС в защищенном исполнении, обрабатывающих информацию ограниченного распространения, включая государственную тайну до степени секретности «совершенно секретно».

ОС СН разработана на основе открытой операционной системы общего назначения «Astra Linux Common Edition» (далее по тексту – ОС ОН). Особенности архитектуры ОС ОН и ее состав описаны в документе [1].

В состав ОС СН входят встроенные СЗИ. Они обеспечивают мандатное разграничение доступа в дополнение к стандартному для Linux-систем дискреционному разграничению доступа, а также контроль целостности системы и регистрацию событий при работе с файлами и процессами.

Специальные библиотеки СЗИ предоставляют разработчикам программный интерфейс для использования средств защиты в своих программных продуктах. Защита графической среды пользователя в ОС СН обеспечивается защищенным графическим сервером и оптимизированным рабочим столом Fly (читается «флай»), который является единственным вариантом многофункционального графического интерфейса пользователя в составе ОС

СН. Программный интерфейс рабочего стола Fly и методы его использования описаны в разделе 2.

ОС СН унаследовала из ОС ОН основу построения дистрибутива — открытый репозиторий пакетов для ОС Debian Linux с применяемой в нем системой пакетирования. Таким образом, общие принципы разработки и сборки программ для ОС СН, а также создания своего репозитория пакетов соответствуют дистрибутивам Linux семейства Debian. Поэтому, кроме документа [1], разработчикам программ для ОС СН можно использовать общие руководства по Debian Linux (например, такие как [3], [4]).

1.1. Основные компоненты

В целях оптимизации состава в дистрибутив ОС СН не включен ряд дублирующих компонентов, входящих в ОС ОН и решающих сходные целевые задачи (например, различные текстовые редакторы, клиенты электронной почты, средства обработки мультимедиа).

Компоненты ОС СН, с которыми интегрированы встроенные СЗИ:

- базовая платформа:
 - ядро Linux версии 4.15.x;
 - утилиты управления СЗИ;
 - сетевые службы;
 - графический сервер Xorg;
 - графическая система Fly;
 - система печати CUPS;
 - средства организации единого пространства пользователей (ЕПП) система организации домена Astra Linux Directory (ALD).

В качестве оконного менеджера в ОС СН возможно использование только `fly-wm`, адаптированного к требованиям защиты информации;

- общее программное обеспечение (ПО):
 - защищенная СУБД PostgreSQL версии 9.6;
 - HTTP-сервер Apache2;
 - защищенные средства обмена электронными сообщениями Exim4 и Dovecot;

1.2. Рекомендации по использованию средств для разработки ПО

Разработку ПО рекомендуется проводить с использованием перечисленных в данном подразделе базовых компонент, библиотек и средств разработки.

1.2.1. Базовые библиотеки и утилиты

Базовые библиотеки и утилиты:

- 1) ядро Linux версии 4.15.x;

- 2) `glibc`;
- 3) `openldap`;
- 4) `openssl`;
- 5) `bash`.

1.2.2. Графическая система и многофункциональный рабочий стол

Разработку прикладных программ с графическим интерфейсом для ОС СН рекомендуется проводить с использованием следующих библиотек:

- 1) `Qt` — основная прикладная библиотека для создания графического интерфейса;
- 2) `Xorg` — библиотеки графического X-сервера;
- 3) `Fly` — библиотеки графического интерфейса пользователя.

1.2.3. Средства разработки и отладки

Окружение разработки, к которому привыкли разработчики Linux-систем, доступно и в ОС СН. Оно содержится на отдельном DVD-диске со средствами разработки.

Разработку прикладного ПО рекомендуется проводить с использованием этих средств (например, для разработки программ на основе библиотек `Qt` удобно использовать `Qt Creator`).

Фундаментальную инфраструктуру окружения C/C++ составляют инструменты компиляции кода: библиотеки C/C++ (`glibc`), компиляторы, средства сборки, отладчики и специализированные пакеты для разработки программ (`devel`-пакеты). Ниже приведен краткий список этих средств:

- GCC — открытый компилятор, соответствующий стандарту ANSI;
- средства сборки:
 - `binutils` — программы линковки и манипулирования объектными файлами;
 - `make` — средство для автоматизации процесса компиляции;
 - `cmake` — кроссплатформенная система автоматизации сборки программного обеспечения из исходного кода;
- отладчик `gdb` — стандартный отладчик GNU.

1.3. Общее программное обеспечение

В состав ОПО входят программы, которые чаще всего востребованы при построении различных АС: системы управления базами данных (СУБД) реляционного типа, средства работы в сетях, набор офисных программ, система верстки текстов, а также средства работы с мультимедиаданными и графикой.

Для разработчиков АС можно выделить следующие важные компоненты:

- 1) для разработки ПО с использованием СУБД:
 - PostgreSQL версии 9.6

2) для разработки документации:

- L^AT_EX;
- LibreOffice;

3) для работы с мультимедиа:

- мультимедиа плеер VLC;
- фреймворк Phonon;

4) для разработки web-ориентированного ПО:

- Apache2;
- Thunderbird;
- Firefox;
- Exim4;
- Dovecot;
- PHP5.

1.4. Релизы дистрибутива и их версии

Дистрибутивы ОС СН отличаются друг от друга именем релиза и номером версии:

- имя релиза — определяет специфику дистрибутива: его аппаратную платформу и область применения;
- номер версии — 2 или 3 цифры, определяющие версию дистрибутива ОС СН.

В инсталлированном дистрибутиве ОС СН версию его релиза можно посмотреть в текстовом файле в каталоге /etc:

```
/etc/astra_version
```

Формат записи в нем следующий:

```
<EDITION> V.U.Y (NAME)
```

где <EDITION> — редакция дистрибутива Astra Linux (для Special Edition — «SE», для Common Edition — «CE»);

V — первая цифра номера релиза, связанная с именем релиза;

U — версия релиза;

Y — номер обновления в пределах данной версии релиза (если таких обновлений не было, отсутствует);

NAME — имя релиза (привязано к первой цифре номера релиза). Для ОС СН в качестве имен релизов используются названия городов-героев России.

Пример

```
SE 1.6 (Smolensk)
```

Настоящие РУК относятся к релизу «Смоленск» ОС СН, предназначенному для функционирования на средствах вычислительной техники с процессорной архитектурой

x86-64.

2. ОСОБЕННОСТИ РАЗРАБОТКИ ПРИЛОЖЕНИЙ С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ, ВЗАИМОДЕЙСТВУЮЩИХ С РАБОЧИМ СТОЛОМ FLY

В данном разделе приведены правила для разработчиков приложений, включаемых в состав рабочего стола Fly или взаимодействующих с ним.

Реализация данных правил поможет добиться:

- унификации сборки и установки приложений;
- легкости модификации и сопровождения;
- единообразия внешнего вида, внутренней структуры и поведения.

От разработчика требуется знание языка программирования C++ и библиотеки графического интерфейса Qt 5. Настоятельно рекомендуется изучить, например, следующие руководства по работе с этой библиотекой:[16]–[21]. Также предполагается знание основных стандартов от `freedesktop.org`.

2.1. Среда сборки

Все Fly-приложения должны не только единообразно выглядеть и взаимодействовать с пользователем, но и единообразно собираться и устанавливаться. В этой связи существует ряд требований.

2.2. Группы и имена приложений

Все приложения, кроме особых (например, менеджер окон), создаются с использованием графических библиотек Qt 5.

Создание приложения для Fly начинается с определения его назначения и места в составе рабочего стола.

Основные группы приложений Fly:

- 1) обычные программы;
- 2) системные программы, т. е. программы, изменяющие конфигурационные файлы ОС;
- 3) библиотеки;
- 4) ключевые или особые программы (графический вход, менеджер окон, менеджер файлов и т. п.).

Имена всех приложений формируются в соответствии с их назначением по единому принципу: `fly[-admin]-имя`.

Все приложения имеют префикс «fly-». Приложения для настройки системы имеют дополнительный префикс «admin-». Имя приложения должно отражать его назначение, быть осмысленным. С учетом имени приложения образуются соответствующие файлы и каталоги, например:

- исполняемый файл (<имя_приложения>);
- файл перевода (<имя_приложения>_ru.ts);
- каталог для данных (/usr/share/fly/data/<имя_приложения>);
- desktop entry-файл (<имя_приложения>.desktop).

Таким образом имя приложения — основа идентификации приложения в системе.

2.3. Сборка

Оформление приложений в виде пакетов регулируется правилами сборки пакетов для ОС. Пакет flybuild предоставляет файлы, автоматизирующие некоторые задачи конфигурирования и установки. Для deb-пакетов он предоставляет файлы /usr/share/flybuild/fly.mk и /usr/share/flybuild/fly_vars.mk, которые предназначены для включения в файл debian/rules. Типичный файл debian/rules выглядит следующим образом:

```
#!/usr/bin/make -f

include /usr/share/flybuild/fly.mk
include /usr/share/cdb/1/class/qmake.mk
```

Также, flybuild содержит вспомогательные файлы для системы сборки qmake: /usr/lib/x86_64-linux-gnu/qt5/mkspecs/features/fly.prf и /usr/lib/x86_64-linux-gnu/qt5/mkspecs/features/fly_vars.prf. fly_vars.prf задает пути, используемые всеми приложениями при установке, fly.prf — основные процедуры и цели для сборки и установки приложений. Обычно их не нужно подключать каким-то особым образом, достаточно добавить строку "CONFIG += fly" в .pro-файл. В этом случае, можно считать, что fly.prf будет добавлен в конец файла проекта. Но если возникает необходимость использовать переменные, заданные в fly_vars.prf, то нужно подключить этот файл командой load(fly_vars) в нужном месте.

При каждой, даже формальной, пересборке пакета для его включения в очередной релиз дистрибутива следует изменять последнюю цифру, т.е. 2.0.1, 2.0.2 и т.д. При изменении функционала программ версия меняется по усмотрению разработчика.

Рекомендуется так же отделять компоненты для разработки (заголовочные файлы, компоненты для Qt Designer) от самих разделяемых библиотек в виде name-dev-пакетов.

2.4. Указание секции дистрибутива в пакетах

В пакетах прикладных программ для Fly следует указывать non-free/fly. Если же программа была основана на opensource-проекте, то следует указывать секцию fly.

2.5. Основные файлы приложения Fly

Получив имя, приложение должно предоставить некоторые файлы:

- pro-файл для сборки;
- desktop entry-файл;
- файлы с переводом;
- возможно также специфичные файлы (иконки и т. п.).

2.5.1. Файл .pro

Требования к pro-файлам приложений:

- имя образуется как имя программы.pro;
- краткость и простота, не избыточность;
- единообразии состава и структуры;
- отсутствие каких-либо абсолютных путей;
- обязательное задание параметра конфигурации `CONFIG += fly`.

Выражение `CONFIG += fly` вызывает подключение файла `fly.prf`, который централизует важнейшие особенности сборки и установки приложений, освобождая программистов от необходимости их повторного определения в своих pro-файлах. `fly.prf` использует переменные, заданные в `fly_vars.prf`, для определения путей установки файлов приложения. Ни один pro-файл ни одной программы не должен без необходимости переопределять переменные, заданные в `fly_vars.prf`, а, наоборот, должен их использовать. Таким образом, создание pro-файла приложения начинается с изучения `fly_vars.prf` и `fly.prf`. Их можно найти в каталоге `/usr/lib/x86_64-linux-gnu/qt5/mkspecs/features`. Приведем пример правильного pro-файла, например `fly-commi.pro`:

```
TEMPLATE = app
TARGET = fly-commi
```

```
QT += widgets
CONFIG += fly
FLY += core ui
```

```
HEADERS += mylineedit.h
SOURCES += main.cpp mylineedit.cpp
FORMS += commimainform.ui optionsform.ui transferform.ui
TRANSLATIONS = fly-commi_ru.ts
```

или `fly-admin-cron.pro`:

```
TEMPLATE = app
TARGET = fly-admin-cron
```

```

QT += widgets
CONFIG += fly
FLY += core ui ui_extra

SOURCES += main.cpp cconfig.cpp
HEADERS += cconfig.h ccrondata.h ...
FORMS += fly-cron.ui
TRANSLATIONS = fly-admin-cron_ru.ts

```

Примерно в такой последовательности и составе должны следовать строки. Обратите внимание на краткость этих pro-файлов и на переменную FLY, которая позволяет подключить такие библиотеки, как: libflycore (FLY += core), libflyui (FLY += ui), libflyuiextra (FLY += ui_extra), необходимые большинству приложений Fly.

Любые отклонения в pro-файлах должны быть обоснованными. Среди часто встречающихся отметим такие:

- необходимость определения наличия каких-либо файлов при сборке;
- необходимость установки дополнительных данных программы.

В первом случае можно использовать конструкции вида:

```

exists( /usr/include/kudzu/isapnp.h ) {
SUBDIRS += fly-admin-snddetect
}

```

Во втором случае допустимо:

```

load(fly_vars)
pics.path = $$${FLY_INSTALL_DATA}/fly-parashoot/icons
pics.files = pics/*
sounds.path = $$${FLY_INSTALL_DATA}/fly-parashoot/sounds
sounds.files = sounds/*
INSTALLS += pics sounds

```

— для файлов, устанавливаемых в собственный каталог приложения (здесь — \$\$\${FLY_INSTALL_DATA}/fly-parashoot), или:

```

pam.path = /etc/pam.d
pam.files = ../pam/fly-dm

```

— для файлов, устанавливаемых в разделяемые каталоги (здесь — /etc/pam.d).

Еще пример фрагмента pro-файла одной из библиотек:

```

...
load(fly_vars)

headersfly.path = $$FLY_INSTALL_HEADERS
headersfly.files = flyui.h flytranslator.h flyabout.h \

```

```
flyhelp.h flyhelpmenu.h flycmdlineargs.h
```

```
uiheadersfly.path = $$FLY_INSTALL_HEADERS
uiheadersfly.extra = $$QMAKE_COPY_FILE $$UI_DIR/ui_flyabout.h \
$(INSTALL_ROOT)/$$uiheadersfly.path}
```

```
INSTALLS += headersfly uiheadersfly
```

```
...
```

Разработчикам необходимо обратить внимание, что Qt как кросс-платформенное средство предоставляет ряд макросов для часто используемых команд, например `QMAKE_COPY_FILE`, `QMAKE_MKDIR` и т.д. (см. файл `$QTDIR/mkspecs/default/qmake.conf`).

Если возникает необходимость удалять каталоги при выполнении `make clean` или `make distclean`, то можно добавить их к переменным `FLY_CLEAN_DIRS` и `FLY_DISTCLEAN_DIRS` соответственно.

2.5.2. Файл `desktop entry`

Почти каждое fly-приложение должно сопровождаться файлом `desktop entry` [6] с именем, формируемым по правилу:

```
<имя_приложения>.desktop
```

Данный файл содержит информацию о том, к какому классу приложений относится поставляемая программа, имя иконки для представления на рабочем столе, способ запуска данной программы и служебную информацию для интеграции приложения в рабочий стол.

Имя иконки приложения должно быть указано в поле `Icon` файла `desktop entry`. В виде исключения в этом поле допускается указывать полный путь к иконке приложения.

Приложения, открывающие специфичные типы файлов, должны объявлять соответствующие MIME-типы данных файлов в поле `MimeType=` своих *.desktop-файлов.

Примеры:

```
1. [Desktop Entry]
```

```
Exec=fly-admin-date
```

```
Icon=date
```

```
Type=Application
```

```
Name=Configure clock
```

```
Name[ru]=Дата и время
```

```
Comment=Configure date & time
```

```
Comment[ru]=Настройка системного времени и даты
```

```
Categories=Settings;SystemSetup
```

```
X-Fly-AdminOnly=true
```

```

2. [Desktop Entry]
Type=Application
Exec=fly-archiver open %f
Icon=package
Name=Archivator
Name[ru]=Архиватор
Comment=Qt RAR, TAR, ZIP, RPM, Gzip, Bzip2 GUI
Comment[ru]=Qt RAR, TAR, ZIP, RPM, Gzip, Bzip2 интерфейс
Categories=Application;Office;
Actions=Open;Decompress;Compress
DocPath=fly-arch/index.html
MimeType=application/x-rar;application/x-bzip-compressed-tar;
application/x-bzip;application/x-compressed-tar;
application/x-tar;application/x-gzip;application/zip;
[Desktop Action Compress]
Name=Add file to archive...
Name[ru]=Добавить файл в архив...
Exec=fly-arch compress %F
[Desktop Action Open]
Name=Open with fly-arch...
Name[ru]=Открыть с помощью fly-arch...
Exec=fly-arch open %f
[Desktop Action Decompress]
Name=Extract from archive...
Name[ru]=Извлечь из архива...
Exec=fly-arch decompress %f

```

В принципе, все поля файла имеют очевидное назначение, подробно описанное в [6]. Слово `Settings` в поле `Categories` говорит о принадлежности программы к средствам администрирования. Рабочий стол `Fly` имеет в своем составе специальную программу «Панель управления», предназначенную для централизованного вызова утилит управления системой. «Панель управления» находит нужные утилиты именно по полю `Categories`.

Категорию можно назначить не только файлу, но и целому каталогу путем заполнения поля `Categories` в файле `.directory` (тип `Directory`), находящемся в этом каталоге. Значение поля `'X-Fly-AdminOnly'` определяет, нужно ли показывать данную программу только администратору, или всем пользователям системы.

Файлы `desktop entry` как правило должны иметь `Type=Application`, но для других случаев возможно использование других типов, разрешенных стандартом [6]:

Application, Link, Directory.

Слово Fly следует использовать в поле Name ярлыков приложений (особенно административных), только если настраивается что-то специфичное для самого Fly, например, «горячие» клавиши или меню самого рабочего стола Fly, т. е. то, что вне Fly неприменимо. Если настраиваются какие-то другие общие службы, то слово Fly не следует использовать. Его также можно использовать, чтобы отличать приложения, разработанные специально для Fly, от аналогичных приложений для других рабочих столов, например текстовый редактор может называться «Текстовый редактор Fly», если предполагается присутствие в системе и других текстовых редакторов. В названиях программ, видимых пользователю (в меню или в панели управления), не следует повторять одни и те же слова, например, «Менеджер камер Fly», «Менеджер сканеров Fly», лучше так: «Камеры», «Сканеры». Чем короче названия, тем лучше, например, не «Редактор переменных окружения», а «Переменные окружения».

2.5.3. Файл перевода

Все приложения должны быть русифицированы с использованием стандартных средств Qt.

Не следует статично вбивать русские названия элементов графического интерфейса непосредственно в код программы или в формы, генерируемые с помощью Qt Designer. Это многократно усложняет локализацию приложений для других стран.

Изначально Fly-приложения должны содержать только английские варианты названий элементов своих интерфейсов. Файл с переводами (ts-файл) должен формироваться с помощью программ `lupdate` и `linguist`, входящих в состав дистрибутива ОС. При установке ts-файл будет сконвертирован в бинарный qm-файл, подходящий для загрузки qt-приложением. Подключение соответствующего текущей локали qm-файла производится в функции `flyInit()` библиотеки `libflyui`, вызываемой в `main`-функции приложения. Для этого необходимо, чтобы файл перевода был соответствующим образом расположен (обеспечивается скриптами установки `fly.prf`) и имел имя, сформированное по шаблону `имя приложения_локаль.qm`, например `fly-admin-mouse_ru.qm`.

2.5.4. Файл для определения иконок приложений

Приложение может установить свою специфичную иконку. Для этого она должна быть в `svg`-формате, лежать в том же каталоге, что и `pro`-файл приложения и название ее файла должно быть производным от переменной `TARGET`. Например, если `TARGET = fly-admin-cron`, то файл иконки должен называться `fly-admin-cron.svg` или `fly-admin-cron.svgz`. Если необходимо установить несколько `svg`-иконок, то нужно в переменной `FLY_INSTALL_SVG_ICONS_FROM` задать каталог относительно `pro`-файла, в котором они лежат. При этом следует учесть, что иконка, про которую говорилось

в предыдущем способе установки, будет проигнорирована. Названия файлов этих иконок должны соответствовать шаблону <контекст>-<имя_иконки>.svg, где <контекст> — название каталога, представляющего контекст иконки в теме. При сборке приложения они будут сконвертированы в png-файлы и скопированы в поддеревья сборки типа ./debian/tmp/usr/share/icons/hicolor/16x16/apps и т.д., т.е. в подкаталоги <контекст> темы иконок hicolor — родительской темы для всех других тем иконок. Приложение должно обеспечить попадание указанных поддеревьев usr/share/icons/hicolor в свой пакет. Все это позволит приложению в процессе работы операционной системы, даже при смене текущей темы, всегда находить свои иконки, т.к. тема hicolor есть всегда.

2.5.5. Дополнительные параметры

Полезные переменные, задаваемые в fly_vars.prf:

- FLY_INSTALL_PREFIX — корневой каталог для установки (обычно /usr);
- FLY_INSTALL_BINS — каталог, в который попадут исполняемые файлы;
- FLY_INSTALL_LIBS — каталог, в который попадут библиотеки;
- FLY_INSTALL_HEADERS — каталог, в который попадут заголовочные файлы;
- FLY_INSTALL_DATA — корневой каталог для файлов данных;

Их можно определить извне, передав qmake в командной строке:

```
qmake FLY_INSTALL_PREFIX=/usr/local
```

При использовании cdbс (рекомендуемый способ сборки пакетов) для этого достаточно добавить их к DEB_QMAKE_ARGS:

```
DEB_QMAKE_ARGS += FLY_INSTALL_PREFIX=/usr/local
```

Но делать это не рекомендуется. Если возникает потребность использовать эти переменные в debian/rules, то можно включить в него файл /usr/share/flybuild/fly_vars.mk.

Пример файла debian/rules:

```
#!/usr/bin/make -f
```

```
include /usr/share/flybuild.mk
```

```
include /usr/share/cdbс/1/class/qmake.mk
```

2.6. Разработка приложения

2.6.1. Общие требования

Fly-приложение должно быть интуитивно понятным, ориентированным на неподготовленного пользователя.

Пользователю обязательно должны задаваться вопросы о сохранении изменений, вступлении изменений в силу, перезаписи существующих настроек или файлов, перезапуске системы и т.п. При этом всегда должна даваться возможность осуществить отмену действий

такого рода.

Главной функцией, которую следует вызвать сразу после создания экземпляра `QApplication` является `flyInit()`.

Ее аргументы — версия приложения и описание назначения программы, понятное неподготовленному пользователю, например, в стиле «Данная программа fly-* предназначена для ...». Имя приложения в `flyInit()` извлекается автоматически и используется при поиске переводов, при работе `QSettings` и т.п. Указанные версия и описание используются при создании диалога «О программе...». Пример, настоятельно рекомендуемый к использованию:

```
flyInit("2.0.0", QT_TRANSLATE_NOOP("Fly", "Fly hotkeys editor"));
```

где 2.0.0 — строка с номером версии для диалога «О программе»;

Fly — фиксированное слово-контекст, остальное задается разработчиком.

Эта функция позволяет потом использовать класс `FlyHelpMenu` без указания каких-либо деталей, использовать `QSettings` с именами параметров без указания имени программы, т.е. в предельно кратком виде типа «/width», «/height».

Программист может автоматизировать подстановку номера версии в `flyInit()` из версии пакета.

Примеры:

1. В rules добавить:

```
SOURCE_VERSION:=$(shell head -1 debian/changelog | \
    cut -d\ ( -f2 | cut > -d\ ) -f1)
```

2. Там же передать эту переменную:

```
qmake: $(QTDIR)/bin/qmake SOURCE_VERSION=$(SOURCE_VERSION)
```

3. При использовании `cdbs` и `flybuild` вместо предыдущих двух пунктов можно использовать:

```
include /usr/share/flybuild/fly.mk
include /usr/share/cdbs/1/class/qmake.mk
```

4. В pro-файл добавить (только если не используется `CONFIG += fly`):

```
DEFINES += SOURCE_VERSION=\\\\"$$SOURCE_VERSION\\\\"
```

5. Подставить в `flyInit`:

```
flyInit(SOURCE_VERSION, \
    QT_TRANSLATE_NOOP("Fly", "Fly rich text > editor"))
```

Таким образом, разработчикам останется только исправить `rules` (1 или 2 строки) и `pro`-файл (1 строка и то, только если не используется `CONFIG += fly`!). При этом, в самой программе перед вызовом `flyInit()` можно, не полагаясь на `rules`, сделать так:

```
#ifndef SOURCE_VERSION
```

```
#define SOURCE_VERSION "2.0.0"
#endif
...
flyInit(SOURCE_VERSION,
        QT_TRANSLATE_NOOP("Fly", "... short description..."));
```

Любое Qt-приложение может принимать ряд аргументов командной строки, специфичных для Qt и X11. Например, `-geometry` и `-display` (см. описание класса `QApplication`).

Приложение `Fly`, если оно самостоятельно (с помощью класса `QCommandLineParser`) анализирует аргументы своей командной строки, должно без препятствий пропускать упомянутые стандартные Qt/X11-аргументы, не останавливая свою работу с сообщениями типа «неверный аргумент».

Также следует обратить внимание на `QtSingleApplication` (см. `libqtsingleapplication-dev`). Этот класс описан в официальной документации от Trolltech. Для его использования нужно добавить опцию `qtsingleapplication` к переменной `CONFIG` в `pro`-файле. Он позволяет обеспечить запуск одного экземпляра приложения в системе. Однако, бывают ситуации, когда надо запустить несколько экземпляров, но только по одному для каждого дисплея.

Различать экземпляры приложений для разных дисплеев можно с помощью переменной окружения `DISPLAY`, например, так

```
QtSingleApplication app("MySingleInstance"+getenv("DISPLAY"), argc, argv)
```

Однако, не стоит забывать, что стандартный X11-аргумент `-display <displayname>` имеет больший приоритет, чем переменная `DISPLAY`. Поэтому сначала надо проверить его наличие и, если он задан, то использовать именно его.

2.6.2. Абсолютные и относительные пути

При разработке приложений следует избегать использования абсолютных и относительных путей для обращения к файлам данных и файлам иконок. Только в случае, если расположение файла заранее предопределено каким-либо стандартом и не меняется от версии к версии системы (например, конфигурационный файл `lilo.conf` загрузчика `LILO` по умолчанию располагается в каталоге `/etc`), можно статично включить абсолютный путь к данному файлу в код программы.

Использование относительных путей, содержащих конструкции типа «.» и «..», может привести к сбою работы программы, т. к. она может быть запущена из любого места.

Если в приложении требуется осуществить доступ к одному из стандартных каталогов пользователя или системы (например, каталогу рабочего стола, который для каждого пользователя свой), то путь к данному каталогу должен быть определен с помощью ме-

тодов класса `QStandardPaths` (в Qt 5). Для получения путей специфических каталогов, используемых в `Fly`, можно воспользоваться средствами, предоставляемыми библиотекой `libflycore`. В заголовочном файле `flyfileutils.h` определена функция:

```
char* GetFlyDir(FlyDir type)
```

где `type` является перечислением, определяющим требуемый каталог:

```
typedef enum _FlyDir {
WM_DIR, USER_TMP_DIR, USER_BASE_DIR, USER_THEME_DIR,
USER_START_MENU_DIR, USER_CPL_DIR, USER_DESKTOP_DIR,
USER_DOCUMENTS_DIR, USER_TRASH_DIR, FLY_WORK_DIR,
APP_SHARED_DIR, APP_TRANS_DIR, APP_DOCS_DIR,
APP_DOCS_HTML_DIR, WM_SOUNDS_DIR, WM_IMAGES_DIR,
WM_KEYMAPS_DIR, USER_TOOLBAR_DIR, ... USER_CACHE_DIR, ...
} FlyDir
```

где `WM_DIR` — каталог общих настроек (`/usr/share/fly-wm`), на основе которых при первом запуске `fly-wm` создаются начальные индивидуальные настройки для каждого пользователя;

`USER_TMP_DIR` — временный каталог;

`USER_BASE_DIR` — каталог настроек для каждого пользователя (`$HOME/.fly`);

`USER_THEME_DIR` — каталог тем оформления рабочего стола (`$HOME/.fly/theme`);

`USER_START_MENU_DIR` — каталог стартового меню (`$HOME/.fly/startmenu`);

`USER_AUTOSTART_MENU_DIR` — каталог автозапуска (`$HOME/.fly/startmenu` или `$HOME/.config/autostart`);

`USER_DESKTOP_DIR` — каталог ярлыков рабочего стола (`$HOME/Desktop`);

`USER_TRASH_DIR` — каталог корзины (сейчас `$HOME/.local/share/Trash`);

`APP_SHARED_DIR` — каталог данных для приложений (`/usr/share/fly`);

`APP_DOCS_DIR` — каталог документации (`/usr/share/doc/fly`);

`APP_DOCS_HTML_DIR` — каталог html-документации (`/usr/share/doc/fly/html`);

`APP_TRANS_DIR` — каталог файлов переводов (`/usr/share/fly/translations`);

`USER_TOOLBAR_DIR` — каталог ярлыков для панели задач (`$HOME/.fly/toolbar`),

т. е. панели быстрого запуска, расположенной на панели задач справа от кнопки «Пуск»;

`USER_DOCUMENTS_DIR` — каталог для хранения документов пользователя (`$HOME/Documents`), с возможными подкаталогами для файлов с ненулевыми мандатными метками.

Поддерживаются все каталоги, предусмотренные пакетом `xdg-user-dir`, а именно: `USER_VIDEOS_DIR`, `USER_MUSIC_DIR`, `USER_PICTURES_DIR`, `USER_DOWNLOAD_DIR`, `USER_PUBLICSHARE_DIR`, `USER_TEMPLATES_DIR`, но в Qt-приложениях следует использовать класс `QStandardPaths`.

Данная функция возвращает указатель на статическую область памяти, содержащую требуемый абсолютный путь. Так, например, все приложения могут получить доступ к своим файлам переводов из каталога, возвращаемого по вызову `GetFlyDir(APP_TRANS_DIR)`.

Оконный менеджер `fly-wm` может запускать приложения с ненулевыми мандатными уровнями. Некоторые приложения могут не работать в силу того, что они производят запись в файлы с ненулевыми уровнями. Для упорядочивания все приложения, которым требуется временный каталог, должны получить путь к нему из переменной `$TMPDIR`. Гарантируется, что если приложение было запущено с ненулевым уровнем, то `$TMPDIR` для этого приложения выставлена на каталог, позволяющий в него писать/читать. Получить свой временный каталог приложение может с помощью стандартной функции `getenv("TMPDIR")` или более общим вызовом `GetFlyDir(USER_TMP_DIR)` из библиотеки `libflycore`. Аналогичное справедливо и для каталога «Мои документы» – `GetFlyDir(USER_DOCUMENTS_DIR)` и (или) переменной окружения `$USERDOCDIR` и каталога с иконками рабочего стола — `GetFlyDir(USER_DESKTOP_DIR)` и (или) переменной окружения `$DESKTOPDIR` и т. д.

Начиная с версии ОС СН Смоленск 1.6 рекомендуется вместо `$TMPDIR` использовать переменную `$XDG_RUNTIME_DIR`.

Способом установки и получения путей к основным каталогам является пакет `xdg-user-dirs` (см. его описание), кооперация с которым уже реализована в `libflycore`.

2.6.3. Средства разработки

Графический интерфейс должен быть интуитивно понятен пользователю. Диалоги и окна разных приложений должны выглядеть единообразно. Частично это достигается за счет использования графической библиотеки Qt без переопределения параметров по умолчанию, но конечная ответственность лежит на разработчике приложения. Для обеспечения легкости модификации и единообразия при разработке графического интерфейса настоятельно рекомендуется использовать формы Qt Designer.

Для облегчения и унификации разработки предоставляется ряд библиотек, основные из которых `libflycore` и `libflyintegration`. В них содержатся общие для приложений классы (в т. ч. и графические), функции и структуры данных.

Библиотека `libflycore` не зависит от Qt. Ее назначение:

- разбор файлов `desktop entry` [6];
- доступ к темам иконок [7];
- взаимодействие с менеджером окон;
- формирование абсолютных путей к файлам рабочего стола, включая файлы русификации, помощи и т. д.;
- дополнительные задачи, не требующие GUI.

Библиотека `libflyintegration` — библиотека, содержащая общие для всех прило-

жений элементы графического интерфейса, базирующиеся на Qt, такие как меню «Помощь» и диалог «О программе...» и др., а также ряд вспомогательных функций.

Использование указанных библиотек может существенно ускорить разработку приложений и их интеграцию с рабочим столом.

Дополнительно предоставляются библиотеки:

- `libflyuiextra` — для дополнительных элементов графического интерфейса;
- `libflyuinet` — для элементов настройки сети;
- ряд других, включая библиотеки для выполнения файловых операций.

2.6.4. Локализация

Все заголовки и надписи должны быть выполнены на русском языке. Локализация выполняется средствами и в соответствии с рекомендациями Qt (подробнее см. 2.5.3).

2.6.5. Диалоги

Во всех диалогах должен использоваться класс `QDialogButtonBox`, как скрывающий детали перевода и размещения кнопок. Аналогично в диалогах-мастерах должен использоваться набор кнопок по умолчанию. Нестандартный состав кнопок, их расположение и расстояния допускаются только для сложных нестандартных диалогов. Программы, имеющие диалоговый интерфейс, должны поддерживать следующие «горячие» клавиши:

- **<Enter/Return>** — эквивалент нажатия кнопки **[Да]**;
- **<Escape>** — эквивалент нажатия кнопки **[Отмена]**;
- **<F1>** — вызов помощи для диалога (опционально).

Для задания «горячих» клавиш для типичных действий в классе `QKeySequence` существует специальный Enum: `QKeySequence::StandardKey`. Его элементы и надо использовать, например, при создании таких типовых действий, как: копирование, вставка, создание, удаление и т. п.

Отметим пару моментов при использовании диалога-мастера (`QWizard`). Не следует без необходимости переопределять стиль этого диалога по умолчанию (`ClassicStyle`). При наличии соответствующих элементов дизайна их надо использовать стандартным способом `QWizardPage::setPixmap()`. При этом первая и последняя страницы, как правило, должны быть с вводными и заключительными пояснениями и иметь только боковые картинки (`watermark`, слева, не более трети ширины диалога и высотой, равной высоте диалога). Промежуточные страницы, на которых осуществляется основное взаимодействие с пользователем, должны иметь картинку сверху, содержащую, как правило, наряду с картинками (`banner`, `logo`) также и пояснения к текущей странице (`title`, `subtitle`).

2.6.6. Меню

Типичное меню приложения должно иметь примерно следующую структуру: «Файл», «Правка», «...», «Помощь».

Пункты «Файл» и «Помощь» являются обязательными пунктами меню.

Меню «Файл» может состоять из следующих пунктов:

«Новый» или «Создать»

«Открыть...»

...

–разделитель–

«Закрыть»

«Сохранить»

«Сохранить как...»

...

–разделитель–

«Печать...»

–разделитель–

«Выход»

–разделитель–

«Список последних открывавшихся файлов»

Пункт «Выход» должен иметь либо стандартную иконку из текущей темы «application-exit», «Actions», либо не иметь иконки вообще.

ВНИМАНИЕ! Никакая другая иконка не должна использоваться для этих целей.

Меню «Помощь» имеет фиксированную структуру, которая должна использоваться во всех приложениях:

«Содержание F1»

«О программе...»

без каких-либо сепараторов и иконок. Как исключение у «Содержание F1» допускается иконка «help-contents», «Actions». Рекомендуется использовать класс `FlyHelpMenu` (только при условии правильного использования `flyInit()`).

2.6.7. Сохранение настроек

Каждое приложение, завершая свою работу, должно сохранять набор параметров. Как минимум, приложение должно сохранить свои размеры (высоту и ширину, установленные пользователем), пропорции разделителей (splitter) и колонок (если есть), размеры важных диалогов, положение плавающих панелей инструментов (toolbars) и т. п. В зависимости от специфики решаемой задачи, может потребоваться сохранение каких-либо дополнительных параметров. Например, рекомендуется сохранять/восстанавливать такое состояние окна,

как «maximized» и/или «minimized», но лучше все состояние окна в целом — `windowState` (см. `QWidget` в Qt-документации). Единственный параметр, который приложению нельзя сохранять при завершении, а потом восстанавливать при запуске, — позиция окна на экране, исключение — какие-либо специальные диалоги и сообщения, требующие немедленного внимания оператора.

Для сохранения/восстановления настроек следует использовать возможности класса `QSettings`, который берет на себя задачи определения места расположения конфигурационного файла, сохранения параметров в виде определенной структуры и извлечения данных параметров. При этом следует:

- полностью полагаться на библиотеку Qt в вопросе выбора места сохранения;
- в качестве `key` в `QSettings::value/setValue (key, value)` использовать `[/programName[/sectionName]]/parameterName`. Тогда автоматически в нужном месте библиотекой Qt будет создан (открыт) конфигурационный файл с секцией `sectionName` и параметром `parameterName = value`. При этом, если использовалась функция `flyInit()`, то в качестве имени параметра следует использовать только `parameterName`, т.к. префикс будет автоматически сформирован с учетом имени приложения и названия организации.

Начиная с версии ОС СН Смоленск 1.5 рабочий стол Fly имеет несколько режимов работы (обычный рабочий стол, планшетный, мобильный, безопасный), выбираемых оператором в одном из меню графического входа. Соответственно, некоторым программам может понадобиться сохранять разные настройки для разных режимов. Имя режима можно определить по переменной `$DESKTOP_SESSION`, и использовать его для сохранения настроек в разных секциях `QSettings`.

2.6.8. Использование тем иконок

Каждое приложение должно использовать тему иконок для общеупотребительных иконок. Для этого в классе `QIcon` есть статический метод `fromTheme`. Также можно воспользоваться методами класса `FlyIcon`: `fromTheme` и `withEmblems`. В отличие от `QIcon::fromTheme`, `FlyIcon::fromTheme` позволяет задавать эмблемы, которые должны быть наложены на иконку. В большинстве ситуаций использование `QIcon::fromTheme` будет наилучшим выбором. Следует отметить, что объекты `QIcon`, возвращаемые этими вызовами, подгружают изображения иконок определенного размера только по мере необходимости.

Пример

```
int main (int argc, char **argv)
{
    QApplication app(argc, argv);
```

```

...
    app.setWindowIcon(QIcon::fromTheme("accessories-text-editor"));
...
}

```

Следует обратить внимание, что нужно задавать только имя иконки без расширения. При неудачном извлечении иконки из темы можно использовать встроенную иконку. Методы `QIcon::fromTheme` и `FlyIcon::fromTheme` имеют для этого параметр `fallback`. Уникальные иконки, которым нет аналогов в теме, могут быть «вшиты» в приложение. Начиная с версии 1.3 в операционной системе используется новая тема `fly-astra`, которая соответствует как `Icon Naming Specification` [7], так и некоторым особенностям KDE4. Теперь использовать имена иконок, принятые в KDE3, уже нельзя.

Начиная с версии ОС СН Смоленск 1.6 в соответствии с общим «плоским» дизайном появилась «плоская» тема `fly-astra-flat`, наследующая тему `fly-astra`.

Если возникает редкая необходимость получать пути к файлам иконок, то можно воспользоваться классом `FlyIconTheme` из `libflycore`, который позволяет

- открывать существующие темы иконок;
- создавать новые темы;
- редактировать темы;
- сохранять изменения;
- получать полный путь к иконке по ее короткому имени, размеру и контексту;
- получать полный путь к иконке по названию MIME-типа;
- ряд других возможностей.

Класс `FlyIconTheme` полностью реализует стандарт «`Icon Theme Specification`» от X Desktop Group, включая механизмы наследования [7].

Для единообразия внешнего вида можно использовать иконки `media-playback-start` и `media-playback-stop` (контекст `Actions`) для запуска и остановки сервисов и служб. Для перемещения элементов между списками подходят иконки `lleftarrow` и `lrightarrow` (контекст `Actions`). В самих списках не следует бесосновательно отключать режим множественного выделения.

2.6.9. MIME-типы

В библиотеку `libflycore` включена часть, реализующая API для детектирования MIME-типа любого файла (реализация соответствующего стандарта `freedesktop.org` [8]). Qt-приложения должны использовать класс `QMimeDatabase`.

2.6.10. Использование звуковой темы

Начиная с версий 2.1.0 пакетов `fly-wm`, `fly-admin-wm`, `flycore`, `flyui`, `fly-data` в рабочем столе Fly внедрена поддержка спецификаций для звуковых тем и наименований звуков [14].

Каждое приложение, желающее пользоваться звуками из этой спецификации, должно использовать текущую звуковую тему, выбранную пользователем в настройках рабочего стола Fly. Для этого предназначен класс `FlyIconTheme`, позволяющий:

- открывать существующие темы звуков;
- создавать новые темы;
- редактировать темы;
- сохранять изменения;
- получать полный путь к файлу звука по его короткому имени, локали и профилю;
- ряд других возможностей.

Класс практически полностью реализует стандарты `Sound Theme and Naming Specifications` [14] от X Desktop Group (freedesktop.org), включая механизмы наследования.

Из текущей для рабочего стола звуковой темы по имени (обязательно), локали (не обязательно) и профилю (не обязательно, но обычно это «stereo») можно получить полный путь к требуемому звуку, например, следующим образом:

Пример

```
...
#include <fly/flysoundtheme.h>
...
int main (int argc, char **argv)
{
    ...
    char soundPath[PATH_MAX];

    //получить полный путь в soundPath по имени ("dialog-warning")
    //по локали (пусто "" - необязательный аргумент) и
    //профилю ("stereo" - обычно стерео, но необязательный аргумент)

    if ( FlySoundTheme::getSound("dialog-warning","", "stereo",soundPath) );
    else { //не найден
        ...
    }
}
```

При этом класс сам подберет расширение для звука (обычно «.wav»).

При неудачном извлечении звука из темы должен использоваться либо встроенный звук, либо ничего. Уникальные звуки, которым нет аналогов в спецификации [14] должны поставляться вместе с приложением.

Для удобства программиста для некоторых функций flycore есть соответствующие Qt-функции (wrappers) во libflyui, например, QString flySound(...).

Спецификации [14] и их поддержка во flycore/flyui (см. flysoundtheme.h и flyui.h) дают возможность получать путь к звуковому файлу. Его дальнейшее использование (проигрывание) полностью лежит на самом приложении.

2.7. Пример простейшего приложения с использованием flybuild

Рассмотрим пример простейшего приложения fly-demo, выводящего при запуске на экран окно с сообщением о наличии или отсутствии прав привилегированного пользователя.

```
#include <QtSingleApplication>
#include <QMessageBox>

#include <flyintegration.h>
#include <unistd.h>

#ifdef SOURCE_VERSION
#define SOURCE_VERSION "2.0.0"
#endif

int main(int argc, char *argv[])
{
    QtSingleApplication app(argc, argv);

    flyInit(SOURCE_VERSION, QT_TRANSLATE_NOOP("Fly", "Simple sample Fly"));

    if (geteuid() != 0)
        QMessageBox::critical(nullptr, QObject::tr("Unprivileged user"),
            QObject::tr("You run the program as an unprivileged user."
        ));
    else
        QMessageBox::critical(nullptr, QObject::tr("Privileged user"),
            QObject::tr("You run the program as an privileged user."
        ));

    return 0;
}
```

2.7.1. Перевод на русский язык

При создании приложения строки текста, подлежащего переводу на русский язык должны быть отмечены вызовами `tr()`:

```
QObject::tr("This text is subject to translate")
```

Для создания исходного файла для русского перевода следует воспользоваться командой `lupdate -verbose fly-demo.pro`. В результате работы этой команды будет создан файл `fly-demo_ru.ts`, содержащий строки, отмеченные в исходном коде как подлежащие переводу.

Собственно перевод осуществляется с помощью графического инструмента `linguist`:

```
linguist fly-demo_ru.ts.
```

При этом каждой строке из исходного файла `fly-demo_ru.ts` ставится в соответствие русский перевод, который и будет использован при работе пакета.

2.8. Взаимодействие с рабочим столом

2.8.1. Иконки и заголовки окон

Qt дает достаточно функций для установки заголовков, иконок и подписей к иконкам для высокоуровневых окон приложений. Следует обратить внимание на обязательные к использованию методы класса `QWidget`: `setWindowTitle` и `setWindowIcon`. При этом следует придерживаться следующих рекомендаций. Заголовок окна формируется в таком порядке:

```
[имя открытого файла - ] имя приложения
```

где `имя открытого файла` соответствует имени файла, как правило, без полного пути, открытого в активном окне приложения;

`имя приложения`, как правило, соответствует его названию в меню «Пуск» (панели управления) или детализирует его.

Подпись к иконке можно задавать с помощью `setWindowIconText`. Она, как правило, соответствует заголовку окна или может быть короче его в связи с тем, что пространство для нее крайне ограничено. Например, в подписи к иконке может отсутствовать имя файла, если приложение изменяет только один файл, например, содержащий тему рабочего стола или «горячие» клавиши. В принципе, допускается, чтобы заголовок окна и (или) подпись к иконке содержали только имя приложения.

Иконку по умолчанию для всех окон приложения можно задать с помощью метода `QApplication::setWindowIcon`.

Заметим, что вызов `setWindowIcon(const QIcon & icon)` принимает `QIcon` как аргумент.

2.8.2. Системный трей

Любая программа рабочего стола, призванная постоянно находиться в работе, для экономии экранного пространства должна предоставлять пользователю возможность своего сворачивания в трей и восстановления из него. Самый простой путь для программной реализации этого — использовать Qt-класс `QSystemTrayIcon`, реализующий стандарты [11], [12]. При установке иконки для `QSystemTrayIcon` рекомендуется использовать `QIcon`, которая может выдавать изображения всех типичных размеров (см., например, `QIcon::fromTheme`).

Начиная с версии ОС CH Смоленск 1.6 появилась возможность использовать для панели задач (трея и тулбара) монохромные темы иконок `fly-astra-flat-white` и `fly-astra-flat-black`. Приложение может разместить свои чёрные и белые иконки, предназначенные для трея, в эти две темы, и переключение будет происходить автоматически при смене цвета панели задач.

2.8.3. Автозапуск

Если приложение требуется запускать сразу после запуска оконного менеджера, его можно поместить в категорию «Автозапуск» с помощью функции библиотеки `libflycore` (см. `flydeapi.h`):

```
bool flyAutostartEnabled(const char *appname);
```

проверяет, есть ли в автозапуске (как в системном, так и в пользовательском) ярлык с именем `appname.desktop` и является ли он корректным.

```
bool enableAutostart(const char *appname, bool enable,
                    char *newExec=NULL, Display *dpy=NULL);
```

устанавливает в пользовательский автозапуск ярлык `appname.desktop` (можно указать полный путь к файлу типа `desktop`), если `enable=true`, или удаляет, если `enable=false`. При этом, в копии `desktop`-файла, которая попадет в автозапуск, можно скорректировать строку запуска `Exec`, задав `newExec`. Это может быть полезно, если программе надо передать дополнительный аргумент, свидетельствующий, например, что ее запускают именно из автозапуска, а не из меню и т. п.

Если в системном автозапуске есть `appname.desktop`, а вызывается `flyEnableAutostart(appname, false)`, то в пользовательском автозапуске будет создан фиктивный `appname.desktop` с полем `Hidden=true`, что согласно стандарту [15] является пользовательским методом отключения системного автозапуска.

2.8.4. Меню «Пуск» и рабочий стол

Меню «Пуск» (стартовое меню) создается для каждого пользователя при первом запуске им рабочего стола Fly. В первый раз меню создается автоматически на основе файлов `/usr/share/applications/*.desktop` с `Type=Applications`. Иконки, имена, параметры запуска и т. п. берутся оттуда же (см. «Desktop Entry Standard» от `freedesktop.org` [6]).

Стартовое меню имеет несколько уровней. Первый уровень обычно формируется на основе `/usr/share/fly-wm/startmenu` и выглядит так:

«Офис»
 «Сеть»
 «Графика»
 «Мультимедиа»
 «Научные»
 «Игры»
 «Мобильные»
 «Разработка»
 «Утилиты»
 «Системные»
 –разделитель–
 «Последние»
 «Панель управления»
 «Менеджер файлов»
 –разделитель–
 «Завершение работы...»

В панель управления попадают только приложения, имеющие определенные поля в `*.desktop`-файле (см. 2.5.2). При этом, если задано поле `X-Fly-AdminOnly=true`, то соответствующий пункт будет виден только пользователю `root`. Последнее правило распространяется не только на «Панель управления», но и на все пункты меню.

Начиная с версии ОС СН Смоленск 1.6 добавлена новая категория Мобильные, предназначенная для программ, сделанных специально под тачскрины, т.е., под управление пальцами.

Формирование указанных выше подменю происходит автоматически на основе `Categories`, определенных в документе «Desktop Menu Specification» от `freedesktop.org` [13].

Подменю и категории, по которым приложения в них попадают:

- «Офис» — Office, Spreadsheet, WordProcessor, Presentation, Calendar, Email;
- «Сеть» — Network, Internet, Email;
- «Графика» — Graphics, VectorGraphics, RasterGraphics, Screensaver;
- «Мультимедиа» — AudioVideo, Multimedia;
- «Научные» — Education, Science, Math, Astronomy, Physics, Chemistry;
- «Игры» — Game, *Game;
- «Мобильные» — Mobile;
- «Разработка» — Development, GUIDesigner, IDE, TextEditor;

- «Утилиты» — System, PackageManager;
- «Системные» — SystemSetup, Settings, AdvancedSettings, Accessibility;
- «Прочие» — все, что не удалось разместить в других меню.

Вышеперечисленные категории представлены в порядке своего приоритета. Так, приложение, содержащее ключевое слово «Email», скорее всего, попадет в подменю «Сеть», чем в «Работа с файлами», так как в «Сеть» оно имеет позицию 3, а в «Работа с файлами» — 5. С другой стороны, одно и то же приложение может задавать в своем *.desktop-файле несколько категорий в порядке приоритета. Тогда приложение со списком категорий «Network»; «Office» попадет в меню «Сеть».

Если у приложения задана только категория «Application», то оно попадет сразу в подменю «Программы», если такое подменю есть. Если указана специальная категория Core, то приложение попадет на нулевой уровень, т. е. прямо в меню «Пуск». При категории None приложение никуда не попадет. Категории можно присваивать и каталогам (см. 2.5.2). В этом случае каталог помещается сразу целиком в соответствии со своими категориями так же, как это делается для обычных программ. Таким образом, разработчик стороннего приложения может разместить информацию для запуска как отдельным ярлыком, так и целым каталогом в любом из подменю меню «Пуск» без каких-либо ограничений.

Сформированное меню на диске располагается в \$HOME/.fly/startmenu и является иерархией каталогов (им соответствуют подменю) и *.desktop-файлов (им соответствуют пункты меню). Есть и специальная разновидность *.desktop-файлов — .directory-файлы, имеющие сходную структуру и задающие внешние имена, иконки категорий для самих каталогов (подменю), а также порядок сортировки при показе содержимого каталогов (см. поле sortOrder в файле .directory).

При старте рабочего стола персональное меню (\$HOME/.fly/startmenu) и меню для всех пользователей (/usr/share/fly-wm/startmenu) сливается воедино в \$HOME/.fly/startmenu. Меню как иерархия в ФС сделано вопреки документу «Desktop Menu Specification» от freedesktop.org (там рекомендуется XML-файл [13]), чтобы облегчить ручную правку меню, как это сделано в MS Windows. С другой стороны, Fly предоставляет пользователю мощную программу fly-menedit для просмотра, редактирования, резервного копирования и автоматического создания не только меню «Пуск», но и вообще любых иерархических совокупностей каталогов и (или) файлов типа *.desktop.

Любое приложение, желающее быть показанным в меню «Пуск» для каждого пользователя, должно в первую очередь разместить в /usr/share/applications/ свой *.desktop-файл или каталог с такими или иными файлами. Тогда при следующем запуске fly-wm или создании меню «с нуля» в fly-menedit приложение автоматически будет включено в меню в соответствии со своей категорией из *.desktop-файла или .directory-

каталога. Однако, если приложение хочет попасть в стартовое меню только одного пользователя, то оно может скопировать свой *.desktop или каталог непосредственно в один из подкаталогов \$HOME/.fly/startmenu. При этом весьма полезная информация не будет доступна никому другому.

Иконки на рабочем столе по сути представляют из себя одноуровневое меню в каталогах \$HOME/Desktop (персональный набор) и /usr/share/fly-wm/desktop (набор для всех пользователей). Все сказанное выше для стартового меню справедливо и для набора иконок на рабочем столе.

При каждом запуске менеджер окон fly-wm контролирует наличие обновлений в /usr/share/applications. И если там объявилось новое приложение, то его ярлык попадает в меню «Пуск» каждого пользователя (при запуске fly-wm).

В дальнейшем пользователь может удалить этот ярлык.

Fly-wm контролирует наличие обновлений в /usr/share/fly-wm/desktop (общие ярлыки рабочего стола для всех пользователей).

Справа от кнопки **[Пуск]** на панели задач располагается панель с кнопками для быстрого запуска наиболее употребительных приложений. Соответствующие ярлыки располагаются в \$HOME/.fly/toolbar, вызывать этот каталог следует с помощью:

```
GetFlyDir(USER_TOOLBAR_DIR)
```

Начиная с версии ОС CN Смоленск 1.6 в desktop-файлах можно определять их участие или неучастие в определённых типах сессий (см. \$DESKTOP_SESSION) путем указания имён сессий в полях NotShowIn и OnlyShowIn.

2.8.5. Корзина

«Корзина» представляет собой каталог, который индивидуален для каждого пользователя. Он предназначен для временного хранения удаленных пользователем документов с возможностью их восстановления.

Путь к корзине программно можно получить с помощью:

```
GetFlyDir(USER_TRASH_DIR)
```

из библиотеки libflycore и обычно он равен \$HOME/.local/share/Trash.

Оконный менеджер fly-wm, файловый менеджер fly-fm и другие программы рабочего стола позволяют выполнять файловые операции. Они доступны как через меню, так и с помощью перетаскивания мышью (drag-n-drop). Одной из таких операций является удаление файлов или папок в «Корзину» и их восстановление. Все операции с корзиной (перемещение в нее, восстановление из нее, получение сведений о размещенных там файлах и каталогах) выполняются в соответствии со стандартом от freedesktop.org. При этом не будет иметь значения, где физически располагается корзина и ее реестр (на самом деле это два подкаталога в каталоге \$HOME/.local/share/Trash), если разработчик использует API корзины, предоставленный классом FlyTrash библиотеки libflycore:

```

static std::string registerFile(const char *srcPath);
static bool unregisterFile(const char *fileName);
static std::string getFilePathToRestore(const char *fileNameOrPath);

static bool isFileInTrash(const char * path);

static bool cleanRegistry();
static bool cleanFiles();
static bool cleanAll();

static bool isEmpty();

static bool moveToTrash(const char *filePath);
static bool restoreFromTrash(const char *fileNameOrPath);

```

Необходимо отметить, что API не предназначен непосредственно для перемещения/восстановления файлов, а только для получения необходимых для этого путей и регистрации/удаления в реестре корзины. Однако две последние из приведенных выше функций упрощают задачу для небольших файлов. Они применимы только, если работа выполняется в рамках одной ФС (одного раздела диска).

2.8.6. Перетаскивание объектов на рабочем столе

Рабочий стол Fly в прямом смысле — это корневое окно с иконками на нем. Иконки — это либо ярлыки (desktop entry), либо собственно файлы или каталоги. Операция drag-n-drop (dnd — перетаскивание мышью) возможна как на сам рабочий стол, так и на иконки на нем. Реализация dnd в fly-wm и приложениях Qt соответствует стандарту [9].

Иконка рабочего стола может принимать dnd, если она представляет собой:

- собственно каталог или ярлык каталога (например, myhome.desktop, mydoc.desktop и ряд других специальных ярлыков, в том числе mytrash.desktop — ярлык корзины);
- ярлык типа Application, если в его поле Exec= есть один из аргументов: %F — список файлов, %f — файл, %u — URL, %U — список URL.

Для поддержки печати с помощью операции dnd на рабочем столе приложение, предназначенное для вывода документов на печать, должно:

- принимать аргумент командной строки --print %F или %f, %u, %U, при этом передавая файлы на печать;
- описать свою возможность печатать в *.desktop-файле как Action с именем «Print» и командной строкой, содержащей --print %F или %f, %u, %U.

При перетаскивании мышью пунктов меню «Приложения» или «Панель управления» на рабочий стол на нем создаются соответствующие копии.

2.8.7. Подсистема помощи

Подсистема помощи Fly состоит из:

- программы показа;
- хранилища файлов;
- клиентов.

Основная программа показа помощи — Qt Assistant. Структура хранилища и формат файлов помощи определяются пакетом `fly-doc`. Клиенты программы показа помощи (все приложения Fly) вызывают несколько простых функций (см. в библиотеке `libflyui` класс `FlyHelpMenu` и функции `flyHelpInstall()`, `flyHelpShow()` и т.п.) и не должны заботиться о месте расположения файлов помощи и способах их показа.

2.8.8. Синхронизация с менеджером окон fly-wm

В связи с тем, что любая программа может перемещать файлы в/из рабочего стола (`$HOME/Desktop`), корзины (`$HOME/.local/share/Trash`), стартового меню (`$HOME/.fly/startmenu`), возникает проблема синхронизации содержимого данных каталогов с оконным менеджером `fly-wm`. Требуется сообщать `fly-wm` о том, что произошли изменения и ему требуется перечитать эти каталоги.

Каждая программа, перемещающая файлы в/из каталогов:

- рабочего стола (по `GetFlyDir(USER_DESKTOP_DIR)`);
- корзины (по `GetFlyDir(USER_TRASH_DIR)`);
- стартового меню (по `GetFlyDir(USER_START_MENU_DIR)`).

должна, по окончании операции, уведомить `fly-wm` следующим способом:

- `SendCommandToWM("FLYWM_UPDATE_SHORTCUT\n");`
- `SendCommandToWM("FLYWM_UPDATE_TRASH\n");`
- `SendCommandToWM("FLYWM_UPDATE_STARTMENU\n").`

соответственно.

В этом случае `fly-wm` обновит рабочий стол, корзину, стартовое меню, соответственно. Функция `SendCommandToWM(...)` содержится в библиотеке `libflycore`.

Начиная с версии ОС CH Смоленск 1.5 эти уведомления посылать не обязательно, т.к., используя механизм `inotify`, `fly-fm` сам «понимает» момент, когда необходимо сделать обновление.

Используя тот же механизм, можно отправлять различные команды оконному менеджеру. Например, если некоторому приложению требуется, чтобы были закрыты все текущие приложения и графическая система была перезагружена, оно может использовать функцию

SendCommandToWM из libflycore с параметром FLYWM_RESTART. Практически полный список команд можно увидеть в приложении fly-admin-hotkeys (редактор «горячих» клавиш), большинство из них можно посылать в fly-wm из других приложений.

Полный список команд, принимаемых менеджером окон, можно получить в файле .xsession-errors после выполнения консольной команды fly-wmfunc FLYWM_FUNC_LIST.

2.8.9. Полноэкранный режим

Данная функция может присутствовать как пункт «Полноэкранный режим» в меню «Окна» (для MDI-интерфейсов) или «Вид» (для SDI). Она выполняется с помощью Qt-вызовов showFullScreen() и showNormal().

2.8.10. Смена раскладки клавиатуры

Сменить раскладку клавиатуры можно одной строкой из программы:

```
SendCommandToWM(QX11Info::display(), "FLYWM_ALT_KB\n")
SendCommandToWM(QX11Info::display(),
"FLYWM_BASE_KB\n")
```

или из командной строки:

```
> fly-wmfunc FLYWM_ALT_KB или FLYWM_BASE_KB
```

Аналогично для переключения дополнительной секции клавиатуры на ввод цифр (NumLock) и наоборот можно использовать команды: FLYWM_NUMLOCK_ON, FLYWM_NUMLOCK_OFF, FLYWM_NUMLOCK_TOGGLE.

2.8.11. Дополнительные панели

Fly-wm предоставляет одну панель задач. Окна при максимизации не накрывают ее, иконки рабочего стола не попадают под нее. Таким образом, панель задач образует некую специальную зону экрана. В [10] описаны такие свойства окон приложений: _NET_WM_STRUT и _NET_WM_STRUT_PARTIAL, позволяющие приложению резервировать области экрана подобно панели задач. Fly-wm поддерживает эту возможность. Однако в Qt нет простой поддержки данной возможности. Рекомендуется использовать такой фрагмент:

```
int v[4];
v[0] = w.x(); v[1] = w.x()+w.frameGeometry().width();
v[2] = w.y(); v[3] = w.y()+w.frameGeometry().height();
Atom a=XInternAtom(QX11Info::display(), "_NET_WM_STRUT", false);
if (a!=None) XChangeProperty(QX11Info::display(), w.winId(), a,
XA_CARDINAL, 32, PropModeReplace, (unsigned char *)&v, 4);
```

где w — главное окно приложения.

Через параметр Qt::WindowFlags при создании виджета можно задать его внешний вид (наличие границы, заголовка и т.п.). Однако, есть более высокоуровневый и переносимый способ задания STRUT - использовать функцию KWindowSystem::setStrut() из

библиотеки libKF5WindowSystem.

Начиная с версии ОС СН Смоленск 1.6 можно задать область экрана, которую займёт рабочий стол, с помощью параметров FlyDesktopX, FlyDesktopY, FlyDesktopWidth, FlyDesktopHeight в файлах `~/HOME/.fly/theme/*themerc*`.

2.9. Плагины для менеджера файлов Fly-fm

В файловом менеджере реализована возможность расширения его функционала с помощью так называемых «плагинов» (см. электронную справку по файловому менеджеру `fly-fm`).

2.10. Настройки планшетного режима

В версии ОС СН Смоленск 1.6 имеется возможность автоматического конфигурирования графического интерфейса пользователя для работы на мобильных устройствах с сенсорными экранами — т.е. в так называемых «планшетной» и «мобильной» типах сессии. Тип сессии выбирается пользователем при входе в систему, см. меню "Тип сессии" графического логина. При входе в систему будет применен комплект конфигурационных файлов соответствующий типу выбранной сессии и выставлена переменная `~$DESKTOP_SESSION`.

2.11. Приложения и права администратора

Большинство утилит настройки требуют привилегий суперпользователя. Есть ряд способов предоставления привилегий: от `sudo` и членства в группах до PolicyKit. Решение этой задачи возможно с использованием программы `fly-su`. Программа, предполагающая действия администратора, обязательно должна информировать пользователя о невозможности выполнения каких-либо функций без соответствующей авторизации и, по возможности, давать пользователю способ выполнить такую авторизацию, например, с помощью рекомендуемой `fly-su`.

Начиная с версии ОС СН Смоленск 1.6 в системе может быть ограничено использование консоли. В таком случае пользователи, не входящие в группы `astra-admin` или `astra-console`, не увидят `desktop`-файлов, в которых используются терминалы. Пользователи, не входящие в группу `astra-admin`, не увидят `desktop`-файлов, в которых есть не только категория `ROOT_ONLY`, но и команды `su-to-root`, `fly-su`.

2.12. Межмандатный буфер обмена

В версии ОС СН Смоленск 1.6 имеется возможность копирования-вставки (`copy-paste`) между программами с различными мандатными метками, даже если эти программы выполняются в разных сессиях. При этом, т.к. весь процесс происходит через файлы, то допустимость копирования-вставки автоматически контролируется на уровне файловой системы. Для включения этой возможности надо в домашнем каталоге пользователя в

файлах `.fly/theme/*.themerc` задать `UseClipboardManagerMAC=true`. Обратите внимание, что для разных мандатных меток имеются разные домашние каталоги, и настройку нужно выполнить во всех этих каталогах.

2.13. Приложения и мандатная политика

Для разработки графических приложений с использованием библиотек графических интерфейсов Qt 5 в ОС CH реализована библиотека `libparsec-mac-qt5-1`, предоставляющая удобный доступ к функциям подсистемы безопасности PARSEC и содержащая набор классов для работы с мандатными атрибутами (3.1). Для разработки приложений с использованием указанной библиотеки необходимо установить пакет `libparsec-mac-qt5-1-dev`.

Далее приведены примеры использования классов из библиотеки `libparsec-mac-qt5-1`.

Примеры:

1. Создание диалога для установки мандатных атрибутов объектов ФС

```
/* *** Файл main.cpp *** */
#include <QApplication>

#include "filemaclabeldialog.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    FileMacLabelDialog dlg("/usr/bin");
    dlg.exec();

    return 0;
}

/* *** Файл filemacdialog.h *** */

#ifndef FILEMACLABELDIALOG_H
#define FILEMACLABELDIALOG_H

#include <QDialog>
#include <QVBoxLayout>
#include <QDialogButtonBox>
#include <QLabel>
```

```

#include <QPushButton>
#include <QMessageBox>

#include <parsec-qt5/filemaclabelwidget.h>

class FileMacLabelDialog : public QDialog
{
    Q_OBJECT
public:
    FileMacLabelDialog(const QString& filePath) : m_filePath(filePath)
    {
        QVBoxLayout *layout = new QVBoxLayout(this);
        QLabel* label = new QLabel(tr("MAC for %1:").arg(m_filePath));
        layout->addWidget(label);

//создание экземпляра класса для работы с мандатными атрибутами
//объектов ФС и получение мандатных атрибутов объекта ФС

        m_fileMacLabelWidget = new ParsecQt5::FileMacLabelWidget;
        layout->addWidget(m_fileMacLabelWidget);
        m_fileMacLabelWidget->getfmac(m_filePath);
        connect(m_fileMacLabelWidget, SIGNAL(changed()), this, SLOT(setChanged()))

        m_buttonBox = new QDialogButtonBox(QDialogButtonBox::Ok |
        QDialogButtonBox::Cancel);
        connect(m_buttonBox, SIGNAL(accepted()), this, SLOT(accept()));
        connect(m_buttonBox, SIGNAL(rejected()), this, SLOT(reject()));
        layout->addWidget(m_buttonBox);
        setChanged(false);
    }

~FileMacLabelDialog() {}

    bool isChanged() const
    {
        return m_buttonBox->button(QDialogButtonBox::Ok)->isEnabled();
    }
}

```

```

private slots:

    void setChanged(bool changed = true)
    {
        m_buttonBox->button(QDialogButtonBox::Ok)->setEnabled(changed);
    }

    void accept()
    {
        if (isChanged()) {
//установка мандатных атрибутов объекта ФС
            if (!m_fileMacLabelWidget->chmac(m_filePath)) {
                QMessageBox::critical(this, tr("Error"), tr("MAC can't be changed"));
                return;
            }
        }
    }

QDialog::accept();
}

protected:

ParsecQt5::FileMacLabelWidget* m_fileMacLabelWidget;
QDialogButtonBox* m_buttonBox;
QString m_filePath;
};

#endif // FILEMACLABELDIALOG_H

```

2. Использование диалога выбора мандатных атрибутов пользователя при входе в систему через графический интерфейс пользователя

```
/* *** Файл main.cpp *** */
```

```

#include <QApplication>
#include <QLocale>
#include <QTranslator>
#include <QX11Info>

```



```
#include <parsec-qt5/loginmaclabeldialog.h>

#include <X11/Xlib.h>
#include <stdio.h>
#include <getopt.h>
#include <pwd.h>
#include <stdlib.h>
#include <unistd.h>
#include <iostream>

#include <fly/flyfileutils.h>

using namespace ParsecQt5;

bool user_id_specified = false;

struct arguments {
    uid_t user_id;
};

static arguments cmd_params;

bool process_user_name( const char * name )
{
    if ( ! name )
    {
        fprintf( stderr, "No name specified\n" );
        return false;
    }

    struct passwd * pwd = getpwnam( name );
    if ( ! pwd )
    {
        fprintf( stderr, "No such user %s\n", name );
        return false;
    }

    cmd_params.user_id = pwd->pw_uid;
}
```

```

    user_id_specified = true;
    return true;
}

void show_help()
{
    printf( "Options :\n"
           "-u, --user : username\n"
           "-h, --help : this help\n");
}

void print_error()
{
    fprintf( stderr, "Error : invalid arguments\n" );
}

/* returns true, if ok */
bool process_args( int argc, char ** argv )
{
    struct option longopts[] = {
        { "user", 1, NULL, 'u' },
        { "help", 0, NULL, 'h' },
        { 0, 0, 0, 0 }
    };

    int c;

    while( ( c = getopt_long( argc, argv,
                             "u:h" /*"u:w:l:c:htea"*/, longopts, NULL ) ) != -1 )
    {
        switch( c ) {
            case 'u':
                if ( ! process_user_name( optarg ) )
                {
                    print_error();
                    show_help();
                    return false;
                } else {

```

```

        break;
    }
    case 'h':
        show_help();
        return false;
    default:
/* Для поддержки опций QApplication,
   игнорируем все неизвестные опции */
        break;
    }
}
return true;
}

void print_result( LoginMacLabelDialog * dialog )
{
    if ( ! dialog ) return;

    fprintf( stdout, "%x ", dialog->level() );

//print in hex, new pam accept hex only!
    fprintf( stdout, "%lx ", dialog->categories() );

    std::cout << std::endl;
}

int main( int argc, char ** argv )
{
    if ( ! getenv( "LANG" ) ) setenv( "LANG", "ru_RU.UTF-8", 0 );

    QApplication app( argc, argv );

    if ( ! process_args( argc, argv ) ) exit( 1 );

    QTranslator translator( 0 );
    translator.load( "fly-mac-dialog_" + QLocale::system().name(),
        QString::fromStdString(GetFlyDir(APP_TRANS_DIR)) );
    app.installTranslator( &translator );
}

```

```
//создание диалога выбора мандатных атрибутов пользователя
//при графическом входе в систему

LoginMacLabelDialog * dialog = new LoginMacLabelDialog(
user_id_specified? cmd_params.user_id:getuid(), 0 );

if ( !dialog->isValid() )
{
    fprintf( stderr, "Dialog invalid\n" );
    exit( 1 );
}

int result = QDialog::Accepted;

if ( dialog->isSuitable() ) //don't show if not suitable but accept
{
    dialog->show();
    dialog->activateWindow();
    dialog->setFocus();
    QObject::connect(dialog, SIGNAL(finished(int)), &app, SLOT(quit()));

    app.exec();

    result = dialog->result();
}

if ( result == QDialog::Accepted )
{
    print_result( dialog );
    delete dialog;
    return 0;
}

delete dialog;
return 1;
}
```

3. ПРОГРАММНЫЙ ИНТЕРФЕЙС МАНДАТНОГО РАЗГРАНИЧЕНИЯ ДОСТУПА

3.1. Общие сведения

В ОС СН реализован механизм мандатного разграничения доступа. При этом, принятие решения о запрете или разрешении доступа субъекта к объекту принимается на основе типа операции (чтение/запись/исполнение), мандатного контекста безопасности субъекта и мандатной метки объекта (подробнее об этом в Руководстве [2], подраздел 4.1). Кроме того, при принятии решения могут учитываться специальные привилегии субъекта (см. [2], подраздел 4.2).

3.2. Сетевое взаимодействие

3.2.1. Передача мандатных меток по сети

В ОС СН сетевые соединения рассматриваются как средство межпроцессного взаимодействия и подвергаются мандатному контролю доступа. Для этого в сетевые пакеты протокола IPv4 в соответствии со стандартом ГОСТ Р 58256-2018 внедряются мандатные метки объекта – сетевое соединение (сокет). При этом метка объекта наследуется от субъекта (процесса). Прием сетевых пакетов подчиняется мандатным правилам разграничения доступа. Метка объекта может иметь тип, позволяющий создавать сетевые сервисы, принимающие соединения с любыми мандатными метками.

Метки передаются в поле опций IPv4 пакета. Для передачи меток по сети используется вариант опции DoD Security Option IPv4 (см. RFC1108 — Security Options for the Internet Protocol — Параметры безопасности для IP-протокола).

В соответствии с ГОСТ Р 58256-2018 метка снабжается классом 0хАВ (Unclassified), при этом последующий битовый список (последовательность байт, в которых младший бит указывает на наличие следующего байта в потоке) опции представляет собой упакованную в соответствии со стандартом структуру мандатного контекста, где уровень конфиденциальности занимает 8 бит, а категории доступа — 64 бита (порядок байт — от младших к старшим). Последние (старшие) нулевые биты могут быть отброшены.

Поле опций IPv4 при передаче мандатных меток представлено в таблице 1.

Таблица 1

Тип	Длина	Описание
130	Значение	DoD Basic Security: используется для передачи классификационных меток и флагов защиты авторизации

Таким образом, для передачи мандатных меток по сети используется формат, полностью соответствующий требованиям ГОСТ Р 58256-2018, и совместимый с международным форматом RFC 1108 DoD Basic Security (см. таблицу 2).

Таблица 2

TYPE = 130	LENGTH	CLASSIFICATION LEVEL (Unclassified)	PROTECTION AUTHORITY FLAGS
10000010	XXXXXXXX	10101011	AAAAAAA[1 0] AAAAAA0

Поле LENGTH содержит значение длины опции. Минимальная длина опции 3 октета, включая поля TYPE и LENGTH. Поле PROTECTION AUTHORITY FLAGS может отсутствовать. Значение поля LENGTH меньше 3 октетов должно сигнализировать об ошибке.

Поле CLASSIFICATION LEVEL всегда выставляется в 10101011 (Unclassified).

Поле PROTECTION AUTHORITY FLAGS имеет переменную длину, младший бит каждого октета (байта) используется для индикации наличия следующего октета. Если бит равен 1, есть следующий октет, если бит равен 0 — октет последний.

Метка в подсистеме безопасности PARSEC в ОС CH имеет следующий формат:

```
struct parsec_mac_label {
uint8_t level;
uint64_t categories;
};
```

где:

level — Уровень (беззнаковое целое число 256 возможных значений);

categories — Категории (битовая маска little-endian, всего 64 категории), структура должна быть упакована (выравнивание полей отключено).

В поле PROTECTION AUTHORITY FLAGS данная метка записывается следующим образом: метка представляется в виде последовательности бит, начиная с младшего байта, вставляется младший бит признака продолжения, со сдвигом остальных бит. Далее анализируются октеты с конца поля PROTECTION AUTHORITY FLAGS. В соответствии со стандартом октеты, полезные 7 бит которых содержат 0, отбрасываются с корректировкой поля длины.

Таким образом, при отображении структуры parsec_mac_label длиной 9 байт в поле PROTECTION AUTHORITY FLAGS длина данного поля может варьироваться от 0 (для пустой метки {0, 0}) до 11 октетов.

Пример кодирования метки для протокола IPv4:

```
IPORT_SEC, 5, 0xAB, 03, 12
```

Поле PROTECTION AUTHORITY FLAGS в виде битового списка:

```
00000011, 00001100
```

Мандатный контекст:

уровень 1, категории 3

Отсутствие метки на объекте доступа является синонимом нулевой мандатной метки. Таким образом, ядро ОС, в которой все объекты и субъекты доступа имеют мандатный уровень 0, функционирует аналогично стандартному ядру ОС Linux.

3.2.2. Разработка ПО для обработки информации с различными уровнями конфиденциальности

В ОС CH реализован прикладной программный интерфейс (API) подсистемы безопасности PARSEC для разработки ПО, предназначенного для обработки информации с различными уровнями конфиденциальности.

Как правило, при разработке программного средства, предназначенного для обработки информации, содержащейся в объектах ФС, от имени пользователя, запустившего данное программное средство, или при разработке программного средства, выполняющего отображение информации, обработка которой осуществляется некоторым сервисом, нет необходимости задействовать API подсистемы безопасности PARSEC.

Наиболее распространенной задачей, при решении которой возникает такая необходимость, является разработка сетевого сервиса, предназначенного для обработки и предоставления информации по запросам различных пользователей, имеющих разные полномочия на доступ к информации. В указанном случае рекомендуется использовать следующий обобщенный алгоритм работы сетевого сервиса при взаимодействии с пользователем в условиях мандатного разграничения доступа в ОС CH:

- 1) основной процесс сетевого сервиса, осуществляющего обработку информации с различными мандатными уровнями, открывает привилегированный слушающий сокет, имеющий возможность принимать соединения с любыми мандатными метками. Основной процесс сетевого сервиса должен обладать определенными привилегиями Linux и привилегиями подсистемы безопасности PARSEC (см. [2], подраздел 4.2);
- 2) выполняется установка соединения;
- 3) выполняется идентификация и аутентификация пользователя, от имени которого функционирует клиентский процесс;
- 4) в случае успешного завершения аутентификации сетевой сервис создает дочерний процесс, в котором будет осуществляться обработка информации в контексте пользователя;
- 5) в дочернем процессе с сокета, обслуживающего соединение, снимается метка, передаваемая по сети;
- 6) в дочернем процессе осуществляется установка необходимых привилегий Linux и привилегий подсистемы безопасности PARSEC;
- 7) осуществляется переключение дочернего процесса в контекст пользователя (установка мандатного уровня, мандатных категорий, gid и uid пользователя);

8) в дочернем процессе осуществляется сброс привилегией Linux и подсистемы безопасности PARSEC.

Дальнейшая обработка информации при взаимодействии с пользователем в рамках установленного соединения осуществляется в контексте дочернего процесса.

Пользователь, от имени которого запускается основной процесс сетевого сервиса, должен обладать следующими привилегиями Linux:

- CAP_SETGID — для переключения дочернего процесса в работу с правами группы пользователя, являющегося клиентом сервиса;
- CAP_SETUID — для переключения дочернего процесса в работу с правами пользователя, являющегося клиентом сервиса.

Примечание. Если разрабатываемый сетевой сервис должен ожидать входящие соединения с использованием значения порта меньшего либо равного 1024, необходимо использовать привилегию Linux CAP_NET_BIND_SERVICE.

Кроме того, пользователь, от имени которого запускается основной процесс сетевого сервиса, должен обладать следующими привилегиями подсистемы безопасности PARSEC (см. [2], подраздел 4.2):

- PARSEC_CAP_PRIV_SOCKET — для создания привилегированного сокета, имеющего возможность принимать соединения с любыми мандатными метками;
- PARSEC_CAP_SETMAC — для переключения дочернего процесса в работу с мандатным контекстом пользователя, являющегося клиентом сервиса.

При разработке сетевого сервиса, предназначенного для обработки и предоставления информации по запросам различных пользователей, имеющих разные полномочия на доступ к информации, рекомендуется использовать принцип наименьших привилегий. Названный принцип означает, что процесс должен обладать только теми привилегиями, которые необходимы для решения его функциональных задач. По завершении решения функциональных задач, требующих наличия привилегией, необходимо сбросить привилегии и продолжить функционирование в соответствии с общими ПРД.

Далее приведен пример реализации рассмотренного обобщенного алгоритма работы сетевого сервиса на языке C.

Пример

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <sys/prctl.h>
#include <sys/types.h>
```



```

#include <sys/capability.h>
#include <unistd.h>
#include <linux/prctl.h>
#include <sys/types.h> /* See NOTES */
#include <sys/socket.h>
#include <inttypes.h>
#include <netdb.h>
#include <parsec/mac.h>
#include <parsec/parsec_integration.h>
#include <parsec/parsec_mac.h>
#include <pwd.h>
#include <fcntl.h>
#include <errno.h>

#define T(fmt, ...) do { \
    fprintf( stderr, "%s %s %d:", __FILE__, __FUNCTION__, __LINE__); \
    fprintf( stderr, fmt, ##__VA_ARGS__); \
    fprintf( stderr, " error: %m\n"); \
    exit(1);\
} while(0)

static volatile int keepRunning = 1;

static void intHandler(int dummy) {
    keepRunning = 0;
}

static int child_process( int clnt_sock)
{
    char user[128];

    // код, выполняемый в дочернем процессе,
    // предназначенном для обработки информации в контексте пользователя
    // объявление переменной для получения с сокета соединения
    // передаваемой по сети мандатной метки клиентского процесса пользователя
    parsec_mac_label_t mac_label;
    // объявление переменной для получения uid и gid пользователя
    struct passwd* pwd = NULL;

```

```
// объявление переменной, используемой при получении
// привилегий Linux дочернего процесса
struct __user_cap_header_struct cur_header;

printf( "child started\n");
//объявление переменной для установки привилегий Linux
// и ее начальная инициализация
linux_caps_t cur_lcaps={0,0,0};

//объявление переменной для установки привилегий PARSEC
// и ее начальная инициализация
parsec_caps_t cur_pcaps = {0,0,0};

//инициализация списка разрешенных привилегий Linux
cur_lcaps.permitted |= CAP_TO_MASK(CAP_SETUID);
cur_lcaps.permitted |= CAP_TO_MASK(CAP_SETGID);

//инициализация списка разрешенных привилегий PARSEC
cur_pcaps.cap_permitted |= CAP_TO_MASK(PARSEC_CAP_SETMAC);

//инициализация списка действующих (эффективных) привилегий Linux
cur_lcaps.effective |= CAP_TO_MASK(CAP_SETUID);
cur_lcaps.effective |= CAP_TO_MASK(CAP_SETGID);

//инициализация списка действующих (эффективных) привилегий PARSEC
cur_pcaps.cap_effective |= CAP_TO_MASK(PARSEC_CAP_SETMAC);

//установка привилегий Linux и PARSEC дочернего процесса сетевого сервиса
if(parsec_cur_caps_set(&cur_lcaps,&cur_pcaps) != 0) {
T("parsec_cur_caps_set");
}

// инициализация переменной, используемой при получении
// привилегий Linux дочернего процесса
cur_header.version = _LINUX_CAPABILITY_VERSION;
cur_header.pid = 0;
```

```

// получение привилегий Linux для дочернего процесса
if(capget((cap_user_header_t)(&cur_header), &cur_lcaps ) != 0) {
T("cap_user_header_t");
}

// проверка наличия необходимых эффективных привилегий Linux
// у дочернего процесса
if((!(cur_lcaps.effective & CAP_TO_MASK(CAP_SETUID))) ||
    (!(cur_lcaps.effective & CAP_TO_MASK(CAP_SETGID)))) {
T("cur_lcaps.effective");
}

// получение привилегий PARSEC для дочернего процесса
if(parsec_capget(0, &cur_pcaps ) != 0) {
T("parsec_capget");
}

// проверка наличия необходимых эффективных привилегий PARSEC у дочернего процесса
if(!(cur_pcaps.cap_effective & PARSEC_CAP_TO_MASK(PARSEC_CAP_SETMAC))) {
T("cur_pcaps.cap_effective");
}

// получение с сокета соединения передаваемой по сети
// мандатной метки клиентского процесса пользователя
if(parsec_fstatmac(clnt_sock,&mac_label)) {
T("parsec_fstatmac");
}

printf( "С сокета получена мандатная метка type=%x mac.lev=%d mac.cat=0x%"PRIx64"\n",
mac_label.type, mac_label.mac.lev, mac_label.mac.cat);
// установка на дочерний процесс мандатной метки,
// полученной с сокета соединения
if(parsec_setmac(0,&mac_label.mac) < 0) {
T("parsec_setmac");
}

// запрос имени пользователя
char prompt[] = "login:";

```

```

if( send( clnt_sock, prompt, strlen( prompt), 0) != strlen( prompt)) {
T("send");
}

// выполнение идентификации и аутентификации пользователя,
// от имени которого функционирует клиентский процесс,
// установка значения переменной user

int rc;
rc = recv( clnt_sock, user, sizeof(user)-1, 0);
if( rc <= 0) {
T("recv");
}
user[rc-2]=0;
printf( "username is \"%s\"\n", user);

// получение gid и uid для пользователя клиентского процесса,
// прошедшего идентификацию и аутентификацию
pwd=getpwnam(user);
if(!pwd) {
T("getpwnam");
}

printf( "user = %s UID=%d GID=%d\n", user, pwd->pw_uid, pwd->pw_gid);

// переключение дочернего процесса в работу с правами группы пользователя,
// прошедшего идентификацию и аутентификацию и являющегося клиентом
// сетевого сервиса
// переключение выполняется до вызова setuid(...)
if(setgid(pwd->pw_gid)) {
T("setgid");
}
// переключение дочернего процесса в работу с правами пользователя,
// прошедшего идентификацию и аутентификацию и являющегося клиентом
// сетевого сервиса
if(setuid(pwd->pw_uid)) {
T("setuid");
}

```

```

// сброс привилегий Linux и PARSEC дочернего процесса сетевого сервиса
if(parsec_cur_caps_set(0,0) != 0) {
T("parsec_cur_caps_set");
}
// код, выполняемый в дочернем процессе,
// для обработки и предоставления информации пользователю
// клиентского процесса

shutdown( clnt_sock, SHUT_RDWR);
close( clnt_sock);
printf( "child ended\n");
exit(0);
}

/*
*****
*/

int main( int argc, char **argv)
{
int sock = 0;
int clnt_sock = 0;
int addr_len = 0;

socklen_t addrlen;
struct sockaddr_in serv_addr, clnt_addr;
pid_t child;

signal(SIGINT, intHandler);

//объявление переменной для установки привилегий Linux и ее начальная инициализация
linux_caps_t lcaps={0,0,0};

//объявление переменной для установки привилегий PARSEC и ее начальная инициализация
parsec_caps_t pcaps = {0,0,0};

memset(&clnt_addr,0,sizeof(clnt_addr));

```

```
addr_len = sizeof(clnt_addr);
serv_addr.sin_port = htons(7777);

//инициализация списка разрешенных привилегий Linux
lcaps.permitted |= CAP_TO_MASK(CAP_SETUID);
lcaps.permitted |= CAP_TO_MASK(CAP_SETGID);

//инициализация списка действующих (эффективных) привилегий Linux
lcaps.effective |= CAP_TO_MASK(CAP_SETUID);
lcaps.effective |= CAP_TO_MASK(CAP_SETGID);

//инициализация списка наследуемых привилегий Linux
lcaps.inheritable |= CAP_TO_MASK(CAP_SETUID);
lcaps.inheritable |= CAP_TO_MASK(CAP_SETGID);

//инициализация списка разрешенных привилегий PARSEC
pcaps.cap_permitted |= CAP_TO_MASK(PARSEC_CAP_SETMAC);
pcaps.cap_permitted |= CAP_TO_MASK(PARSEC_CAP_PRIV_SOCKET);

//инициализация списка действующих (эффективных) привилегий PARSEC
pcaps.cap_effective|=CAP_TO_MASK(PARSEC_CAP_SETMAC);
pcaps.cap_effective|=CAP_TO_MASK(PARSEC_CAP_PRIV_SOCKET);

//инициализация списка наследуемых привилегий PARSEC
pcaps.cap_inheritable|=CAP_TO_MASK(PARSEC_CAP_PRIV_SOCKET);
pcaps.cap_inheritable|=CAP_TO_MASK(PARSEC_CAP_SETMAC);

//установка флага наследования привилегий
if(prctl(PR_SET_KEEPCAPS,1)) {
T("prctl");
}

//установка привилегий Linux и PARSEC основного процесса сетевого сервиса
if(parsec_cur_caps_set(&lcaps,&pcaps) < 0) {
T("parsec_cur_caps_set");
}

serv_addr.sin_family = AF_INET;
```

```
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);

// создание привилегированного сокета
printf("создание привилегированного сокета\n");
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
T("socket");
}

// привязка привилегированного сокета
printf("привязка привилегированного сокета\n");
if(bind(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) {
T("bind");
}
printf("привязка привилегированного сокета выполнена\n");

// включение ожидания запроса на соединение на привилегированном сокете
if(listen(sock,5) < 0) {
T("listen");
}

// перевод сокета в неблокирующее состояние
if( fcntl( sock, F_SETFL, O_NONBLOCK) < 0 ) {;
T("fcntl");
}

while( keepRunning) {
// создание соединения на привилегированном сокете
clnt_sock = accept(sock, (struct sockaddr*)&clnt_addr, &addr_len);
if( clnt_sock < 0) {
if( errno == EAGAIN)
continue;
T("accept");
}
printf( "Access detected\n");

// создание дочернего процесса для обработки информации в контексте пользователя
child = fork();
if(child<0) {
```

```

T("fork");
}

if(!child) {
child_process( clnt_sock);
}
}
shutdown( sock, SHUT_RDWR);
close( sock);
printf( "Done\n");
return 0;
}

```

Рекомендуется сбросить в основном процессе привилегию `PARSEC_CAP_PRIV_SOCKET`, как только исчезла необходимость ее использования, поскольку любые сокеты при наличии эффективной названной привилегии будут создаваться вызовом функции `socket(...)` как привилегированные.

Для сборки и запуска примера нужно скопировать код в отдельный файл, и использовать следующую команду:

```
gcc имя_файла_примера -lparsec-base -lparsec-mac && sudo ./a.out
```

В примере используется номер порта `7777`, и для подключения к запущенному примеру можно использовать команду `telnet`:

```
telnet <IP_адрес_компьютера> 7777
```

В случае локального компьютера можно использовать IP-адрес `127.0.0.1`

Вместо используемой в примере привилегии `PARSEC_CAP_PRIV_SOCKET` для создания привилегированного слушающего сокета, принимающего соединения с любыми мандатными метками, может быть использована следующая комбинация привилегий `PARSEC`:

- `PARSEC_CAP_CHMAC` — предоставляет право менять мандатную метку файла;
- `PARSEC_CAP_MAC_SOCKET` — предоставляет право менять мандатную метку сетевого соединения.

При наличии указанных привилегий после создания сокета необходимо с использованием функции `parsec_fchmac(...)` установить на сокет мандатные атрибуты игнорирования мандатных уровней и категорий при выполнении операция чтения и записи как показано в примере.

Пример

```

if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    // обработка ошибки
}

```



```
}

parsec_mac_label_t m_label;
m_label.type = MAC_ATTR_IGNORE;
m_label.mac.lev=0;
m_label.mac.cat=0;

if(parsec_fchmac(sock,&m_label) != 0)
{
    // обработка ошибки
}

if(bind(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)))
{
    // обработка ошибки
}
```

4. ОПИСАНИЕ ИСПОЛЬЗОВАНИЯ API СИСТЕМЫ РАСШИРЕННОГО АУДИТА

Для реализации функционала по аудиту рекомендуется использовать интегрированную систему аудита, которая входит в состав ОС СН.

Для получения подробного описания интегрированной системы аудита необходимо получить описание API и прикладных утилит вызовом электронной справки в ОС СН.

Доступ к электронной справке в системе осуществляется либо на рабочем столе Fly, либо вызовом справочной документации man.

В частности, для системы аудита вызывать команду man по следующему списку функций API и утилит:

```
aud_copy_ext
aud_get_type
aud_set_type
useraud
aud_create_entry
aud_init
aud_to_text
psaud
aud_dup
aud_next_entry
aud_valid
setaudent_r
aud_from_text
aud_set_file
getfaud
setaudent
aud_get_file
aud_set_pid
setfaud
```

5. РАЗРАБОТКА ПО ДЛЯ ВЗАИМОДЕЙСТВИЯ С СУБД POSTGRESQL

СУБД PostgreSQL предоставляет доступные программные интерфейсы, предназначенные как для разработки клиентского ПО, реализующего функции по вводу, формированию запросов и отображению данных, так и программные интерфейсы, предназначенные для расширения функциональности сервера. Основная документация по общим вопросам использования СУБД PostgreSQL расположена на официальном сайте разработчиков (<http://www.postgresql.org/docs/9.4/static/index.html>).

5.1. Мандатное разграничение доступа в СУБД PostgreSQL

Расширение команд SQL, используемое при мандатном разграничении в СУБД PostgreSQL, приведено в Руководстве [2](подраздел 4.10).

6. РАЗРАБОТКА ПО ДЛЯ WEB-СЕРВЕРА

При необходимости обеспечения сквозной аутентификации из скриптов, запускаемых на WEB-сервере, которые должны взаимодействовать со другими службами в режиме ЕПП, например, с защищенным сервером СУБД, в конфигурационном файле виртуального хоста следует дополнительно указать:

```
KrbSaveCredentials on
```

В настройках браузера Mozilla Firefox необходимо задать в качестве значений параметра `network.negotiate-auth.delegation-uris`, маски доменов которым можно передавать данные для сквозной аутентификации. В запускаемых скриптах необходимо выставить переменную окружения `KRB5CCNAME`. Например, для скрипта на языке PHP установка значения переменной окружения будет иметь следующий вид:

```
putenv("KRB5CCNAME=" . $_SERVER['KRB5CCNAME'] );
```

7. ИНТЕГРАЦИЯ ОПЕРАЦИОННЫХ СИСТЕМ СЕМЕЙСТВА LINUX С ОС СН

Анализ возможностей совместного применения ОС СН и других операционных систем семейства Linux в автоматизированных системах в защищенном исполнении показал, что для успешного решения данной задачи необходимо:

- обеспечить совместимость при сетевом взаимодействии в условиях мандатного разграничения доступа;
- обеспечивать совместимость с единым пространством пользователей (ЕПП) в ОС СН;
- обеспечить совместимость с системой централизованного протоколирования ОС СН.

Для обеспечения совместимости при сетевом взаимодействии в условиях мандатного разграничения доступа необходимо учесть, что в ОС СН реализована передача (в поле опций каждого пакета IPv4) классификационной метки сформировавшего его процесса (субъекта доступа) в формате, реализованном в соответствии со стандартом RFC1108. Использование данного стандарта обеспечивает в том числе совместимость с ОС семейства МСВС. Подробная информация приведена в 3.2.1.

7.1. Интеграция с ЕПП

Существует возможность интеграции операционных систем семейства Linux с ЕПП ОС СН. Для интеграции с ЕПП необходимо реализовать в операционной системе семейства Linux определенный порядок настройки и функционирования ряда программных средств.

Для этого необходимо понимать, что ЕПП в ОС СН представляет собой средства организации работы пользователя в сети компьютеров, работающих под управлением ОС СН. В основу положен доменный принцип построения сети, подразумевающий объединение в одну сеть логически связанных компьютеров, например, принадлежащих одной автоматизированной системе. При этом пользователь получает возможность для работы с разделяемыми сетевыми ресурсами и взаимодействия с другими пользователями.

Организация ЕПП предоставляет следующие возможности:

- сквозная аутентификация в сети;
- централизация хранения информации об окружении пользователей;
- централизация хранения настроек системы защиты информации на сервере.

Сетевая аутентификация и централизация хранения информации об окружении пользователя подразумевает использование двух основных механизмов: NSS и PAM.

Сквозная доверенная аутентификация реализуется технологией Kerberos.

В качестве источника данных для базовых системных сервисов на базе механизмов NSS и PAM используется служба каталогов LDAP.

Централизация хранения информации об окружении пользователей подразумевает и возможность централизованного хранения домашних каталогов пользователей. Для этого используется сетевая защищенная файловая система (СЗФС) CIFS.

Служба Astra Linux Directory (ALD) представляет собой систему управления ЕПП. Она является надстройкой над технологиями LDAP, Kerberos 5, CIFS и обеспечивает автоматическую настройку всех необходимых файлов конфигурации служб, реализующих перечисленные технологии, а так же предоставляет интерфейс управления и администрирования.

Для решения задачи функционирования в ЕПП и компьютеров, работающих под управлением других операционных систем семейства Linux необходимо:

- 1) Обеспечить соответствующую настройку сети, при которой разрешение имен настроено таким образом, чтобы имя системы разрешалось, в первую очередь, как полное имя (например, `mysomr.example.ru`). Сервер ALD должен быть доступен по своему полному имени.
- 2) Обеспечить получение необходимой информации о пользователях от службы каталогов ALD.
- 3) Обеспечить прохождение аутентификации по Kerberos пользователей и служб системы.
- 4) Обеспечить подключение (при необходимости) домашних каталогов пользователей.

Помимо перечисленных вопросов должна быть описана и реализована процедура введения и вывода компьютера в домен ALD доступными средствами системы.

При этом необходимо понимать, что ряд специфичных для ОС СН функций ALD (ограничение входа в систему, назначение локальных групп пользователям домена, подключение домашних сетевых каталогов и т.п.) для других ОС семейства Linux может быть доступно при условии реализации специализированного PAM-модуля или посредством использования дополнительных PAM-модулей типа `pam_group`, `pam_mount` и т. п.

7.1.1. Служба каталогов

В ALD в качестве службы каталогов используется LDAP-сервер OpenLDAP. Взаимодействие с ним может осуществляться как с помощью набора утилит OpenLDAP (как правило, входящих в пакет `ldap-utils`), так и с помощью API (`libldap`).

В ALD используется разграничение доступа к данным, расположенным в службе каталогов. Общедоступные данные могут быть получены любыми аутентифицированными пользователями, часть данных может быть получена только самим пользователем, а системные данные могут быть получены или изменены только пользователями с административным доступом.

Для хранения информации в службе каталогов используются форматы, схемы данных и подходы, регламентированные следующими стандартами:

- RFC 4515 – Lightweight Directory Access Protocol (LDAP). Описание структуры запросов к службе каталогов.
- RFC 1274 – The COSINE and Internet X.500 Schema. Описание работы с атрибутами и классами.
- RFC 2307, RFC 2307 bis – An Approach for Using LDAP as a Network Information Service. Схема атрибутов и классов для хранения информации об учетных записях пользователей и другой информации, необходимы для организации домена.
- RFC 2798 – Definition of the inetOrgPerson LDAP Object Class. Схема атрибутов и классов, расширяющих класс пользователя дополнительными сведениями.

Разработчики, решающие задачу по интеграции операционной системы семейства Linux с ALD, могут получить сведения об используемой в ALD схеме LDAP, описание атрибутов, классов и порядок доступа к ним посредством развертывания стенда, включающего компьютер, под управлением ОС CH с установленным и настроенным сервером ALD. Подробная информация о ЕПП приведена в разделе «Средства организации ЕПП» документа «Операционная система специального назначения «Astra Linux Special Edition» Руководство администратора. Часть 1» РУСБ.10015-01 95 01-1)

7.1.2. NSS

Механизм NSS предоставляет всем программам и службам, функционирующим на локальном компьютере, системную информацию через соответствующие программные вызовы. Он обращается к конфигурационному файлу `/etc/nsswitch.conf`, в котором указаны источники данных для каждой из системных служб. Для работы в составе ЕПП в качестве источника данных должна быть указана служба каталогов LDAP.

В конфигурационный файл `/etc/nsswitch.conf` должны быть внесены следующие изменения (для служб `passwd`, `group`, `shadow`):

```
passwd: compat ldap
group:  compat ldap
shadow: compat ldap
```

Указанные изменения регламентируют обращение за информацией об учетных записях пользователей и групп в первую очередь в локальных файлах компьютера, затем с помощью службы каталогов.

Для возможности получения информации с помощью службы каталогов необходимо наличие расширения NSS для работы с LDAP, как правило в ОС семейства Linux, таким расширением является `nslcd` (Daemon for NSS and PAM lookups using LDAP). В конфигурационном файле `nslcd.conf` должны быть указаны следующие сведения для доступа к

службе каталогов:

```
uri ldap://<сервер каталогов>
base ou=users,<dc суффикс домена>
ldap_version 3
```

В параметре URI указывается имя сервера службы каталогов (сервера ALD), в параметре BASE указывается путь поиска учетных записей пользователей и групп (dc суффикс домена, например для домена `.example.ru`, выглядит следующим образом `dc=example,dc=ru`).

После данной настройки информация о пользователях домена станет доступной с помощью механизма NSS. Получить список всех пользователей и групп можно с помощью вызовов:

```
# getent passwd
# getent group
```

7.1.3. Аутентификация

Для сквозной аутентификации используется MIT-реализация Kerberos. На компьютере должны быть установлены соответствующие пакеты:

- `krb5-config` – конфигурационные файлы;
- `krb5-user` – клиентские программы аутентификации Kerberos;
- `libpam-krb5` – PAM модуль для аутентификации пользователей при входе в систему;

Наименования пакетов указаны для ОС Debian. Для других ОС семейства Linux наименования пакетов могут отличаться.

Для работы в домене ALD должен быть соответствующим образом настроен конфигурационный файл `/etc/krb5.conf`:

```
[libdefaults]
    default_realm = <REALM>
...
[login]
    krb5_get_tickets = true
...
[realms]
    <REALM> = {
    kdc = <сервер ALD>
}
```

Где в качестве `<REALM>` указывается область Kerberos, соответствующая домену (например, для домена `.example.ru` таким значением будет `EXAMPLE.RU`).

Поскольку в конфигурационном файле указывается дополнительная информация по

используемым криптографическим преобразованиям и т.п. целесообразно скопировать рассматриваемый конфигурационный файл непосредственно с компьютера под управлением ОС CH, входящего в тот же домен ALD.

Проверка корректности настройки может быть выполнена попыткой аутентификации от имени пользователя домена ALD с использованием следующей команды:

```
# kinit <пользователь ALD>
```

В случае успеха аутентификации будет получен TGT билет Kerberos, который можно просмотреть с помощью утилиты `klist`:

```
# klist
```

Для возможности аутентификации пользователя при входе в систему необходимо соответствующим образом настроить PAM сценарий входа с использованием PAM модуля `pam_krb5`.

Вариант PAM сценария `login`:

```
auth [success=ignore default=die] pam_tally.so per_user deny=10
auth [success=2 default=ignore] pam_unix.so
auth [success=1 default=ignore] pam_krb5.so minimum_uid=<MINUID>
auth requisite pam_deny.so
auth required pam_permit.so
```

В аргументе `minimum_uid` должен быть указан минимальный идентификатор пользователя домена ALD (по умолчанию в ОС CH используется значение 2500).

7.1.4. Домашние каталоги пользователей

Для хранения домашних каталогов пользователей в Astra Linux Directory применяется СЗФС CIFS. Без необходимой доработки модуля ядра `cifs` для работы с СЗФС CIFS, содержащей мандатные метки, монтирование домашнего каталога под уровнем конфиденциальности отличным от 0 невозможно.

Однако для любой учетной записи пользователя домена ALD можно определить использование локального каталога системы. Соответствующее значение указывается в свойствах учетной записи пользователя домена ALD. Дополнительная информация приведена в справке для утилиты `fly-admic-smc` и в руководстве `man` для утилиты `ald-admin`.

При необходимости автоматического подключения домашнего каталога с использованием CIFS при входе пользователя в систему может быть использован PAM модуль `pam_mount` (пакет `libpam-mount`):

- 1) Подсистема работы CIFS должна быть настроена согласно документации, входящей в состав целевой ОС, для работы с аутентификацией по Kerberos.
- 2) PAM модуль `pam_mount` должен быть указан в PAM-сценариях согласно своей документации.

3) Опции монтирования в конфигурационном файле `/etc/security/pam_mount.conf.xml` могут быть указаны следующим образом:

```
<cifsmount>mount.cifs //%(SERVER)/%(VOLUME) %(MNTPT) -o %(OPTIONS) </cifsmount>
<volume fstype="cifs" uid="{MINIMUM_UID}-65000" server="{SERVER}" path="{MOUNTPOINT}"
options="user=(USER),sec=krb5,rw,setuids,perm,soft,icharset=utf8"
/>
```

Заключенные в круглые скобки параметры являются внутренними параметрами `pam_mount`. Параметры, заключенные в фигурные скобки, относятся к ALD и должны быть прописаны вручную. В качестве точки монтирования каталогов под 0 уровнем используется каталог вида `/ald_home/<имя пользователя>`.

7.1.5. Администрирование

Административные действия по вводу компьютера в домен и настройке локальных служб для работы в ЕПП могут быть выполнены либо прямыми изменениями в ALD с помощью утилит `Kerberos` и `ldap` с добавляемой системы, либо путем операций в домене ALD средствами `Astra Linux` с последующими дополнительными действиями, выполняемыми на локальной системе.

Например: добавление пользователей, создание учетных записей служб и управление принадлежностью к группам могут выполняться администратором ALD, а выгрузка ключей служб и их настройка для работы с ЕПП должны выполняться на локальной системе.

Порядок введения компьютера в домен:

- 1) Должно быть настроено разрешение имен для разыменования полного имени сервера ALD и самого компьютера.
- 2) Для вводимого компьютера должна быть создана учетная запись `Kerberos` вида `host/<полное имя компьютера>` и добавлена в соответствующую ветку службы каталогов.
- 3) Ключи учетной записи должны быть выгружены средствами `Kerberos` (утилитой `ktutil`) системы в локальный файл `/etc/krb5.keytab`.

Создание ученых записей `Kerberos` для сетевых служб, функционирующих на локальном компьютере, и выгрузка их ключей выполняется аналогично учетной записи `host`. Настройка сетевых служб для работы с аутентификацией по `Kerberos` выполняется согласно документации на эти службы. Примеры создания в ALD учетных записей для сетевых служб приведены в пункте «Настройка сетевых служб» документа «Операционная система специального назначения «Astra Linux Special Edition» Руководство администратора. Часть 1» РУСБ.10015-01 95 01-1.

7.2. Интеграция с системой централизованного протоколирования ОС СН

Средства централизованного протоколирования основываются на программном пакете Zabbix. Порядок решения задачи централизованного сбора и анализа журналов регистрации событий в ОС СН описан в разделе «Средства централизованного протоколирования» документа «Операционная система специального назначения «Astra Linux Special Edition» Руководство администратора. Часть 1» РУСБ.10015-01 95 01-1. Таким образом, для взаимодействия с сервером централизованного протоколирования в ОС семейства Linux могут быть использованы стандартные реализации агентов Zabbix.

Кроме того, для ОС семейства Linux существует возможность применения службы `rsyslog` для передачи данных о регистрируемых событиях на сервер централизованного протоколирования. Пример использования службы `rsyslog` в ОС СН приведен в разделе «Средства централизованного протоколирования» документа «Операционная система специального назначения «Astra Linux Special Edition» Руководство администратора. Часть 1» РУСБ.10015-01 95 01-1.

8. РЕКОМЕНДАЦИИ ПО СПОСОБАМ МИГРАЦИИ ПРИЛОЖЕНИЙ В СРЕДУ ОС СН С ДРУГИХ ПЛАТФОРМ

8.1. Особенности переноса программ с 32-разрядных платформ на 64-разрядную

Задача переноса существующих 32-разрядных приложений в среду 64-разрядной ОС СН может решаться различными методами в зависимости от тех или иных особенностей данных приложений. В некоторых простых случаях для работы таких программ достаточно настройки режима исполнения 32-разрядных приложений (8.2). Но, в общем случае, перенос на 64-разрядную платформу потребует внимательного анализа исходных текстов программ с целью выявления особых участков программного кода, которые необходимо исправить или заново переписать (подробнее об этом в статье [22]).

Объем работ по переводу программных средств на 64-разрядную платформу ОС СН варьируется в зависимости от ответов на следующие ключевые вопросы:

- 1) В какой мере программные средства используют программный интерфейс СЗИ ОС?
- 2) Какие используются версии библиотек графического интерфейса?
- 3) Имеется ли в составе программных средств функционал, работающий в пространстве ядра ОС (например, специализированные модули ядра)?

ОС СН является развитием стандартных Linux-дистрибутивов, поэтому для переноса в ее среду программ, не взаимодействующих в процессе своей работы со встроенными СЗИ, можно использовать весь набор рекомендаций, касающийся разработки, переноса и миграции ПО с других платформ в ОС Linux. Достаточно хорошая подборка статей на эту тему предоставлена Linux-порталом IBM, как, например [23] [24]. При этом надо учесть особенности как 64-разрядной процессорной архитектуры и ее поддержки ядром Linux [5], так и новых системных и графических библиотек (см. раздел 2).

Основная системная библиотека ОС СН — `glibc` — предполагает некоторую обратную совместимость с предыдущими ее версиями в части программного интерфейса. Таким образом, многие старые программы смогут с ней корректно работать. Напротив, у графических библиотек такая обратная совместимость не наблюдается. В связи с этим в состав ОС СН помимо новой прикладной графической библиотеки Qt 5 входит и библиотека предыдущего поколения Qt 4 с поддержкой Qt 3. Более ранние версии библиотек Qt 1 и Qt 2 в ОС СН не поддерживаются.

Процесс портирования становится намного сложнее в том случае, если в состав программного средства входят специализированные модули, созданные под конкретные версии устаревших ядер операционных систем. В связи с большими различиями в архитектурах

построения новых и старых ядер разработчикам потребуется почти полная переработка этих модулей на базе нового интерфейса (достаточно полное его описание можно найти в руководстве [5]). Советы по переносу программ управления устройствами с платформы Windows в Linux можно найти в статье [27].

Если в программном средстве явно используется специфический программный интерфейс СЗИ, то необходимо учитывать особенности реализации мандатного разграничения доступа на конкретной платформе. Следует учесть, что из-за более высокой разрядности в ОС СН диапазоны мандатных уровней и категорий, а также набор специальных системных вызовов шире, чем на ОС предыдущих поколений (см. раздел 3). Для ускорения процесса перевода таких программных средств может потребоваться написание «программ-оберток» с помощью макросов С-препроцессора, которые будут отображать старый программный интерфейс СЗИ на новый.

8.2. Настройка среды исполнения 32-разрядных приложений

Процессоры с архитектурой x86-64 (AMD64) способны работать как в 64-разрядном, так и 32-разрядном режимах.

Если рассматривать небольшие 32-разрядные Linux-приложения для x86, которые могут быть статически слинкованы со всеми необходимыми библиотеками, то они с большой вероятностью будут успешно работать в среде 64-разрядной ОС СН.

Для работы тех приложений, которые используют динамически подгружаемые библиотеки, в ОС СН необходимо настроить среду выполнения 32-разрядных программ. Для этого следует установить пакет `ia32-libs`, который содержит основные системные разделяемые 32-разрядные библиотеки. В консоли суперпользователя выполнить команду:

```
% Вопрос: видимо, менять на libgtk*???
```

```
# apt-get install ia32-libs ia32-libs-gtk
```

Это создаст в корневом каталоге папку `/emul/ia32-linux` с 32-разрядными библиотеками. По умолчанию они поддерживают только консольные приложения, базовые X- и GTK-приложения. Для Qt-приложений необходимо самостоятельно установить 32-разрядные библиотеки в каталог `/emul/ia32-linux`. Также надо поступать и с другими недостающими 32-разрядными пакетами. Сделать это можно, например, так:

```
# sudo dpkg -X <имя_пакета>-<версия>_i386.deb /emul/ia32-linux
```

8.3. Перенос программ из среды Windows

При переносе программных средств автоматизированных комплексов из ОС Windows на платформу ОС СН можно руководствоваться общими подходами, разработанными по миграции в среду Linux. Хорошее и последовательное описание технических решений и методов, необходимых для успешного перехода пользователей на Linux-систему можно про-

читать в статьях [25] [26]. Также следует ознакомиться с рекомендациями в статье [27] при переносе программ управления устройствами, если таковые имеются в системе. Особенности компиляции программ, разработанных на 32-разрядной платформе Windows, приведены в 8.4.

8.4. Технология компиляции при переносе ПО

При переносе ПО может потребоваться перекомпиляция 32-разрядных программ в том виде, как они написаны, без адаптации к 64-разрядной системе. Далее эти программы можно будет запускать на ОС СН при правильно настроенной 32-разрядной среде исполнения (см. 8.2). Чтобы компиляция в ОС СН проходила корректно и давала на выходе 32-разрядные исполняемые приложения, необходимо использовать для компилятора `gcc` — ключ `-m32`, для линковщика `ld` — ключ `-melf_i386`. Например, если сборка программы осуществлялась с помощью средств Autotools, то могут применяться следующие опции для `configure`:

```
./configure --target=i386-linux --cc="gcc -m32" --as="as --32" \  
--with-extralibdir=/usr/lib32
```

В качестве значений параметра `--with-extralibdir=` необходимо указывать каталоги, где расположены требуемые для сборки приложения 32-разрядные библиотеки.

В случае, если перенос осуществляется из среды Windows, тогда компиляция требует более сложной настройки. Обзор приемов такой компиляции можно найти в статье [28].

9. МОБИЛЬНАЯ СЕССИЯ

При разработке ПО ориентированного на мобильные устройства, рекомендуется применять технологии Qt/QML. Чтобы использовать мобильный стиль Fly нужно взять за основу приложения библиотеку компонентов `fly-qml-components`.

9.1. Пакет `fly-qml-components`

Библиотека `fly-qml-components` представляет собой набор готовых QML компонентов, которые вы можете использовать в своем приложении. Полный набор компонентов находится тут: `/usr/lib/x86_64-linux-gnu/qt5/qml/Fly/Components`. Для подключения библиотеки необходимо в `.qml` файле указать: `import Fly.Components 1.0`. Базовый компонент — `FlyRoot`, основанный на `QQuickWindow`. Самому создавать окно не нужно, достаточно `QQmlApplicationEngine`. В основе лежит страничное отображение. Для добавления страницы (`FlyPage`) используется функция `root.addPage(<pathToPage>, {properties})\verb`. При нажатии на «х», кнопку «Отмена» или кнопку «Назад» страница автоматически закроется и вызовет `cancelFunction`. При нажатии на «ОК» вызовется `okFunction`. В базовом компоненте есть функции для создания и отображения некоторых основных виджетов с заданными параметрами. Можно отобразить виджеты сообщения, списка, выбора числа, календаря, строки ввода, выбора цвета. Например, следующий код отобразит страницу сообщения:

```
root.showMessagePage({
  "title": qsTr("Warning"),
  "message": qsTr("Hello!"),
  "showCancelButton": true,
  "okFunction": function() { /*do something*/ }
})
```

Основные цвета содержатся в компоненте `FlyStyle`, который доступен по ссылке `styles`. Например:

```
Rectangle { color: styles.colors.white }
```

При указании размеров элементов желательно отталкиваться от `root.itemHeight`. Для отображения изображения из темы (например, логотипа) используется следующая запись:

```
Image { source: "image://ThemeImageProvider/astra" }.
```

9.2. Структура приложения

Базовая структура приложения следующая: Файл `main.cpp`:

```
QQmlApplicationEngine engine;
engine.load(QUrl(QStringLiteral("qrc:///main.qml")));
```

Файл `main.qml`:

```

import QtQuick 2.3
import Fly.Components 1.0

FlyRoot {
id: root
minimumWidth: 600; minimumHeight: 600
title: qsTr("Application")
icon: "my-application"
initialPage: Component { MainPage { focus: true } }
}

```

Файл MainPage.qml:

```

import QtQuick 2.3
import Fly.Components 1.0

FlyPage {
id: mainPage
frontRect.visible: false
centerRect.visible: false
backRect.color: styles.colors.white

FlyShadow { target: toolbar; horizontalOffset: 0; verticalOffset: 1 * dpiRatio; z: 0 }
Rectangle {
id: toolbar
anchors { left: parent.left; top: parent.top; right: parent.right }
height: root.itemHeight
color: styles.page.titleColor
z: 1
/* toolbar content */
}

Item {
anchors { left: parent.left; top: toolbar.bottom; right: parent.right; bottom: parent.bottom }
/* main content */
}
}

```


9.3. Клавиша «Назад»

При нажатии на клавишу «Назад» (левая кнопка в нижней панели) генерируется нажатие Alt+F4. Соответственно чтобы отловить нажатие на эту кнопку нужно обработать событие закрытия окна. При использовании `fly-qml-components`, открытые страницы(`FlyPage`) будут автоматически закрываться при нажатии на кнопку «Назад». При этом будет вызвана функция `cancelFunc`. Есть возможность добавления своей команды. Например, следующий код выведет в консоль «Hello»:

```
Item {
function processBackClick() {
root.popFunctionFromBackStack();
console.log("Hello");
}
```

```
Component.onCompleted: root.addFunctionToBackStack(processBackClick);
}
```

При использовании функции `addFunctionToBackStack` в обработке нужно самостоятельно вызывать `root.popFunctionFromBackStack()` или `root.clearBackStack()`, в зависимости от логики приложения.

9.4. Виртуальная клавиатура

При установленном пакете `qtvirtualkeyboard` виртуальная клавиатура будет автоматически показана при фокусе на текстовом поле в приложениях на базе Qt. Есть 2 способа создания клавиатуры: встроенный в приложение и внешний. При использовании `fly-qml-components` виртуальная клавиатура встраивается в приложение и плавно выезжает при отображении. При этом она поджимает только самую верхнюю страницу. Для доступа к клавиатуре используйте `root.virtualKeyboard`. Например, чтобы установить английскую раскладку можно вызвать `root.virtualKeyboard.setEnglishLayout()`. При использовании горизонтальной ориентации экрана, если высота клавиатуры будет больше 70% высоты экрана, то на пустом месте будет показано отдельное текстовое поле ввода. В приложениях без использования `fly-qml-components`, клавиатура отображается вне приложения поджимая его снизу (без анимации). В коде приложения дополнительно ничего писать не нужно. Обратиться к ней можно только через `Qt.inputMethod`. Используйте `Qt.inputMethod.visible` чтобы узнать отображается ли клавиатура. Используйте флаги `inputMethodHints` чтобы настроить клавиатуру. Более подробно можно почитать тут: <http://doc.qt.io/QtVirtualKeyboard/deployment-guide.html>

9.5. Основные компоненты fly-qml-components

Описание основных компонентов:

- FlyButton — стандартная кнопка. Используется в основном в различных формах.
- FlyCloseButton — кнопка закрытия форм.
- FlyControlButton — круглая кнопка с иконкой и текстом. Используется в панели управления.
- FlyMenuButton — слева иконка, справа текст. Можно указать только иконку или текст. Используется в тулбарах.
- FlyColoredImage — изображение с возможностью затенения, наложения цвета, обесцвечивания, добавления мини изображения в угол.
- FlyColorPicker — виджет выбора цвета.
- FlyCalendar — виджет календаря.
- FlyLabel — текст.
- FlyLineEdit — виджет однострочного поля ввода текста.
- FlyTextEdit — виджет многострочного поля ввода текста.
- FlyListView — виджет вертикального списка. В качестве модели должен быть использован массив. Поддерживает удаление элементов.
- FlyListViewRowFlick — виджет горизонтального списка. Используется в панели уведомлений.
- FlySlider — виджет горизонтального слайдера.
- FlySpinBox — виджет выбора числа.
- FlyScrollBar — используется для отображения полос прокруток.
- FlySeparator — вертикальная/горизонтальная линия.
- FlyInnerShadow — внутренняя тень. Должна быть поверх элемента.
- FlyShadow — внешняя тень. Должна быть под элементом.
- FlyTouchAnimation — анимация прикосновения к экрану.
- FlyWindow — используется при необходимости создания нескольких внутренних окон в одном приложении. Для каждого окна создается отдельная постраничная структура с поддержкой виртуальной клавиатуры.

9.6. Добавление пользовательского виджета в fly-launcher

Вы можете создать любой QML виджет. Он может быть основан на fly-qml-components. Точка входа это .qml файл. Вы можете посмотреть примеры в пакете fly-phone-widgets (/usr/lib/x86_64-linux-gnu/qt5/qml/Fly/Widgets). Будьте осторожны с Flickable и сложной обработкой мыши. На текущий момент MouseArea находится поверх виджета и пропускает только составные события. URI должен быть

Fly.Widgets.MyWidget, где MyWidget имя виджета. Простейший виджет календаря выглядит так:

```
import QtQuick 2.3
import Fly.Components 1.0
```

```
FlyCalendar {
```

Структура виджета (файлы .pro, qmlDir, _plugin опущены):

- MyWidget — Папка проекта. Последний пункт URI и имя папки должны совпадать.
- MyWidget.qml — Точка входа. Имя должно совпадать с последним пунктом в URI или быть указано в config файле.
- MyWidget.png — Изображение предпросмотра. Имя должно совпадать с последним пунктом в URI или быть указано в config файле.
- MyWidget_ru.qm — Файл перевода. Имя должно совпадать с последним пунктом в URI или быть указано в config файле.
- config — Конфигурационный файл. Имя: 'config'.
- Settings.qml — Файл настроек. Видимые пользователю настройки виджета. Любое имя.

9.7. Пакет fly-phone-dbus

Данный пакет предоставляет несколько классов для работы с различными событиями. Dbus интерфейсы находятся в /usr/include/fly-phone/fly-phone-dbus/dbus.

Для подключения используйте:

```
CONFIG += link_pkgconfig
PKGCONFIG += fly-phone-dbus
```

Например, для того чтобы позвонить по номеру 123456789:

```
import QtQuick 2.3
import Fly.Components 1.0
```

```
Item {
    FlyCallDbusNotifier { id: callDbusNotifier }
```

```
Component.onCompleted: callDbusNotifier.emitDial(123456789);
}
```

СПИСОК ЛИТЕРАТУРЫ

Общие вопросы построения операционной системы Linux

- [1] Руководящие указания по конструированию прикладного программного обеспечения для операционной системы общего назначения «Astra Linux Common Edition», ОАО «НПО РусБИТех», 2010.
(<ftp://astra-linux.ru/docs/Ruk-Devel-AstraLinuxCE.pdf>)
- [2] РУСБ.10015-01 97 01-1 «Операционная система специального назначения «Astra Linux Special Edition». Руководство по КСЗ. Часть 1»
- [3] Осаму Аоки: Справочник по Debian
(<http://qref.sourceforge.net/Debian/reference/index.ru.html>)
- [4] Руководство по Debian Policy
(<http://www.debian.org/doc/debian-policy/>)
- [5] Д.Бовет, М.Чезати: Ядро Linux (3-е издание); БХВ-Петербург, 2007.

Стандарты открытого рабочего стола Freedesktop.org

- [6] Desktop Entry Standard
(<http://standards.freedesktop.org/desktop-entry-spec>)
- [7] Icon Theme Specifications и Icon Naming Specification
(<http://standards.freedesktop.org/icon-naming-spec>
<http://standards.freedesktop.org/icon-theme-spec>)
- [8] Shared MIME-info database
(<http://standards.freedesktop.org/shared-mime-info-spec/>)
- [9] Drag-and-Drop Protocol for the X Window System
(<http://www.freedesktop.org/wiki/Specifications/XDND>)
- [10] Extended Window Manager Hints
(<http://standards.freedesktop.org/wm-spec/wm-spec-latest.html>)
- [11] XEmbed Protocol Specifications
(<http://standards.freedesktop.org/xembed-spec/xembed-spec-latest.html>)
- [12] System Tray Protocol Specifications
(<http://standards.freedesktop.org/systemtray-spec/systemtray-spec-latest.html>)

- [13] Desktop Menu Specifications
(<http://standards.freedesktop.org/menu-spec/menu-spec-latest.html>)
- [14] Sound Theme and Naming Specifications
(<http://www.freedesktop.org/wiki/Specifications/sound-theme-spec>)
- [15] Desktop Application Autostart Specification
(<http://standards.freedesktop.org/autostart-spec/autostart-spec-latest.html>)

Книги по графической библиотеке Qt

- [16] Жасмин Бланшет, Марк Саммерфилд: QT4. Программирование GUI на C++. Изд. 2-е. Официальное руководство от Trolltech; КУДИЦ-Пресс, 2008.
- [17] Макс Шлее: Qt4. Профессиональное программирование на C++. Наиболее полное руководство; БХВ-Петербург, 2006.
- [18] Юрий Земсков: Qt 4 на примерах, БХВ-Петербург, 2008.
- [19] Daniel Molkentin: The Book of Qt4: The Art of Building Qt Applications; No Starch Press San Francisco, 2007.
- [20] Johan Thelin: Foundations of Qt Development; APress, 2007.
- [21] Alan Ezust and Paul Ezust: An introduction to Design Patterns in C++ with Qt4; Prentice Hall, 2006.

Проблемы переноса программ на платформу Astra Linux Special Edition

- [22] Анализ уязвимостей при переходе на 64-х разрядные системы
(<http://www.viva64.com/ru/a/0046/#ID0ER3AE>)
- [23] Общие вопросы переноса приложений между UNIX-платформами
(<http://www.ibm.com/developerworks/ru/library/multiunix-i/>)
- [24] Приемы переноса Linux-приложений на 64-х разрядные системы
(<http://www.ibm.com/developerworks/ru/library/l-port64/index.html>)
- [25] Руководство по миграции на Linux. Часть II
(http://www.ibm.com/developerworks/ru/library/linux_migr/part2.html)
- [26] Руководство по миграции на Linux. Часть III
(http://www.ibm.com/developerworks/ru/library/linux_migr/part3.html)

- [27] Перенос приложений управления устройствами с Windows на Linux
(<http://www.ibm.com/developerworks/ru/library/l-devctrl-migration/index.html>)
- [28] Перенос программ из Windows в UNIX
(<http://www.ibm.com/developerworks/ru/library/au-porting2/index.html>)