ООО «Юниконтроллерз»  Unicontrollers Ltd.

Программа управления домашней автоматикой

FFazenda

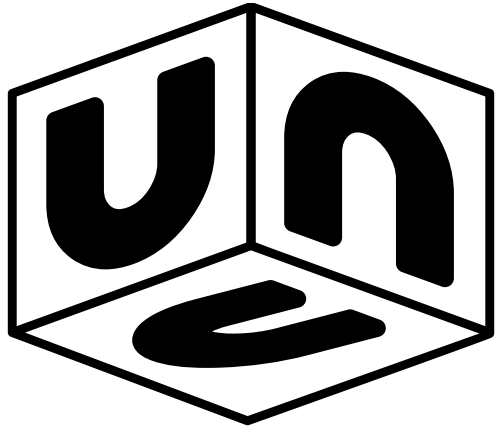
версия 1.65

руководство по эксплуатации

Москва — 2016

УДК 681.5
С81

Столяров А. В.
С81 **Программа управления домашней автоматикой FFazenda (версия 1.65): руководство по эксплуатации.** — М.:Юниконтроллерз, 2016 — 37 с.



Оглавление

I. Общее описание

Программа управления домашней автоматикой FFazenda (далее — «программа») предназначена для управления любительскими комплексами домашней автоматики «Умный дом»/«Умная дача» и расчитана на работу с универсальными контроллерами серий UNC001 и UNC01x, выпускаемыми ООО «Юникон-троллерз», но может при необходимости быть адаптирована к взаимодействию с другими аппаратными реализациями систем умного дома/дачи.

Основные возможности программы:

- гибкое описание логики работы системы домашней автоматизации в терминах конечных автоматов и событий;
- приём и отправка SMS-сообщений (управление и оповещение через SMS);
- локальное управление через Web-интерфейс;
- локальное управление через консольный порт (telnet);
- описание пользовательских команд;
- взаимодействие с другими системами через запуск внешних команд и анализ внешних источников событий.

В настоящее время существует только версия программы для ОС Linux, работа которой проверяется на обычных PC-совместимых компьютерах, а также на Raspberry Pi. Имеются основания предполагать, что программа будет работать также в других операционных системах семейства Unix, таких как FreeBSD. Программа распространяется в виде архива исходных текстов на языке C++, поставляется бесплатно под лицензией GNU GPL v.3. В будущем возможен выпуск версии программы для Windows, если будет обнаружен спрос на такую версию; отметим на всякий случай, что использование Windows на компьютере, управляющем домашней автоматикой (и, как следствие, постоянно включённом), на наш взгляд, является идеей неудачной, как и вообще использование в этой роли компьютера общего назначения (например, настольного).

Программа имеет **встроенный web-сервер**, так что для её работы не требуется настраивать серверное программное обеспечение для web.

Для приёма и отправки SMS-сообщений программа взаимодействует со свободно распространяемым (и входящим в большинство дистрибутивов Linux) пакетом SMSTools.

Для компиляции программы в системе должны быть установлены:

- архиватор `tar` и компрессор `gzip`, либо другое программное обеспечение, позволяющее распаковать архив в формате `.tar.gz`;
- компилятор `gcc` с поддержкой языка C++;
- утилита автоматической сборки GNU Make;

- библиотека `libusb` с заголовочными файлами (пакет `libusb-dev`).

Всё остальное, что необходимо для сборки, включая используемые библиотеки, содержится в поставляемом архиве исходных текстов; библиотеки, естественно, предствлены в виде исходных текстов.

В откомпилированном виде программа представляет собой единый исполняемый файл; для её запуска необходимо создать файл конфигурации. Во время работы программа записывает информацию о происходящем в файл журнала, а информацию о текущем состоянии сохраняет в файле статуса; имена этих файлов задаются ы файле конфигурации, имя самого файла конфигурации может быть задано в командной строке при запуске программы. Никаких других файлов для работы программы не требуется. Программа не требует инсталляции, не прописывается в настройки системы, и т. д., хотя пользователь волен сделать необходимые настройки сам (например, с целью автоматического запуска программы при старте системы).

При работе программа требует прав суперпользователя (пользователя `root`) для (1) получения порта web-сервера и (2) для обращения к устройствам UNC0xx через `libusb`. Программу можно запускать с правами обычного пользователя, если при этом вместо стандартного порта № 80 использовать порт с номером, превосходящим 1024, например порт 8080 или 8888, а доступ к устройствам UNC0xx разрешить непривилегированному пользователю, например, с помощью утилиты `unc_chown` из стандартного комплекта поставки программного обеспечения к устройствам UNC0xx (см. руководство по эксплуатации вашего устройства). Следует отметить, что утилита `unc_chown` не входила в комплект ПО версий до 3140909 включительно и не описана в соответствующих руководствах, однако вы можете взять её из более нового комплекта, доступного на сайте ООО «Юниконтроллерз».

II. Используемые концепции

Программа широко использует всевозможные именованные сущности: именами снабжаются события, состояния, управляемые цепи, датчики и т. п. В качестве таких имён используются слова, называемые ***идентификаторами***.

Напомним, что ***идентификатором*** в программировании называется последовательность букв и цифр, используемая для именования той или иной сущности (переменной, константы, подпрограммы и т. п.). Программа FFazenda позволяет в качестве идентификаторов использовать имена, состоящие из латинских букв, арабских цифр и знака подчёркивания, при этом идентификатор не должен начинаться с цифры. Например, допустимыми идентификаторами будут: `start`, `long_pause`, `room722`, `grant4all`, `k9_`, `_quit` и т. п. В некоторых случаях допускается использование также символов «+» и «-». Кроме того, программа FFazenda использует несколько встроенных (предопределённых) идентификаторов, имена которых начинаются со специальных знаков: `@ACT+`, `!ready` и т. п.

2.1. Состояния, события и оповещения

Под ***конечным автоматом***, как известно, понимается абстрактный объект, который в каждый момент находится в одном из предопределённых ***состояний*** (множество которых конечно, отсюда название), может изменить своё состояние при наступлении тех или иных ***событий***, при этом (при смене состояния) может также выполнить некие действия.

Программа FFazenda реализует один или несколько конечных автоматов. Множества состояний для этих автоматов задаёт пользователь в конфигурационном файле. Один из конечных автоматов считается основным, остальные — дополнительными. Состояния основного конечного автомата задаются простыми ***идентификаторами***. Дополнительные конечные автоматы имеют свои имена, основной конечный автомат имени не имеет; имена дополнительных конечных автоматов также задаются простыми идентификаторами, а имена их состояний состоят из двух идентификаторов — имени автомата и имени состояния,— разделённых точками. Например, основной автомат может иметь состояния `standby`, `nobodyhome`, `security`, `alert` и т. п., а дополнительный автомат с именем `power` — состояния `power.ok`, `power.failing` и `power.longfailure`.

Программа запоминает (записывает в файл статуса) список состояний при каждой их смене; это позволяет при перезапуске программы начать работу в том же состоянии, в котором программа находилась на момент прекращения её работы (например, на момент выключения компьютера).

Программа поддерживает работу со свето-звуковым оповещателем (в просторечии — «лампочка с сиреной»), позволяя настроить в зависимости от текущего состояния различные режимы горения/мигания лампочки и постоянного либо прерывистого звучания сигнала. Предполагается, что именно свето-звуковой

оповещатель будет основным средством для пользователя узнать, в каком состоянии находится программа в настоящий момент.

Для каждого состояния в конфигурационном файле можно задать:

- режим работы свето-звукового оповещателя: лампочку оповещателя можно погасить, зажечь или заставить мигать, причём при мигании можно задать длительности для фаз включения и выключения; звук можно отключить, включить, заставить звучать синхронно с миганием лампочки или в противофазе к таковому;

- отправку сообщений пользователю при переходе программы в данное состояние; в нынешней версии единственным видом таких сообщений являются SMS;

- автоматический переход в другое состояние и/или событие, наступающее по прошествии определённого периода времени;

- переход в другое состояние вместо данного в случае, если в настоящее время имеются причины, не позволяющие поставить помещение на охрану (удобно для состояний, соответствующих режиму «объект поставлен на охрану», так как позволяет дождаться готовности к постановке на охрану и только после этого сменить режим).

Отметим, что обычно режимы свето-звукового оповещателя задают только для состояний основного конечного автомата, а дополнительные конечные автоматы на оповещатель не влияют, хотя программа позволяет при желании установить такое влияние.

Состояния конечного автомата могут меняться при наступлении ***событий***, которые также задаёт пользователь в конфигурационном файле. Источниками событий могут быть:

- команды пользователя, переданные через SMS, Web-интерфейс или управляющую консоль;

- замыкание и размыкание входных линий устройств UNC0xx, с которыми могут быть связаны датчики движения, датчики открытия дверей и окон, а также установленные в различных местах вашего дома кнопки, тумблеры и герконы;

- данные с датчиков температуры, подключённых через устройства UNC0xx; точнее — события возникают, если температура, передаваемая таким датчиком, становится ниже заданной нижней границы или выше заданной верхней границы, указанных в конфигурационном файле для этого датчика, причём таких пар границ может быть несколько для одного датчика.

Событие имеет имя (идентификатор) и ***параметры***, которые можно использовать для передачи дополнительной информации о наступившем событии; например, в системе может произойти событие «`temperature cellar freezes`»,

где `temperature` — имя события, а `cellar freezes` — его параметры; всё вместе будет свидетельствовать о том, что, согласно информации с датчиков температуры, в подвале здания температура опустилась ниже установленной границы.

Для каждого события, определённого в конфигурационном файле, можно задать:

- смену состояния одного или нескольких конечных автоматов в зависимости от их текущих состояний;
- выполнение одного из предопределённых ***действий***, в качестве которых в нынешней версии поддерживаются: включение управляемой цепи одного из подключённых устройств UNC0xx, выключение такой цепи, а также пополнение и сокращение списка действующих в настоящий момент причин, препятствующих постановке помещения на охрану (например, открытые двери, сработавшие датчики движения и т. п.);
- отправку сообщения пользователю; в нынешней версии единственным видом такого сообщения является SMS;
- выполнение внешней команды, т. е. запуск другой программы из числа имеющихся в системе на управляющем компьютере.

Все конфигурационные характеристики события — таблицу смены состояний, выполняемое действие, отправку сообщения (***оповещение***) и выполнение команды — можно поставить в зависимость от текущего состояния одного из имеющихся конечных автоматов. Кроме того, действие, оповещение и команда могут, как и событие, иметь параметры, для формирования которых можно использовать параметры события.

Программа поддерживает два ***встроенных события***, связанных со списком причин, препятствующих постановке помещения на охрану; событие `!activity` возникает, когда в пустой список вносится идентификатор причины, событие `!ready` возникает, когда из списка удаляется последний идентификатор и список, таким образом, становится пустым.

Рассылка сообщений о происшедших событиях (оповещение) также настраивается в конфигурационном файле. Программа позволяет описать произвольное количество «оповещений» (нотификаций), для каждого из которых задаётся текст оповещения и список номеров телефонов, на которые это оповещение должно быть отправлено в виде SMS-сообщения.

2.2. Взаимодействие с устройствами UNC0xx

Программа расчитана на работу с произвольным количеством устройств UNC0xx; единственное ограничение обусловлено возможностями USB и формально составляет 128 устройств, но в реальности сложно себе представить разветвлённую систему USB-концентраторов, охватывающую такое количество устройств при соблюдении максимально допустимой общей длины шины в пять

метров; можно считать, таким образом, что программа справится с обслуживанием любого количества устройств UNX0xx, какое только удастся корректно физически подключить к управляющему компьютеру.

Программа различает контроллеры UNC0xx по ***пользовательской части серийного номера***, для установки которой можно использовать команду `uncctl -T`. Подробности можно узнать из руководств по эксплуатации соответствующих устройств.

Для каждого контроллера можно в конфигурационном файле программы задать имена (идентификаторы) используемых входных и линий и управляемых цепей, события, связанные с замыканием и размыканием входных линий, а также указать на необходимость опроса контроллера на предмет статуса подключённых к нему устройств 1-Wire (датчиков температуры и ключей-«таблеток»).

Если для данного контроллера UNC0xx задана хотя бы одна входная линия и/или включён опрос 1-Wire, программа будет ежесекундно опрашивать такой контроллер на предмет происшедших изменений.

2.3. Пользовательские команды

Пользовательские команды используются в управляющих SMS-сообщениях, а также при работе с управляющей консолью и в настройках web-интерфейса.

Программа обрабатывает строки, содержащие одну или несколько команд. Каждая команда может требовать или не требовать указания параметров, причём количество таких параметров может быть фиксированным или переменным; в последнем случае в качестве параметров рассматривается весь остаток обрабатываемой строки. Всё это позволяет задать сразу несколько команд, например, в одном SMS-сообщении.

Большинство команд может быть исполнено, только если точно известно, что они получены из доверенного источника. Некоторые источники могут считаться доверенными *á priori* (так, исходно доверенной может считаться управляющая консоль при условии, что соединение с ней установлено с локальной машины), но в большинстве случаев для определения полномочий источника команд необходимо указать пароль. Нынешняя версия программы использует пароли трёх видов: главный пароль (master password), частные пароли, привязанные к номеру мобильного телефона при управлении через SMS, а также пары «имя–пароль» при управлении через web-интерфейс; к командам имеют отношение только первые два, а веб-интерфейс после успешной авторизации обычно считается доверенным источником команд. Пароль указывается с помощью отдельной команды, которая в командной строке обычно идёт самой первой. Главный пароль отличается от частных тем, что его можно использовать вне зависимости от того, через какой канал подаётся команда: например, можно прислать SMS с постороннего (не указанного в конфигурации) мобильного телефона. Рекомендуется оставить этот способ для самых крайних случаев, а в повседневной работе главный пароль не применять.

Выполнение команды может предусматривать ***ответ***, причём при работе через управляющую консоль или web-интерфейс ответ показывается пользователю в любом случае, тогда как ответ на SMS отправляется только в случае, если это в явном виде затребовано (либо в описании одной из выполненных команд, либо специальной командой в составе командной строки).

Программа поддерживает пять ***встроенных команд***, имена которых начинаются с символа «&»:

- `&M` — указание главного пароля (от слова «Master»);
- `&P` — указание частного пароля (от слова «Password»);
- `&A` — затребовать ответ на команду, даже если в обычных условиях таковой не предусмотрен («Answer»);
- `&R` — затребовать полный отчёт о состоянии системы («Report»);
- `&E` — завершить интерпретацию команд («End»); всё, что встретится в командной строке после этой команды, будет проигнорировано. Это может потребоваться, например, при отправке SMS через систему, которая добавляет рекламу в конец сообщения.

В конфигурационном файле можно описать ***пользовательские команды***, именами которых будут обычные идентификаторы. Учитывая, что эти команды придётся посылать в SMS-сообщениях, целесообразно сделать идентификаторы короткими, а для наиболее часто употребляемых команд применить идентификаторы, состоящие из одной буквы.

Пользовательскую команду можно описать одним из следующих способов:

- как синоним другой команды, встроенной или пользовательской;
- как команду-событие (то есть команду, выполнение которой считается возникновением ***события***, см. § 2.1);
- как команду-действие (то есть команду, выполняющую одно из предопределённых ***действий***, см. стр. 8);
- как команду-запрос, выполнение которой состоит в генерации определённого текста, отправляемого пользователю.

Для всех видов команд, кроме команд-синонимов, необходимо указать количество используемых параметров. Кроме того, для команды любого вида (в том числе и для команды-синонима) можно указать шаблон ответа пользователю; команда-запрос отличается от других тем, что для неё только этот шаблон и указывается.

Если при выполнении команды-синонима программа обнаруживает, что шаблон ответа указан как для синонима, так и для основной команды, используется шаблон, указанный для синонима, что позволяет легко описывать новые команды, отличающиеся от существующих только шаблоном ответа.

2.4. Готовность к постановке на охрану

Программа поддерживает список причин, препятствующих переходу в режим охраны. Такими причинами могут быть показания соответствующих датчиков, свидетельствующие о наличии движения в помещениях, об открытых окнах и дверях и т. п. Список состоит из ***идентификаторов***, внесение которых в список и исключение их из списка производится с помощью ***действий***; соответствущее действие можно предусмотреть как реакцию на ***событие*** или ***команду***.

Когда в пустой список вносится первый идентификатор, программа генерирует ***встроенное событие*** с именем «`!activity`»; когда из списка удаляется последний идентификатор и список, таким образом, становится пустым, генерируется событие «`!ready`».

2.5. Мониторинг температуры

Для мониторинга температуры с помощью датчиков, подключённых шиной 1-Wire через устройства UNC0xx, предусмотрены две основных возможности:

- показания датчика можно *включить в отчёт о состоянии системы*;
- для каждого датчика можно задать одну или несколько пар нижних и верхних границ тепературы, при переходе через которые генерируется событие; при возникновении такого события можно произвести те или иные действия (например, включить или выключить отопительные или вентиляционные приспособления), а также, например, разослать SMS по заданному списку телефонных номеров.

Для каждой пары температурных границ, заданных в конфигурации, программа помнит ***текущее положение***, которое может быть:

- неопределённым (undef), если с указанного датчика температуры ещё ни разу не приходило показаний;
- низким (low), если, согласно показаниям датчика, температура опустилась ниже нижней границы и после этого ещё ни разу не проходила через верхнюю границу (переход температуры выше *нижней* границы не изменяет это положение);
- высоким (high), если температура поднялась выше верхней границы и после этого ещё ни разу не проходила через нижнюю границу (переход ниже *верхней* границы положения не изменяет);
- средним (medium), если со времени начала работы с указанным датчиком температура ещё ни разу не опускалась ниже нижней границы и не поднималась выше верхней.

При каждом изменении текущего положения по любой из сконфигурированных пар температурных границ программа записывает новое положение в файл статуса, а пи запуске — прочитывает этот файл, что позволяет сохранять информацию о текущих положениях при перезапусках программы.

Как можно заметить, положения «неопределённое» и «среднее» используются только при начале активной работы с датчиком; программа никогда не возвращается в эти положения, если хотя бы один раз произошел переход в положение «низкое» или «высокое». В ходе дальнейшей работы эти два положения сменяют друг друга, причём для такой смены температура должна, вообще говоря, преодолеть всё расстояние между нижней и верхней границами. Это позволяет исключить «дёргания» при колебаниях температуры в окрестностях одной из границ. Например, для обычного помещения можно установить границы на уровне 1^oC и 4^oC и при наступлении соответствующих событий отправлять SMS-сообщения: если температура упала ниже 1^oC, возникает угроза промерзания помещения, и её следует оперативно устранить, но если температура поднялась выше 4^oC, можно считать, что угроза миновала. При этом температура может сделать несколько скачков вниз-вверх в окрестностях 1^oC, но SMS-сообщение будет послано только одно.

Возможность задания нескольких пар границ для одного датчика позволяет предусмотреть *для того же самого помещения*, например, события при нагревании свыше 37^oC (каковое может вывести из строя технику) и падении температуры обратно до 30^oC, и так далее.

2.6. Встроенный web-сервер

Для пользовательского управления системой в программе предусмотрен встроенный сервер протокола HTTP (web-сервер). Реализованное подмножество протокола HTTP/1.1 включает обработку запросов GET и POST, а также механизм Digest Authentication, что позволяет, во-первых, защитить паролями элементы web-интерфейса, и, во-вторых, объединить запросы от одного и того же пользователя в сеанс работы без использования cookies.

Конфигурационный файл позволяет описать произвольное количество «web-документов», которые будет обрабатывать встроенный сервер; содержимое каждой страницы может быть задано в виде текста (HTML или простого) непосредственно в конфигурационном файле, либо в виде внешнего файла, содержимое которого должно быть передано клиенту. Для каждой страницы отдельно можно задать тип данных (mime-type).

Некоторые страницы можно объявить публичными (не требующими аутентификации), другие — закрытыми, доступ к которым осуществляется через ввод пароля. Одну или несколько закрытых страниц можно сделать «командными»; такие страницы позволяют обращение к себе запросом типа POST, причём переданное тело запроса рассматривается как командная строка, подлежащая выполнению — именно этим способом производится управление системой.

III. Конфигурационный файл

Все настройки и параметры работы программы задаются в ***конфигурационном файле***, который представляет собой обычный текстовый файл и редактируется любым редактором текстов.

3.1. Синтаксис и структура конфигурационного файла

Формат файла несколько напоминает классические ini-файлы, но имеет достаточно серьёзные отличия.

Файл состоит из ***секций***, каждая из которых имеет ***заголовок секции*** и ***содержимое секции***. Содержимое, в свою очередь, состоит из ***параметров***, которые имеют ***имя*** и ***значение***. Значение параметра может быть многострочным. Имя параметра может в некоторых случаях иметь ***уточняющий квалификатор***, означающий, что в отдельном случае следует использовать одно значение, а во всех остальных случаях — другое.

Заголовок секции записывается на отдельной строке и заключается в квадратные скобки; имя секции может состоять из одного слова, либо из двух слов — в этом случае секции, имеющие одинаковое первое слово имени, образуют ***группу секций***. Примеры заголовков секций:

```
[general]

[sms]

[console]

[event alarm]

[event breakin]
```

(в данном случае секции `[event alarm]` и `[event breakin]` образуют группу с именем `event`).

Параметр записывается в виде строки, обязательно начинающейся в первой позиции, то есть не имеющей пробелов в начале строки, и состоящей из имени параметра, обязательного пробела, знака «=» и значения параметра, в качестве которого используется всё содержимое строки от знака «=» до её конца. Многострочное значение можно задать, если начать следующую строку в файле с пробела или знака «+». Примеры параметров:

```
log = /var/log/ffazenda

port = 80

watermarks =
```

```
    0 : 3 :  climate freezes : climate unfreezes : 3
    3 : 15 :  climate goes low : climate goes normal : 3
    20 : 37 : : climate goes too warm : 3

greeting = Dear user,
+
+you are now connected to the FFazenda
+control console.  Please be careful.
```

При описании параметров во всех случаях игнорируются пробелы, оставленные в конце строки; в многострочных параметрах, заданных с помощью строк, начинающихся с пробела, игнорируются также пробелы в начале строки.

В имени параметра может присутствовать ***квалификатор***, отделённый двоеточием от основного имени. Например, запись

```
notification:standby = none
notification = daily
```

может означать, что в режиме `standby` значение параметра `notification` принимается равным `none`, тогда как во всех остальных режимах — равным `daily`.

Строки, начинающиеся с символов ';' и '#', считаются комментариями и игнорируются. Комментарий также можно разместить в одной строке с заголовком секции; обратите внимание, что в строке, описывающей параметр, комментарий разместить нельзя.

Пустые строки и строки, состоящие только из пробелов, игнорируются.

3.2. Подстановки значений

В тексте некоторых параметров можно использовать так называемые ***обращения к именованным значениям***, вместо которых программа подставит те или иные значения в зависимости от ситуации. Такие обращения записываются между символами '%', например: `%date%`, `%state%`, `%full_report%` и т. п. Конкретный список именованных значений зависит от того, в теле какого параметра производится подстановка. Во всех случаях, когда подстановки вообще выполняются, доступны по меньшей мере следующие именованные значения:

- `%date%` — текущие дата и время в формате YYYYMMDD HH:MM, например, `20130915 13:27`;
- `%sysname%` — имя системы, заданное параметром `name` в секции `[general]`;
- `%state%` — список из текущих состояний основного и дополнительных конечных автоматов;
- `%devices_report%` — отчёт о состоянии устройств UNC0xx;
- `%onewire_report%` — отчёт о состоянии датчиков температуры;

- `%full_report%` — полный отчёт о состоянии программы;
- `%stateof:`*NAME*`%` — текущее состояние конечного автомата с именем *NAME* (пустое имя обозначает основной конечный автомат);
- `%temp:`*NAME*`%` — текущее значение температуры, прочитанное с температурного датчика с именем *NAME* (если датчик недоступен, выдаётся пустая строка);
- `%line:`*NAME*`:`*ON*`:`*OFF*`:`*BAD*`%` — текущее состояние входной или выходной линии с именем *NAME*, параметр *ON* задаёт текст, который необходимо выдать при включённой/замкнутой линии, *OFF* — при выключенной/разомкнутой, а *BAD* выдаётся при отсутствии связи с устройством.

Кроме перечисленного, в отдельных случаях могут быть доступны также и другие именованные значения.

Для секций, описывающих сущности с параметрами (события, команды и оповещения), поддерживаются обращения к значениям параметров: `%1%` означает первый параметр, `%2%` — второй параметр и так далее, `%*%` — все параметры, `%0%` обозначает имя самого события/команды/оповещения.

3.3. Изменяемые переменные

Конфигурационный файл позволяет использовать ***переменные***, значения которых могут изменяться во время работы программы с помощью внешних команд. Переменная описывается с помощью секции `[variable ]`, которая в нынешней версии имеет всего два параметра: `enable` (значение должно быть `yes`) и `default` (задаёт значение переменной по умолчанию). Например:

```
[variable low1]
enable = yes
default = 29

[variable high1]
enable = yes
default = 31
```

Обращение к переменным производится аналогично обращению к именованным сущностям (см. § 3.2); перед именем переменной ставится вопросительный знак. Так, к переменным из примера можно обратиться следующим образом: `%?low1%`, `%?high1%`

Изменение переменных производится с помощью ***действия*** `@SET` (см. § 3.11).

3.4. Общие конфигурационные параметры

Общие параметры работы программы задаются в секции `[general]`, например:

```
[general]

name = ffazenda
log = /var/log/ffazenda
state = /var/run/ffazenda.run
password = abrakadabra
defaultstate = standby
watchdog = 0
```

Параметр `name` — имя системы, которое используется в генерируемых сообщениях и в некоторых других случаях. Параметры `log` и `state` задают имена файлов, соответственно, для журнала (`log`) и хранения текущего статуса (`state`). Параметр `password` задаёт ***главный пароль*** (см. стр. 9). Параметр `defaultstate` указывает, в каком ***состоянии*** конечного автомата (автоматов) программе следует начинать работу, если файл статуса от предыдущего запуска не найден или не существует. Наконец, параметр `watchdog` указывает, с каким интервалом (в секундах) программа должна проверять свою собственную работоспособность; в приведённом примере значение `0` отключает проверку. Значение по умолчанию — 30 (каждые 30 секунд состояние сторожевого таймера сбрасывается; если в течение удвоенного интервала времени (60 секунд) этого не произойдёт, программа аварийно завершится с сигналом `SIGALRM`).

3.5. Взаимодействие с SMSTools

Для отправки и получения SMS-сообщений программа взаимодействует с пакетом smstools, который должен быть установлен в вашей системе и сконфигурирован; подробности можно узнать из документации по smstools. Для успешного взаимодействия программе необходимо знать пути к директориям, используемым smstools, а также пути к директориям, которые будет использовать сама программа при работе с SMS-сообщениями. Все эти параметры задаются в секции `[sms]`:

```
[sms]

incoming = /var/spool/sms/incoming
outgoing = /var/spool/sms/outgoing
failed = /var/spool/sms/failed
composing = /var/spool/ffazenda/sms
proceeded = /var/spool/ffazenda/proceeded
```

Параметры `incoming`, `outgoing`, `failed` задают директории, используемые smstools для, соответственно, входящих, исходящих и неотправленных сообщений; параметры `composing` и `proceeded` задают директории, которые программе следует использовать, соответственно, для хранения формируемых сообщений и обработанных сообщений.

Все эти директории должны располагаться на одной файловой системе! Это обусловлено тем, что файлы, содержащие SMS-сообщения, перемещаются между перечисленными директориями путём их переименования, а эта операция безопасна (атомарна) только при работе в рамках одной файловой системы.

Не забудьте создать директории, указанные в параметрах `composing` и `proceeded`; остальные директории, скорее всего, уже созданы при установке smstools.

3.6. Управляющая консоль

Управляющая консоль позволяет подключиться к ней с помощью утилит `telnet` или `netcat` и давать команды программе. Настройка консоли производится в секции `[console]`, где необходимо указать:

- ip-адрес для tcp-сервера (параметр `ip`); используйте `127.0.0.1`, чтобы к консоли можно было обратиться только с локального компьютера, или `0.0.0.0`, чтобы можно было присоединиться с другого компьютера по сети; другие значения вам, скорее всего, не потребуются;

- номер порта (параметр `port`); используйте любой номер порта, не занятый другими сервисами; например, `7779` вам, скорее всего, подойдёт;

- величину таймаута (параметр `timeout`): через заданное количество секунд, если пользователь всё это время не вводил команд, соединение с консолью автоматически закрывается;

- доверенные ip-адреса (параметр `trusted`) задаются в виде «префикс/длина маски», можно перечислить несколько префиксов через запятую; если оставить значение пустым, доверенными не будут никакие ip-адреса, так что работать с консолью можно будет только указав ***главный пароль***; будьте ***крайне осторожны*** при выборе доверенных адресов, поскольку при работе с доверенного адреса программа выполняет любую команду, не спрашивая никаких паролей;

- текст приглашения консоли (параметр `greeting`) — текст, выдаваемый пользователю сразу после присоединения к консоли; можно отдельно указать параметр `greeting:trusted`, чтобы приглашение для пользователей, пришедших с доверенных ip-адресов, отличалось от недоверенного.

Приведём пример:

```
[console]

ip = 0.0.0.0
port = 7779
```

```
timeout = 60
trusted = 127.0.0.1/8, 192.168.0.5/32
greeting = ffazenda console (type empty line to quit)
+
greeting:trusted = ffazenda trusted console, be careful
+
```

(тексты приглашений сделаны многострочными, чтобы после текста вставлялся перевод строки)

3.7. Номера телефонов, допущенных к управлению через SMS

Для номеров мобильных телефонов, с которых предполагается давать команды программе, следует указать способ авторизации (в настоящее время поддерживается только авторизация по паролю, `password`) и частные пароли. Это делается группой секций `[phone NNNNNNNNNNN]`, где `NNNNNNNNNNN` — номер телефона в международном формате, но без указания лидирующего «+», например `79267900950`. Пример секции:

```
[phone 79267900950]

auth = password
password = abrakadabra
```

3.8. Конфигурация устройств UNC0xx

Входящие в систему устройства UNC0xx должны быть описаны в группе секций `[device NNNN]`, где `NNNN` — ***пользовательская часть серийного номера*** устройства. Напомним, что заводское значение пользовательской части серийного номера — `7775`, но это значение может быть изменено с помощью команды `uncctl -T`; если вы используете больше одного устройства, необходимо назначить им разные значения, чтобы можно было их различать. Подробности можно найти в руководствах по эксплуатации устройств.

Для каждого устройства следует описать задействованные управляемые линии (параметры `output:0`, `output:1`, ..., `output:11`; номерами от 0 до 3 обозначаются линии основной группы, номерами от 4 до 7 и от 8 до 11 — линии модулей UNC010/out, установленных соответственно во второй и в первый слот расширения[1]), назначив каждой из них некое имя (идентификатор). Линии, которые используются для управления свето-звуковым оповещателем, если таковой в вашей системе используется, следует указать специальные имена `^lamp` и `^sound`.

[1] Это верно для устройств с версией прошивки, начиная с 4141014; в более ранних прошивках нумерация выходных линий отличалась; соответствующую информацию вы можете найти в руководстве пользователя к своей версии устройства.

Имена управляемых линий служат параметрами для ***действий*** по их включению и выключению.

Для управляемых линий, работу с которыми предполагается осуществлять через программу FFazenda, имеет смысл установить ***режим синхронизации***, при котором программа будет периодически (в нынешней версии — каждые 7,5 секунд) проверять, соответствует ли реальное состояние линии тому, которое было установлено самой программой, и при обнаружении несоответствия — переключать линию в ожидавшееся положение. Это может быть полезно в случае, если связь с устройством UNC0xx была временно потеряна, а затем восстановлена, и за это время состояние линии изменилось; например, при временном отключении питания устройства после восстановления питания все линии окажутся выключенными, что может не соответствовать решаемой задаче. Режим синхронизации устанавливается параметром `output_sync:N = yes`, где `N` — номер линии.

Входные линии описываются параметрами `input:N`, `input_on:N` и `input_off:N`, где `N` — номер линии от 0 до 7 (номера от 0 до 3 соответствуют первому слоту расширения, номера от 4 до 7 — второму). Параметр `input` задаёт имя линии, используемое для отчётов о состоянии; параметры `input_on` и `input_off` задают имена и параметры событий, генерируемых, соответственно, при замыкании и размыкании линии.

ВНИМАНИЕ! Для входных линий устройств UNC0xx, используемых в качестве источников событий программой `FFazenda`, необходимо при настройке самого устройства установить режим увеличения счётчика по включению и выключению с соблюдением соответствия чётности счётчика состоянию линии. Это означает, что в соответствующем регистре Rn должны быть взведены (установлены в 1) биты 31, 30 и 29. Это можно сделать, например, командами

```
uncctl -R 2 70000000
uncctl -W
```

(замените 2 на номер вашей входной линии).

В случае, если устройство необходимо опрашивать на предмет информации от шины 1-Wire (то есть если к нему подключены температурные датчики и/или зонды для ключей-«таблеток»), этот факт следует указать параметром `onewire`, указав в качестве значения слово «`yes`».

Рассмотрим пример. Допустим, используется устройство UNC011, снабженное модулем UNC010/in (в первом слоте) и модулем UNC010/out (во втором слоте), и оно имеет идентификатор (пользовательскую часть серийного номера) 2500. К линиям основной группы подключены вентилятор, две осветительные лампы и цепь питания GSM-модема, к входным линиям первого слота — тумблер «присутствие», датчик движения, датчик открытия двери и кнопка отмены тревоги, к дополнительным управляемым линиям второго слота — световая и звуковая цепи оповещателя и два внешних реле, управляющие двумя отопительными приборами. При этом вентиляция и отопительные приборы могут

включаться и выключаться в обход программы `ffazenda`, тогда как питанием GSM-модема программа должна управлять только сама. Кроме того, к устройству по шине 1-Wire подключены датчики температуры. Конфигурация такого устройства может выглядеть, например, так:

```
[device 2500]

onewire = yes

output:0 = vent
output:1 = light1
output:2 = light2
output:3 = modem_power
output_sync:3 = yes

input:0 = Presence
input_on:0 = someonehome
input_off:0 = nobodyhome

input:1 = MotionSensor
input_off:1 = activity motion
input_on:1 = silence motion

input:2 = DoorOpen
input_off:2 = activity door_open
input_on:2 = silence door_open

input:3 = CancelButton
input_on:3 = cancel

output:4 = ^lamp
output:5 = ^sound
output:6 = warm1
output:7 = warm2
```

Итак, мы задали управляемым линиям имена `vent`, `light1`, `light2`, `modem_power`, `warm1` и `warm2`, указали, что линии 4 и 5 используются для управления оповещателем, также указали, что для устройства должна быть задействована функция опроса 1-Wire.

Что касается *входных* линий, то мы установили, что тубмлер присутствия будет при его включении и выключении генерировать ***события*** `someonehome` и `nobodyhome` (без параметров), кнопка отмены тревоги будет (только при её включении) генерировать событие `cancel` (также без параметров), датчик движения и открытия двери будут при размыкании генерировать событие `activity`

(для датчика движения размыкание означает наличие движения, для датчика двери — открывание двери), а при замыкании — событие `silence`. Чтобы можно было различать датчики между собой, мы снабдили эти события параметром — `motion` для датчика движения и `door_open` для датчика открывания двери.

К конфигурированию событий и реакции на них мы вернёмся в соответствующем параграфе.

3.9. Датчики температуры и связанные с ними события

Для каждого температурного датчика, подключённого к системе, необходимо задать имя (идентификатор) для включения в отчёты, адрес 1-Wire (строчка из 16 шестнадцатеричных знаков, начиная от контрольной суммы и заканчивая идентификатором типа устройства, `10` или `28` в зависимости от типа датчика), указать, нужно ли включать показания датчика в отчёты, и — опционально — задать граничные температурные значения и события, которые должны генерироваться при выходе за эти границы.

Конфигурация датчиков задаётся секциями группы `[temp_sensor NNN]`, где `NNN` — имя датчика. Конфигурация никак не зависит от того, через какое из подключённых устройств UNC0xx датчик присоединён к системе; достаточно, чтобы для этого устройства UNC0xx был включён опрос шины 1-Wire (параметр `onewire` в секции `device`). Датчик идентифицируется только своим уникальным кодом 1-Wire (параметр `id`). Пример конфигурации температурного датчика:

```
[temp_sensor kitchen]

id = 66000003de07a728
report = yes
watermarks =
  0 : 3 :  climate freezes : climate unfreezes : 3
  3 : 15 :  climate goes low : climate goes normal : 3
  20 : 37 : : climate goes too warm : 3
  21 : 25 : warm1_on : warm1_off : 3 : warm1_on : warm1_off
```

Поясним, что параметр `watermarks` задаёт пары температурных границ и генерируемые события; параметр обычно оформляют как многострочный, каждая пара границ указывается на отдельной строке. Конфигурация пары границ состоит из пяти или семи полей, разделённых двоеточием; первые два поля задают значение нижней и верхней границы (в ^oC), следующие два поля — события, которые следует сгенерировать при проходе через эти границы, и пятое поле задаёт количество опросов датчика, после которого факт перехода через температурную границу считается подтверждённым. Как показывает практика, датчики склонны иногда выдавать значения, не имеющие ничего общего с действительностью, но случается это довольно редко и практически никогда — дважды подряд, так что значение `3` установлено с некоторым запасом.

Шестое и седьмое поля, если таковые присутствуют, задают события, которые следует сгенерировать при подключении датчика (то есть в ситуации, когда температура измерена впервые, предыдущего её значения нет и, таким образом, формально нельзя констатировать переход через границу), если при этом температура соответственно ниже нижней или выше верхней границы. Также эти события, если они заданы, генерируются при старте программы, если во время предшествующей работы для данной пары границ было запомнено нахождение ниже нижней или выше верхней границы; в этом случае события генерируются раньше, чем произойдёт первое измерение температуры.

Использование конфигурации температурных границ в форме пяти или семи полей зависит от назначения этих границ. В большинстве случаев, если целью отслеживания границ является оповещение пользователя, шестое и седьмое поле задавать не нужно, тогда как если границы отслеживаются с целью автоматического включения и отключения отопления и/или вентиляции, то шестое и седьмое поля имеет смысл задать такими же, как третье и четвёртое. Всё это объясняется тем, что при перезапуске всей системы (например, после аварии питания) пользователя, возможно, следует оповестить о её перезапуске, но вряд ли нужно оповещать об изменениях температуры, если таковая с предыдущего запуска не изменилась; с другой стороны, отопление и вентиляция, скорее всего, окажутся после перезапуска отключены, и их состояние следует привести в соответствие с текущими значениями температуры.

Температурные границы можно задать с использованием ***переменных***, что позволит изменить границы во время работы с помощью внешней команды, не редактируя конфигурационный файл; например:

```
watermarks =
  %?low1% : %?high1% : warm1_on : warm1_off : 3 : warm1_on : warm1_off
```

Учтите, что само по себе изменение этих переменных не приведёт к немедленному изменению сконфигурированных границ температуры. Для вступления изменений в силу необходим перезапуск программы; например, после соответствующих ***действий*** `@SET` можно сделать действие `@RESTART`; см. пример команды `sett1` на стр. 31.

3.10. Состояния конечных автоматов

Состояния конечных автоматов описывать в явном виде не обязательно; это делается ради указания дополнительных параметров, но все эти параметры факультативны. Если идентификатор состояния указан в одной из таблиц перехода, связанных с событиями, этого уже достаточно, чтобы состояние существовало.

Если состоянию необходимо приписать дополнительные параметры, в конфигурационный файл включают описание состояния в виде секции `[state NAME]`, где `NAME` — имя состояния (идентификатор для состояний главного конечного

автомата; два идентификатора, разделённые точкой — для состояний дополнительных конечных автоматов).

Если предполагается отражать нахождение системы в данном состоянии с помощью светозвукового оповещателя, для такого состояния следует указать параметры `lamp` и `sound`. Значением параметра `lamp` могут быть слова `on`, `off` (соответственно, лампа влючена/погашена) или два целых числа, разделённых двоеточием; последнее означает, что лампа должна мигать, а числа задают длительность периода свечения и периода отключения в миллисекундах. Например,

```
lamp = on
```

означает, что при попадании в это состояние лампа светозвукового оповещателя будет непрерывно включена, а

```
lamp = 500:2000
```

означает, что лампа будет включаться на $\frac{1}{2}$ с., после чего на 2 с. выключаться.

Параметр `sound` может иметь одно из четырёх возможных значений: `on`, `off`, `withlamp` и `against`, которые означают, соответственно, постоянно включённый звук, постоянно выключенный звук, звук, включаемый синхронно с мигающей лампой и звук, включающийся с лампой в противофазе (то есть когда лампа погашена).

Параметр `transit` позволяет задать автоматический переход в другое состояние по прошествии заданного периода времени (в секундах); кроме того, можно задать также событие, которое система должна при этом отработать. Величина интервала, состояние, в которое нужно перейти, и событие разделяются двоеточием. Как состояние, так и событие можно оставить пустыми. Например:

```
[state breakin]

transit = 30:alert
```

После перехода программы в состояние `breakin` (например, если система находилась в режиме охраны и датчики показали нарушение периметра), программа выждет 30 секунд и перейдёт в состояние `alert` (тревога). За эти 30 секунд пользователь может деактивировать систему тем или иным способом; запланированный переход в другое состояние отменяется, если до истечения заданного периода конечный автомат, к которому относятся оба состояния, по какой-либо причине своё состояние сменит. Событие здесь оставлено незаданным. То же самое можно было бы сделать иначе:

```
transit = 30::make_alert
```

В этом случае через 30 секунд после перехода в состояние `breakin` система отработает событие `make_alert`, которое, в свою очередь, может изменить состояние. Наконец, возможен и такой вариант:

```
transit = 30:alert:make_alert
```

В этом случае по истечении заданного интервала система перейдёт в состояние `alert` и отработает событие `make_alert`.

Параметр `unready` позволяет указать программе, что в данное состояние нельзя переходить, если список причин, препятствующих постановке помещения на охрану, не пуст, и указывает, в какое состояние следует перейти вместо данного. Например:

```
[state armed]

unready = wait_for_ready

[event !ready]

transit =
    wait_for_ready -> armed
```

Здесь состояние `armed` не может быть установлено, если есть причины, препятствующие постановке помещения на охрану; вместо него устанавливается состояние `wait_for_ready`. При наступлении предопределённого события `!ready`, которое генерируется при исчезновении последней такой причины, основной конечный автомат, если таковой находился в состоянии `wait_for_ready`, перейдёт в состояние `armed`. Описание секции `event` будет дано в соответствующем параграфе.

Параметр `notification` позволяет задать рассылку оповещений при переходе в данное состояние. Значение этого параметра задаёт имя и аргументы для оповещения; выполняются стандартные подстановки именованных значений (см. стр. 14); дополнительно доступно именованное значение `%newstate%`, обозначающее имя нового состояния (того, для которого описывается данное оповещение). Например:

```
[state alert]

notification = notify_owner ALERT! %date%

[notification notify_owner]

phones = 79267900950 79998877665
message = Dear owners of %sysname%! %*%
```

(описание секции `notification` будет дано в соответствующем параграфе).

3.11. Действия

При взаимодействии с работающей программой через команды, а также при насуплении определённых событий может возникнуть потребность в выполне-

нии определённого ***действия*** (включить или выключить управляемую нагрузку, перезапустить программу, изменить значения переменных и т. п.)

Нынешняя версия программы поддерживает следующие виды ***действий***:

- `@ACT+` — внесение элемента в список причин, препятствующих постановке на охрану (см. § 2.4);
- `@ACT-` — удаление элемента из списка;
- `@LN+` — включение управляемой цепи;
- `@LN-` — выключение управляемой цепи;
- `@SET` — установка значения *переменной* (см. § 3.3);
- `@RESTART` — перезапуск программы;
- `@QUIT` — завершение выполнения программы.

Первые четыре действия предполагают ровно один параметр: для `@ACT+` и `@ACT-` это идентификатор причины, а для `@LN+` и `@LN-` — идентификатор управляемой цепи (см. описание параметра `output` на стр. 20). Действие `@SET` принимает произвольное количество параметров (но не менее двух), первый параметр задаёт имя переменной, а новое значение для этой переменной образуется соединением (конкатенацией) остальных параметров. Действия `@RESTART` и `@QUIT` параметров не предполагают.

Действия могут быть использованы в описаниях ***событий*** (§ 3.12) и ***пользовательских команд*** (§ 3.14).

3.12. События

Реакция программы на наступление ***событий*** определяется группой секций `[event NAME]`, где `NAME` — ***имя события***.

Все параметры, задаваемые для события, могут зависеть от текущего состояния конечного автомата; по умолчанию используется состояние основного конечного автомата, но можно это изменить, задав параметром `fsm` имя дополнительного конечного автомата, состояния которого будут использоваться для модификации значений параметров. Например:

```
[event activity]

notification:standby = none
notification:wait_for_ready = none
notification = breakin_notification Activity detected: %1%
```

Здесь при наступлении события `activity` (см. стр. 21) во всех состояниях, кроме `standby` и `wait_for_ready`, производится рассылка сообщений о нарушении периметра (вторжении). Другой пример:

```
[event powerfail]

fsm = power
notification:ok = power_failure_notification
transit = power.* -> power.gone
```

Здесь нотификация `power_failure_notification` производится только в том случае, если до этого конечный автомат с именем `power` находился в состоянии `ok`.

Как можно догадаться из вышеприведённых примеров, параметр `notification` задаёт оповещение, которое необходимо произвести при наступлении события. Кроме этого, для события можно указать ***действие*** (параметр `action`), ***исполнение внешней команды*** (параметр `exec`), ***таблицу переходов*** (параметр `transit`) и ***подчинённые события*** (параметр `subevents`).

Начнём с описания параметра `action`, задающего *действия*, которые следует выполнить при наступлении данного события. Виды поддерживаемых действий перечислены в § 3.11. Если необходимо выполнить несколько действий сразу, их записывают на отдельных строках многострочного параметра. Например:

```
[event activity]

action = @ACT+ $1$

[event silence]

action = @ACT- $1$

[event need_warm]

action = @LN+ warm1

[event terribly_need_warm]

action =
    @LN+ warm1
    @LN+ warm2
```

Как и остальные параметры секции `event`, параметр `action` может быть вариативным относительно текущего состояния одного из конечных автоматов. Например:

```
[event sync_warmers]

fsm = climate
```

```
action:very_cold =
    @LN+ warm1
    @LN+ warm2
action:cold =
    @LN+ warm1
    @LN- warm2
action =
    @LN- warm1
    @LN- warm2
```

(в этом примере предполагается существование конечного автомата `climate`, имеющего, по меньшей мере, состояния `climate.very_cold` и `climate.cold`).

Параметр `exec` задаёт внешнюю команду, которая должна быть выполнена при наступлении данного события. После подстановки именованных значений (поддерживается стандартный набор таковых, см. стр. 14) полученная строка выполняется в специально порождённом для этого дочернем процессе с помощью стандартной функции `system()` (заметим, что эта функция, в свою очередь, порождает процесс, так что итоговая команда выполняется в процессе третьего поколения). Например:

```
[event reboot_computer]

exec = /sbin/shutdown now -r
```

Как и другие параметры, `exec` может быть задан вариативно относительно текущего состояния конечных автоматов.

Наиболее сложно организован параметр `transit`, задающий таблицу переходов. Прежде всего необходимо отметить, что по одному и тому же событию сменить состояние могут больше одного конечного автомата, причём смена состояния зависит от исходного состояния. Все такие переходы задаются в теле параметра в виде пар «исходное–новое»; в каждой паре как исходное, так и новое состояние должны принадлежать одному и тому же конечному автомату. Возможно задать правило вида «из любого другого в данное», для этого вместо имени состояния используют символ «`*`». Если таблица переходов состоит более чем из одного правила, её оформляют как многострочный параметр, указывая каждое правило на отдельной строке. Имена исходного и нового состояний разделяют стрелкой «`->`», которая обязательно **обрамляется пробелами**. Например:

```
[event cancel]

transit = * -> cancelling

[event activity]
```

```
action = @ACT+ %1%
transit =
    before_security -> before_security
    security -> breakin
    relaxed_security -> relaxed_breakin

[event powerback]

transit =
    power.gone          -> power.restoring
    power.*             -> power.ok
```

Кроме того, как и другие параметры события, параметр `transit` может быть сделан вариативным по состояниям одного из имеющихся конечных автоматов, что позволяет задать разные таблицы переходов для разных случаев, например:

```
[event powerfail]

transit:standby =
    power.*             -> power.standby_fail
transit =
    power.ok            -> power.failing
    power.restoring     -> power.failing
    power.standby_fail -> power.failing
```

Параметр `subevents` можно использовать в ситуации, когда одно событие необходимо влечёт за собой другое, либо если описания одного события недостаточно (в частности, если нужно произвести различные действия или сделать различные оповещения в зависимости от состояния не одного, а двух и более конечных автоматов). Например:

```
[event auto_on]

transit = thermo.* -> thermo.auto
subevents =
    auto_on_c1
    auto_on_c2

[event auto_on_c1]

fsm = climate1
action:low = @LN+ warm1
action:high = @LN- warm1

[event auto_on_c2]
```

```
fsm = climate2
action:low = @LN+ warm2
action:high = @LN- warm2
```

3.13. Оповещения

Варианты оповещения задаются секциями группы `[notification NAME]`, где `NAME` — имя оповещения. В нынешней версии программы эти секции подразумевают только два параметра — `phones`, задающий список номеров мобильных телефонов для отправки SMS-сообщений, и `message`, задающий шаблон сообщения.

Значение параметра `phones` формируется из номеров телефонов в международном формате, но без символа «+» в начале; отдельные номера разделяются пробелами, также возможно использование многострочного параметра, в котором номера будут разделяться символом перевода строки. В теле параметра `phones` подстановки именованных значений не производятся.

Тело параметра `message` может быть как однострочным, так и многострочным, хотя этот вариант не рекомендуется. Поддерживается стандартный набор подстановок именованных значений и позиционных параметров (см. стр. 14).

Параметры секции не предусматривают вариативности по состояниям; рекомендуется реализовывать необходимую вариативность в секциях `[event ]` и `[state ]`, где она поддерживается, в том числе и для параметра `notification`.

Пример описания оповещения:

```
[notification notify_owner]

phones = 79267900950 79998877665
message = Dear owners of %sysname%! %*%
```

Текущая версия программы не предусматривает перекодировку текста между различными кодовыми таблицами, при этом кодировка, используемая в SMS-сообщениях для символов, не входящих в ASCII, отличается от всех кодировок, применяемых в операционных средах общего назначения. В связи с этим символы non-ASCII, **в том числе русские буквы**, в сообщения включать не следует, поскольку работать это всё равно не будет. Добавить поддержку русских букв планируется в одной из следующих версий.

3.14. Пользовательские команды

Пользовательские команды (см. § 2.3) описываются секциями группы `[command NAME]`, где `NAME` — имя команды.

Напомним, что пользовательская команда может быть командой-синонимом, командой-событием, командой-действием и командой-запросом. Для всех видов команд допустимы параметры `response` (шаблон ответа на команду) и `unpriv`

(значение `unpriv = yes` указывает, что команда допустима к выполнению без авторизации).

Для всех видов команд, кроме команд-синонимов, допустимо (и необходимо) указание количества аргументов команды, для чего используется параметр `args`. Отсутствие этого параметра или указание значения `-1` приведёт к тому, что при интерпретации такой команды все слова, оставшиеся до конца командной строки, будут восприняты как параметры данной команды.

Команды-синонимы задаются параметром `alias_for`, значением которого должно быть имя (но не аргументы! только имя) другой команды, возможно, встроенной. ***Команды-события*** задаются параметром `event`, значением которого должно быть имя (и, возможно, аргументы) события. ***Команды-действия*** задаются параметром `action`, значением которого должно быть имя действия (см. стр. 25) и аргумент действия.

Параметры `alias_for`, `event` и `action` являются взаимно исключающими. Если в секции не задано ни одного из параметров `alias_for`, `event` и `action`, команда считается командой-запросом.

В телах параметров `event`, `action` и `response` выполняется стандартный набор подстановок именованных значений и позиционных параметров (см. стр. 14). Вариативность для параметров секции `command` не предусмотрена.

Примеры команд:

```
[command pass]
alias_for = &P

[command master]
alias_for = &M

[command report]
alias_for = &R

[command lineon]
action = @LN+ %1%
args = 1

[command lineoff]
action = @LN- %1%
args = 1

[command paranoid]
event = go_paranoid
args = 0

[command event]
event = %*%
```

```
args = -1

[command sett1]

args = 2
action =
  @SET low1 %1%
  @SET high1 %2%
  @RESTART
```

3.15. Web-интерфейс и web-страницы

Общие параметры встроенного web-сервера задаются секцией `[webserver]`, а каждая из поддерживаемых «страниц» — секцией `[webpage PATH]`, где `PATH` — локальная часть HTTP-адреса, обязательно начинающаяся с символа «/».

В секции `[webserver]` параметры `ip` и `port` задают, соответственно, ip-адрес и tcp-порт для сервера; в качестве ip-адреса рекомендуется использовать адрес `0.0.0.0`, означающий «все адреса системы», порт в большинстве случаев используется 80, стандартный для HTTP, но можно указать любой другой (например, в случае, если на стандартном порту работает другой web-сервер). Если порт отличается от 80, при обращении к web-серверу придётся указывать номер порта в адресной строке браузера.

Параметр `timeout` задаёт максимальный временной период неактивности клиента (в секундах), по истечении которого сервер закрывает соединение.

Параметр `realm` задаёт идентификатор «реалма» для Digest Authentication; можно выбрать произвольный идентификатор, например, «`FFazenda`». Именно это название появится в диалоговом окне браузера при запросе имени пользователя и пароля.

Параметр `users` задаёт список имён пользователей и соответствующих им паролей. Имя и пароль разделяются двоеточием, отдельные пары разделяются запятыми, либо располагаются на отдельных строках. Например:

```
users = vasya:1234321, masha:PussyCat, yanggeek:1sS-7sL
```

`Пробельные символы игнорируются` и не должны встречаться внутри имён и паролей, также в именах и паролях нельзя использовать запятую и двоеточие, т. к. они используются в качестве разделитетей.

Параметр `p404` задаёт содержимое (текст) страницы, выдаваемой в ответ на ошибочные запросы (ошибка 404). Текст должен быть представлен в формате HTML.

Наконец, группа параметров `html:XXXX`, где `XXXX` — макроидентификатор, позволяет задать фрагменты текста HTML, которые затем можно использовать

при построении страниц. Например, значение параметра `html:header` будет подставляться в качестве именованного значения `%html:header%`; такие подстановки, наряду со стандартными, выполняются в телах всех параметров, предполагающих HTML-текст, в том числе в секции `[webserver]` — в параметре `p404` и самих параметрах группы `html:XXXX`, что позволяет этим параметрам использовать друг друга. Пример законченной секции `[webserver]`:

```
[webserver]

ip = 0.0.0.0
port = 80
timeout = 60
realm = FFazenda
html:header =
 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
 <HTML><HEAD><TITLE>The FFazenda web interface</TITLE></HEAD>
 <BODY bgcolor="grey" text="white" link="blue" vlink="orange">
html:footer =
 </BODY></HTML>
html:p404 =
 <H1>This is FFazenda 404 page</H1>
 <P>The local path that you've requested is unknown for the system.
 </P>
p404 = %html:header% %html:p404% %html:footer%
users = vasya:1234321, masha:PussyCat, yanggeek:1sS-7sL
```

Секции группы `[webpage ]` позволяют описывать «страницы» трёх типов:

- `page` — обычная страница, содержимое которой задано в конфигурационном файле;
- `file` — документ, получаемый из внешнего файла (удобно использовать, например, для вставки изображений);
- `command` — страница, обрабатывающая запросы типа `POST` и интерпретирующая такие запросы в качестве команд.

Тип страницы задаётся параметром `type`.

Кроме того, поддерживаются страницы-синонимы, единственным параметром которых является параметр `alias_for`; его значением является локальный путь другой страницы, остальные параметры игнорируются.

Для всех типов страниц (кроме страниц-синонимов) поддерживаются параметры `mimetype`, задающий тип содержимого, и `public`, значение которого (`yes` или `no`) указывает, разрешено ли обращаться к этой странице без прохождения аутентификации (ввода пароля).

Для страниц типа `page` поддерживается параметр `content`, значением которого является полное содержимое данной страницы (обычно на языке HTML). В

большинстве случаев для экономии места при описании тел страниц используют именованные значения %html:XXXX%, заданные в секции [webserver].

Для страниц типа file поддерживается параметр path, задающий путь к файлу содержимого. Рекомендуется использовать полный путь. **Не забудьте указать тип файла, используя параметр** mimetype**!**

Наконец, для страниц типа command поддерживается параметр content — аналогично типу page, но в данном случае этот параметр задаёт содержимое, отдаваемое на запрос типа GET; и параметр response, задающий содержание, отдаваемое сервером в ответ на запрос POST. В теле параметра response можно использовать именованное значение %cmdresponse%, вместо которого программа подставит результат выполнения присланной команды. **Не забудьте сделать страницу закрытой, установив** public = no!

Страницы типа command обрабатывают данные, полученные методом POST от веб-форм, содержащих либо один элемент типа input с именем (параметром name), равным "command", либо последовательность элементов типа input с именами "cmd0", "cmd1", "cmd2" и т. д. В первом случае значение, полученное из веб-формы, рассматривается как команда с параметрами; во втором случае значением cmd0 рассматривается как имя команды, а остальные значения — как её параметры. Например, если вы сконфигурировали страницу /cmd с типом command, то следующая веб-форма позволит вам выполнять через эту страницу произвольные пользовательские команды, вводимые в поле ввода:

```
<form name="command_form" action="/cmd" method="POST">
  Type your command:
  <input type="text" name="command">
  <input type="submit" value="Submit it!">
</form>
```

Следующая форма, состоящая из одной кнопки с надписью WARM1 ON, позволяет по нажатию этой кнопки выполнить пользовательскую команду lineon warm1:

```
<form name="warm1_on" action="/cmd" method="POST">
  <input type="hidden" name="command" value="lineon warm1">
  <input type="submit" value="WARM1 ON">
</form>
```

Следующая форма, состоящая из двух полей ввода и кнопки, позволит изменить значения ***переменных*** (см. § 3.3) low1 и high1, если в вашем конфигурационном файле описана пользовательская команда sett1, как показано на стр. 31.

```
<form name="set_t1" action="/ctrl" method="POST">
  <strong>Set t1</strong>
  <input type="hidden" name="cmd0" value="sett1">
  Low: <input type="text" name="cmd1" value="%?low1%" size=5>
  High: <input type="text" name="cmd2" value="%?high1%" size=5>
  <input type="submit" value="Set!">
</form>
```

Приведём примеры описаний страниц:

```
[webpage /test.html]

type = page
public = yes
content = %html:header% <h1>IT WORKS</h1> %html:footer%

[webpage /cmd]

type = command
public = no
content = %html:header% <h1>The Command Page</h1>
    <form name="command_form" action="/cmd" method="POST">
    Type your command: <input type="text" name="command">
    <input type="submit" value="Submit it!">
    </form><br />
    %html:footer%
response = %html:header% <h1>COMMAND RESPONSE</h1>
    <pre>%cmdresponse%</pre>
    %html:footer%

[webpage /logo.png]

type = file
public = yes
path = /usr/local/ffazenda/logo.png
mimetype = image/png

[webpage /]

type = page
public = yes
content = %html:header% <img src="/logo.png" alt="Logo" />
   <p><a href="/test.html">Go to the test page</a></p>
   <p><a href="/cmd">Go to the command page</a></p>
   %html:footer%
```

IV. Установка и запуск программы

Программа поставляется в двух вариантах: в виде архива исходных текстов и в виде статически собранного исполняемого файла для архитектуры i386 и ОС Linux. При использовании архива исходных текстов следует раскрыть его:

```
tar -xzf ffazenda-1.1.65.tgz
```

зайти в появившуюся директорию:

```
cd ffazenda-1.65
```

и дать команду `make`, после чего, скорее всего, исполняемый файл будет собран. **Для успешной сборки в вашей системе должна быть установлена библиотека `libusb` с заголовочными файлами**. Если при сборке возникли проблемы, убедительно просим сообщить о них.

Если сборка прошла успешно, то в директории, в которой вы дали команду `make`, появится исполняемый файл `ffazenda`; это и есть собранная программа. Рекомендуется после этого дать команду `make clean`, чтобы удалить все временные файлы, возникшие в процессе сборки; это сэкономит вам около 2 Mb дискового пространства.

Обратите внимание на поддиректорию `samples`, в которой находятся примеры конфигурационных файлов. Один из них, например `security.ini`, вы можете взять за основу для написания вашего собственного конфигурационного файла; см. гл. III настоящего руководства.

Инсталляции в обычном смысле программа не требует. Рекомендуется (но не обязательно) скопировать исполняемый файл `ffazenda` в директорию `/usr/local/bin`. Для использования программы потребуется файл конфигурации, структура и семантика которого были описаны выше; рекомендуется (но не обязательно) назвать этот файл `ffazenda.ini` и расположить его в директории `/usr/local/etc`, но с таким же успехом можно расположить его где угодно и выбрать любое удобное вам имя.

Если вы используете возможности интеграции с пакетом smstools для отправки и получения SMS-сообщений, необходимо будет создать директории для временного хранения создаваемых сообщений и для хранения архива обработанных сообщений. **Эти директории должны находиться на одной файловой системе с теми, которые использует для нужд обработки сообщений пакет smstools,** в противном случае программа не сможет использовать операцию переименования файла для перемещения файлов между этими директориями; как следствие, работать ничего не будет. Обычно smstools использует поддиректории `/var/spool/sms/` (`incoming` для входящих сообщений, `outgoing` для исходящих, `failed` для сообщений, попытка отправки которых окончилась неудачей). Мы рекомендуем использовать для работы программы директории `/var/spool/ffazenda/composing` и `/var/spool/ffazenda/proceeded`. Создать их можно, например, так:

```
mkdir /var/spool/ffazenda
mkdir /var/spool/ffazenda/composing
mkdir /var/spool/ffazenda/proceeded
```

Напоминаем, что имена этих директорий указываются в конфигурационном файле в секции `[sms]` параметрами `composing` и `proceeded`, например:

```
[sms]

mynumber = 79995551234
incoming = /var/spool/sms/incoming
outgoing = /var/spool/sms/outgoing
failed = /var/spool/sms/failed
composing = /var/spool/ffazenda/sms
proceeded = /var/spool/ffazenda/proceeded
```

Последним этапом инсталляции является организация автоматического запуска программы при старте вашей системы. Как правило, для этого достаточно добавить в файл `/etc/rc.local` строку вида

```
/usr/local/bin/ffazenda -c /usr/local/etc/ffazenda.ini &
```

(если вы решили использовать другие имена для исполняемого файла и файла конфигурации, команда в вашем случае будет соответствующим образом отличаться). Если в вашей системе отсутствует файл `/etc/rc.local` или команды из него по каким-то причинам не выполняются при загрузке системы, обратитесь к документации по вашему дистрибутиву Linux.

ПРЕДУПРЕЖДЕНИЕ: Параметр опции `-c`, задающий имя конфигурационного файла, **должен быть указан в виде полного пути**, то есть обязан начинаться с символа «/», в противном случае у вас могут возникнуть проблемы с автоматическим перезапуском программы. Дело здесь в том, что после запуска программа проходит стандартную процедуру *демонизации*, в ходе которой, в частности, меняет текущую директорию на корневую; как следствие, относительные пути к файлам сразу после демонизации работать перестают.

Дополнительно можно использовать флаги `-f` (запрещает программе уходить в фоновый режим и переходить в состояние «демона»), а также `-l` <*имя-файла*> (задаёт имя лог-файла, которое будет использоваться вместо указанного в конфигурации).

Компания «Юниконтроллерз» будет признательна за любые замечания по работе, набору возможностей и внутреннему устройству программы `FFazenda`, а также по тексту настоящего руководства.

Мы всегда рады ответить на любые вопросы относительно выпускаемой нами продукции и программного обеспечения. Кроме того, при желании вы можете заказать у нас услуги по подбору и закупке оборудования для построения систем «Умный дом/умная дача» на основе наших универсальных контроллеров, а также по проведению пусконаладочных работ, включающих установку, конфигурирование и настройку всего необходимого программного обеспечения.