PYLON
Application Platform
Руководство программиста

Версия 0.1.1 (5 Июля 2007 г.)

Pylon Software Development Group — 2007

Цена надежности — это погоня за предельной простотой.
Такая цена по карману не каждому богатому.

Сэр Энтони Хоар, 1980

Тот, кто хочет выполнить работу хорошо,
должен сперва привести в порядок инструмент.

Конфуций, VI век до н.э.

Оглавление

Глава 1

Общие сведения

Единственной целью, которую преследовал автор при создании этого продукта и документации к нему, являлось создание совершенной, то есть полностью лишенной недостатков (и имеющей только ограничения) платформы для создания встраиваемых приложений. Под совершенством здесь подразумевается не совершенство с точки зрения пользователя, работающего с продуктом, построенным на основе этой платформы, а совершенство с точки зрения программиста, использующего эту платформу для создания приложения. У автора есть глубокое убеждение, что программисты[1], работающие с хорошими инструментами, создают хорошие продукты.

1.1 Основные принципы

Следующие принципы были положены в основу платформы.

1. **Один процесс.** Подразумевается, что приложение, созданное с использованием Пилона, представляет собою один исполняемый файл, порождающий один процесс. При проектировании автор совсем не думал о том, что приложение может быть многопоточным.

2. **Unix forever.** При проектировании везде, где это казалось естественным, автор старался скопировать способы решения проблем из Unix'а.

3. **Qt forever.** Из Qt скопированы сигналы и слоты. Иерархия классов отличается сильно, но прослеживаются некоторые перепевы. Соглашения об именовании взяты из Qt.

4. **Все для программиста.** При проектировании основной упор делался на облегчение программирования с использованием платформы. Одна из целей проекта — дать возможность писать меньше вспомогательного кода и сосредоточиться на алгоритмах.

5. **Одно дерево исходников — много приложений.** Система сборки позволяет "класть" рядом несколько приложений, которые будут компилироваться из одних исходников.

6. **Forth тоже forever.** В качестве языка для скриптования и для шела выбран Форт.

1.1.1 Соглашения о стиле оформления кода

Автор считает этот раздел чрезвычайно важным, так как поддержание унифицированного стиля оформления исходного кода не только делает этот код более красивым с эстетической точки зрения, но и позволяет лучше его понимать нескольким авторам.

Следующие правила должны соблюдаться при оформлении кода.

1. **Отступы.** Размер отступов — 2 пробела. В тексте нигде не должен использоваться символ табуляции.

2. **Фигурные скобки.** Приведенный ниже пример иллюстрирует расположение фигурных скобок.

[1] За исключением быть может нескольких долбо...бов.

```
int f()
{
  int i = 0, j;

  if (i < 0) {
    i = 0;
    j = 0;
  } else {
      i = 1;
      j = 1;
    }
  return i;
}
```

Listing 1.1: Расположение фигурных скобок

Другими словами, при оформлении кода функции открывающая фигурная скобка должна располагаться на следующей строке после декларации функции. При использовании же блоков для циклов и для условного оператора открывающая фигурная скобка должна находиться на той же строке. else в условном операторе должен находиться на той же строке, что и закрывающая фигурная скобка.

3. **Имена.** В имени любой сущности, если оно состоит из нескольких слов или из сокращений нескольких слов, каждое новое слово, кроме первого, должно начинаться с заглавной буквы. Регистр первой буквы первого слова определяется видом сущности. В именах сущностей следует избегать символов подчеркивания. Это означает, что символы подчеркивания могут употребляться только в именах, сгенерированных макросами препроцессора, и глобальных константах.

   - **Имя типа.** Имя типа должно начинаться с заглавной буквы.
   - **Имя функции или метода.** Имя функции должно начинаться со строчной буквы или, если это статическая функция в коде на C, с символа подчеркивания.
   - **Имя переменной.** Имя переменной должно начинаться со строчной буквы или, если это приватный член класса или статическая переменная в коде на C, с символа подчеркивания.

4. **Декларация класса.** В декларации класса вначале должны объявляться публичные члены, затем — защищенные, и в конце — приватные. Порядок реализации членов должен совпадать с порядком их декларации.

5. **Круглые скобки.** При использовании круглых скобок они не должны окружаться пробелами.

```
int f( int a1, int a2 ); – неправильно.
int f(int a1, int a2); – правильно.

int a = ( a + b ) * ( c - d ); – неправильно.
int a = (a + b) * (c - d); – правильно.

if ( a < b ); – неправильно.
if (a < b); – правильно.
```

Listing 1.2: Использование круглых скобок

6. **Константы.** Имена констант, объявленных при помощи define и имена глобальных констант должны содержать только заглавные буквы. Слова, из которых состоит имя константы, должны быть разделены символом подчеркивания. Следует избегать объявления локальных констант. Существует специальный файл *src/lib/general/include/Limits.h*, который содержит определения всех глобальных констант. Объявление констант в одном файле значительно облегчает их поиск. Само собой разумеется, что нужно стремиться к минимизации числа глобальных констант. Если объявляются несколько констант, объединенных одним смыслом, то их объявления могут быть выровнены.

```
// Error codes.
static const int OK      =  0;
static const int FAILURE = -1;
```

Listing 1.3: Объявление группы констант

7. **Сообщения.** Несмотря на то, что сообщения являются глобальными текстовыми константами, они объявляются в том же файле, в котором объявляется класс их использующий. Исключение составляют сообщения, имеющие очень общий смысл (файловые ошибки, нулевой указатель и пр.), которые могут использоваться сразу несколькими классами. Такие сообщения должны быть объявлены

в файле *src/lib/general/include/Messages.h*. Имена констант-сообщений должны формироваться из префикса "MSG_", имени класса в верхнем регистре и описательной части как показано в следующем примере.

```
// Messages.
static const char MSG_SOCKET_CANNOT_CREATE[]      = "Cannot create network connection (%s).";
static const char MSG_SOCKET_CANNOT_CONNECT[]     = "Cannot open network connection (%s).";
static const char MSG_SOCKET_CANNOT_READ[]        = "Cannot read data from network (%s).";
static const char MSG_SOCKET_CANNOT_WRITE[]       = "Cannot write data to network (%s).";
static const char MSG_SOCKET_CANNOT_CLOSE[]       = "Cannot close network connection (%s).";
static const char MSG_SOCKET_CANNOT_SHUTDOWN[]    = "Cannot shutdown network connection (%s).";
static const char MSG_SOCKET_UNKNOWN_HOST[]       = "Unknown host name '%s'.";
static const char MSG_SOCKET_UNKNOWN_SERVICE[]    = "Unknown service name '%s'.";
static const char MSG_SOCKET_CANNOT_SET_OPTION[]  = "Cannot set socket option (%s).";
static const char MSG_SOCKET_CORRUPTED[]          = "Socket corrupted.";
static const char MSG_SOCKET_CANNOT_GET_INFO[]    = "Cannot get socket information (%s).";
static const char MSG_SOCKET_CANNOT_BIND[]        = "Cannot bind socket (%s).";
```

Listing 1.4: Объявление констант-сообщений

8. **Локальные переменные.** Объявления локальных переменных не должны быть выровнены.

```
int          a; — неправильно.
char         b;
string       c;
SomeTypeName d;

int a; — правильно.
char b;
string c;
SomeTypeName d;
```

Listing 1.5: Объявление локальных переменных

9. **Проверка кода возврата.** Поскольку PYLON не использует механизм исключений, программист должен всегда проверять код возврата используемых методов и функций. Создаваемые программистом функции и методы также должны использовать коды возврата. Типичный код для вызова метода класса выглядит следующим образом.

```
int res;
if (FAILED(res = _core->spawn(f->second, params)))
  return ERROR_BACKTRACE(res);
```

Listing 1.6: Проверка кода возврата

10. **Создание объектов.** При создании объектов невозможно проверить код возврата конструктора — конструктор не имеет возможности его вернуть обычным образом. Для решения этой проблемы используется класс `Support`. Все классы в системе должны быть наследниками этого класса. `Support` предоставляет два метода — `failure` и `failureCode`. Первый используется для установки кода ошибки внутри конструктора класса.

```
int res;
if (FAILED(res = CONNECT(server, accepted, this, Inetd, startService))) {
  failure(ERROR_BACKTRACE(res));
  return;
}
```

Listing 1.7: Использование `failure` в конструкторе класса

Второй используется для проверки успешности создания объекта.

```
int res;
if (FAILED(res = _instance->failureCode())) {
  ERROR_BACKTRACE(res);
  delete _instance;
  ERROR_FATAL(MSG_OBJECT_CANNOT_CREATE, typeid(T).name());
}
```

Listing 1.8: Проверка успешности создания объекта

1.1.2 Структура каталогов и система сборки

Исходники платформы жестко разделены по каталогам.

Каталог	Описание
pylon	Корень иерархии.
pylon/build	Правила системы сборки.
pylon/doc	Эта документация.
pylon/etc	Примеры файлов конфигурации.
pylon/src	Корень иерархии исходников.
pylon/src/app	Приложения.
pylon/src/lib	Библиотеки.
pylon/src/srv	Сервисы.
pylon/utils	Перловые утилиты, использующиеся системой сборки.

Программист, использующий платформу, может добавлять свой код в три каталога: *pylon/src/app*, *pylon/src/lib* и *pylon/src/srv*.

Подкаталоги этих каталогов имеют специальное значение. Имена этих подкаталогов становятся в процессе компиляции именами библиотек (в случае каталога *pylon/src/app* — именами исполняемых файлов).

Все подкаталоги каталогов *pylon/src/lib* и *pylon/src/srv* должны содержать подкаталоги *include* и *src*, причем последний из них должен содержать файл *Makefile*. Если процесс сборки библиотеки, сервиса или приложения не имеет никаких особенностей, то этот файл должен быть следующим.

```
###
### libgeneral --- General stuff.
###

LOCAL_CFLAGS   =
LOCAL_CPPFLAGS =
LOCAL_INC      =
LOCAL_DEFS     =
C_SRCS         = $(wildcard *.c)
CPP_SRCS       = $(wildcard *.cpp)

include $(PYLON_PATH)/build/make_dep_rules.mak
```

Listing 1.9: *Makefile* для библиотеки `libgeneral` (файл *../../../src/lib/general/src/Makefile*)

В каталоге приложения должен находиться файл *app.mak*, определяющий тулчейн, при помощи которого это приложение строится.

```
###
### Platform definitions.
###
### $Header$
###

AR       = ar cr
CC       = gcc
CLINK    = gcc
CPP      = g++
CPPLINK  = g++
INC      =
LFLAGS   =
LIBS     =
CFLAGS   = -Wall -Wextra -Werror -Wno-unused
CPPFLAGS = -Wall -W -Werror -fno-exceptions
DEFS     = -DPYLON_APP_$(PYLON_APP)
ifdef MAKE_DEBUG
CFLAGS   += -O0 -g
CPPFLAGS += -O0 -g
DEFS     += -DDEBUG_MODE
else
CFLAGS   += -O2
CPPFLAGS += -O2
endif
```

Listing 1.10: *app.mak* для тестового приложения (файл *../../../src/app/pylon/app.mak*)

Кроме этого, в каталоге приложения находится файл *BUILD*, содержащий номер текущего билда приложения. Каждый раз при построении приложения это число будет увеличиваться на единицу утилитой `buildinfo.pl`.

Главный *Makefile* (*pylon/Makefile*) анализирует значения нескольких переменных окружения.

Имя переменной	Описание
PYLON_APP	Если установлена, то ее значение используется как имя каталога приложения — подкаталога *pylon/src/app*. Если она не установлена, то используется первый с точки зрения сортировки по алфавиту по убыванию подкаталог этого каталога.
MAKE_NOCOLORS	Если установлена, то в процессе сборки при выводе сообщений не будет использоваться цвет.

1.1.3 Обзор библиотек

В приведенной ниже таблице кратко описываются библиотеки, являющиеся частью PYLON (расположенные в каталоге *pylon/lib*).

Библиотека (без префикса *lib*)	Описание
atlast	Форт.
config	Сохранение и восстановление значений реестра из конфигурационного файла.
core	Классы для создания сервисов, таймеров и пр.
general	Классы общего назначения (генерация ошибок, сигналы, слоты, корень иерархии объектов).
getline	История, поддержка управляющих символов для шела.
license	Проверка лицензии.
memory	Замена стандартных функция работы с памятью (malloc и др.). Позволяет контролировать выделение памяти. Используется только для отладки.
network	Работа с сетью.
registry	Реестр.
syslog	Протокол syslog.
telnet	Протокол telnet.
value	Классы для хранения и сериализации значений разных типов. Используются реестром.

1.1.4 Обзор сервисов

В приведенной ниже таблице кратко описываются сервисы PYLON'а (катаклог *pylon/srv*).

Сервис	Описание
fsh	Форт-шел. Все взаимодействие пользователя с платформой происходит при помощи этого шела.
inetd	Этот сервис выполняет те же функции, что и одноименный демон в Unix'е. Он слушает заданные порты TCP и запускает соответствующие сервисы, когда на эти порты приходит входящее соединение.
init	Этот сервис запускается ядром платформы автоматически при старте. Функция этого сервиса — запуск других сервисов при старте приложения.

Глава 2

Особенности реализации

2.1 Иерархия классов

Два класса платформы являются основными. Это классы `Support` и `Object`. `Support` имеет только одно назначение. Он предоставляет средства для проверки успешности выполнения конструктора.

Любой класс, использующийся в приложении, построенном на основе ℙ𝕐𝕃ℂℕ'а, должен быть потомком `Support`.

`Object` Является более сложным классом. Он предоставляет поддержку сигналов, слотов и реестра. Кроме этого, любой объект класса, унаследованного от `Object`, получает уникальный целый положительный идентификатор, который можно использовать для поиска этого объекта. Если нет каких-то особых обстоятельств, нужно стремиться к тому, чтобы все классы приложения были наследниками `Object`.

Каждый потомок `Object` должен использовать в своем объявлении специальный макрос `PYLON_OBJECT`. В противном случае во время компиляции вы увидите ошибки, связанные с наличием абстрактных методов в вашем классе.

```
#ifndef __SIGNALBASE_H__
#define __SIGNALBASE_H__

/*

$Header$

*/

#include "Object.h"
#include "String.h"
#include "Slot.h"

using namespace std;

class SignalBase : public Object
{

  PYLON_OBJECT

public:

  SignalBase(const String &className, const String &signalName);
  virtual ~SignalBase();

  String name() const;
  String ownerClassName() const;

  virtual int disconnect(const Object *slotObject) = 0;

  int listSlots(String &text);

  const Object *_orphanSlotObject;
  SlotMethod _orphanSlotMethod;
  SlotMap _slots;

protected:

  int bind(Object *obj);
  int unbind(Object *obj);

private:

  String _className;
  String _name;
};
```

```
inline
String SIGNAL_COLOR_TITLE(const String &str)
{
  return COLOR_STRING(str, LIGHT_CYAN, NONE);
}

inline
SignalBase *SIGNAL_BASE(Object *obj)
{
  return obj ? dynamic_cast<SignalBase *>(obj) : NULL;
}

#endif
```

Listing 2.1: Пример потомка класса `Object` (файл *../../../src/lib/sigslot/include/SignalBase.h*)

2.2 Сигналы и слоты

Как и в Qt [3], концепция сигналов и слотов является фундаментом ℙ𝕐𝕃ℕ'а. Любой класс, который использует сигналы и(или) слоты должен быть потомком `Object`. В отличие от Qt ℙ𝕐𝕃ℕ не использует внешнего метакомпилятора для реализации сигналов и слотов. Вместо этого используются шаблонные класы и макросы препроцессора. При этом автор старался достичь такого же удобства использования сигналов и слотов, которое предоставляет Qt.

Слоты ℙ𝕐𝕃ℕ'а представляют собою колбэки на методы класса. При этом программист может быть уверен в том, что он не сможет соединить несовпадающие по сигнатурам сигнал и слот. Проверка этих сигнатур выполняется на этапе компиляции.

Реализация сигналов и слотов таким образом оказалась возможной благодаря тому, что указатель на виртуальный метод класса хранит и индекс в VMT, и указатель на функцию [2].

```
#include <iostream>
#include <string>

using namespace std;

class A
{
public:
  virtual void f()
  {
    std::cout << "A::f()\n";
  }
  void g()
  {
    std::cout << "A::g()\n";
  }
};

class B : public A
{
public:
  virtual void f()
  {
    std::cout << "B::f()\n";
  }
  void g()
  {
    std::cout << "B::g()\n";
  };
};

typedef void (A::*VirtualFunctionPointer)();

int main()
{
  A a;
  B b;
  A* pb = &b;

  VirtualFunctionPointer p = (VirtualFunctionPointer)&B::f;

  (pb->*p)();

  p = &A::g;
  (pb->*p)();

  return 0;
}
```

Listing 2.2: Использование указателей на методы класса (файл *src/sigslot.cpp*)

```
B::f()
A::g()
```

Listing 2.3: Вывод предыдущего кода

Для декларации сигнала нужно использовать макрос `SIGNAL` внутри объявления класса.

```
SIGNAL( имясигнала [, типаргумента 1 [, типаргумента 2 [..., типаргумента n ]]] );
```

Listing 2.4: Декларация сигнала

Сигнал может быть объявлен с любым доступом: **private**, **protected** или **public**. При этом смысл ограничения доступа сохраняется. То есть подписываться на private-сигналы может только сам объект класса, подписываться на protected-сигналы может сам объект и объекты классов-потомков, public-сигналы доступны для подписки всем.

З а м е ч а н и е 2.2.0.1. *Как и в случае со всеми другими членами класса, нужно стремиться публиковать минимальное количество сигналов, необходимых для использования класса*

Следующий пример воспроизводит объявление сигнала `accepted` в классе `TcpServer`. Этот сигнал срабатывает, когда к серверу пытается присоединиться новый клиент.

```
#ifndef __TCPSERVER_H__
#define __TCPSERVER_H__

/*

$Header$

*/

#include <netinet/in.h>

#include "Poller.h"
#include "TcpConnection.h"

// Messages.
static const char *const MSG_SOCKET_CANNOT_LISTEN           = "Cannot listen socket (%s).";
static const char *const MSG_SOCKET_CANNOT_ACCEPT_CONNECTION = "Cannot accept imcoming connection (%s).";

static const unsigned int TCP_SERVER_CONNECTION_QUEUE_SIZE = 16;

using namespace std;

#define TCP_SERVER(c) (dynamic_cast<TcpServer *>(c))

class TcpServer : public Socket
{

friend class Poller;

  PYLON_OBJECT

public:

  TcpServer(const String &bindAddress, const String &service);
  virtual ~TcpServer();

  virtual int accept(TcpConnection **connection);

  // Signals.
  SIGNAL(accepted, TcpConnection *);

private:

  String _service;
  struct sockaddr_in _addr;
};

#endif
```

Listing 2.5: Объявление сигнала в классе `TcpServer` (файл *../../../src/lib/network/include/TcpServer.h*)

Для декларации слота используется макрос `SLOT` внутри объявления класса.

```
SLOT( имяслота [, типаргумента 1 [, типаргумента 2 [..., типаргумента n ]]] );
```

Listing 2.6: Декларация слота

Как и сигнал, слот тоже может быть обявлен с любым доступом.

```
#ifndef __FSH_H__
#define __FSH_H__

/*

$Header$

*/

#include "pcrecpp.h"
#include "Service.h"
#include "TcpConnection.h"
#include "FShGetLine.h"
#include "Telnet.h"
#include "Registry.h"

static const char *const NAME_FSH = "fsh";
static const char *const MAN_FSH  = "\
Service <service>fsh</service> is a <keyword>Forth Shell</keyword>. That is what you are \
working with right now. To see all the available <keyword>Forth</keyword> words say <word>words</word>. \
To discover what a cool thing <keyword>Forth</keyword> is go to <url>http://www.forth.org.ru</url>. \
";

// Registry manuals.

static const char *const MAN_FSH_COLORS_ON = "\
<registry>bool fsh.colors.on(true)</registry>\n\n\
Turn on and off colored output in the Forth shell (<service>fsh</service> service). \
It is a global property. It affects all Forth shells in the system.\
";

static const char *const MAN_FSH_LOG = "\
<registry>bool fsh.log.errors.on(false)</registry>\n\
<registry>String fsh.log.errors.regexp()</registry>\n\
<registry>bool fsh.log.warnings.on(false)</registry>\n\
<registry>String fsh.log.warnings.regexp()</registry>\n\
<registry>bool fsh.log.debug.on(false)</registry>\n\
<registry>String fsh.log.debug.regexp()</registry>\n\n\
Entries <registry>fsh.log.*.on</registry> turn on and off \
logs printing to a Forth shell. <registry>fsh.log.*.regexp</registry> \
allow to filter those logs. See man page for word <word>GREP</word> \
for <keyword>regular expressions</keyword> explanation. \
These are private properties. They affect only the Forth shell they belong to.\
";

using namespace pcrecpp;

class FSh : public Service<FSh>
{

friend class Service<FSh>;
friend class FShGetLine;

  PYLON_OBJECT

public:

  inline
  static String NAME();

  inline
  static String MANUAL();

protected:

  FSh(const ValueList &params);
  virtual ~FSh();

private:

  int start();

  SLOT(dataReady);
  SLOT(die);

  SLOT(setColorEnabled, const bool &yes);

  SLOT(setLogErrorsEnabled, const bool &yes = true);
  SLOT(setLogErrorsRegexp, const String &re);

  SLOT(setLogWarningsEnabled, const bool &yes = true);
```

```
  SLOT(setLogWarningsRegexp, const String &re);

  SLOT(setLogDebugEnabled, const bool &yes = true);
  SLOT(setLogDebugRegexp, const String &re);

  SLOT(logError, const String &timestamp, const String &fileName,
                 const unsigned int lineNum, const String &functionName, const String &msg);
  SLOT(logWarning, const String &timestamp, const String &fileName,
                   const unsigned int lineNum, const String &functionName, const String &msg);
  SLOT(logDebug, const String &timestamp, const String &fileName,
                 const unsigned int lineNum, const String &functionName, const String &msg);

  REGISTRY(bool, colors__on, true, MAN_FSH_COLORS_ON);

  REGISTRY_PRIVATE(bool,   log__errors__on,       true,  MAN_FSH_LOG);
  REGISTRY_PRIVATE(String, log__errors__regexp,   "",    MAN_FSH_LOG);
  REGISTRY_PRIVATE(bool,   log__warnings__on,     false, MAN_FSH_LOG);
  REGISTRY_PRIVATE(String, log__warnings__regexp, "",    MAN_FSH_LOG);
  REGISTRY_PRIVATE(bool,   log__debug__on,        false, MAN_FSH_LOG);
  REGISTRY_PRIVATE(String, log__debug__regexp,    "",    MAN_FSH_LOG);

  TcpConnection *_connection;
  FShGetLine *_getLine;
  Telnet *_telnet;

  bool _logErrors;
  RE _logErrorsRe;

  bool _logWarnings;
  RE _logWarningsRe;

  bool _logDebug;
  RE _logDebugRe;
};

inline
String FSh::NAME()
{
  return NAME_FSH;
}

inline
String FSh::MANUAL()
{
  return MAN_FSH;
}

#endif
```

Listing 2.7: Декларация слота в классе `Fsh` (файл *../../../src/srv/fsh/include/FSh.h*)

З а м е ч а н и е 2.2.0.2. *Фактически слот представляет собою виртуальный метод класса, возвращающий целый код ошибки.*

Реализация слота — это реализация метода класса.

```
/*

$Header$

*/

#include "Core.h"
#include "TcpConnection.h"
#include "Registry.h"

#include "FSh.h"

// Protected methods.

FSh::FSh(const ValueList &params) : Service<FSh>(params),
  _connection(NULL),
  _getLine(NULL),
  _telnet(NULL),
  _logErrors(false),
  _logErrorsRe("", RE_Options().set_caseless(true)),
  _logWarnings(false),
  _logWarningsRe("", RE_Options().set_caseless(true)),
  _logDebug(false),
  _logDebugRe("", RE_Options().set_caseless(true))
{
  BIND(colors__on);

  BIND(log__errors__on);
  BIND(log__errors__regexp);
```

```
  BIND(log__warnings__on);
  BIND(log__warnings__regexp);
  BIND(log__debug__on);
  BIND(log__debug__regexp);

  int res;

  if (params.size() < 1) {
    failure(ERROR(SERVICE_MSG_INVALID_PARAMS, CSTRING(NAME())));
    return;
  }

  ValueObject *obj = queryInterface<ValueObject>(params[0]);
  if (!obj || !(_connection = queryInterface<TcpConnection>(obj->get()))) {
    failure(ERROR(SERVICE_MSG_INVALID_PARAM_TYPE, CSTRING(NAME())));
    return;
  }

  if (FAILED(res = start())) {
    failure(ERROR_BACKTRACE(res));
    return;
  }
}

FSh::~FSh()
{
  if (_telnet)
    delete _telnet;
  if (_getLine)
    delete _getLine;
  if (_connection)
    delete _connection;
}

// Private methods.

int FSh::start()
{
  String re;
  bool yes;
  int res;

  if (FAILED(res = REGISTRY_GET(FSh, log__errors__on, yes, this)))
    return ERROR_BACKTRACE(res);
  if (FAILED(res = setLogErrorsEnabled(yes)))
    return ERROR_BACKTRACE(res);

  if (FAILED(res = REGISTRY_GET(FSh, log__errors__regexp, re, this)))
    return ERROR_BACKTRACE(res);
  if (FAILED(res = setLogErrorsRegexp(re)))
    return ERROR_BACKTRACE(res);

  if (FAILED(res = REGISTRY_GET(FSh, log__warnings__on, yes, this)))
    return ERROR_BACKTRACE(res);
  if (FAILED(res = setLogWarningsEnabled(yes)))
    return ERROR_BACKTRACE(res);

  if (FAILED(res = REGISTRY_GET(FSh, log__warnings__regexp, re, this)))
    return ERROR_BACKTRACE(res);
  if (FAILED(res = setLogWarningsRegexp(re)))
    return ERROR_BACKTRACE(res);

  if (FAILED(res = REGISTRY_GET(FSh, log__debug__on, yes, this)))
    return ERROR_BACKTRACE(res);
  if (FAILED(res = setLogDebugEnabled(yes)))
    return ERROR_BACKTRACE(res);

  if (FAILED(res = REGISTRY_GET(FSh, log__debug__regexp, re, this)))
    return ERROR_BACKTRACE(res);
  if (FAILED(res = setLogDebugRegexp(re)))
    return ERROR_BACKTRACE(res);

  if (FAILED(res = _connection->setNonBlock()))
    return ERROR_BACKTRACE(res);

  if (!(_getLine = new FShGetLine(this)))
    return ERROR(MSG_OBJECT_CANNOT_CREATE, "FShGetLine");
  if (FAILED(res = _getLine->failureCode()))
    return ERROR_BACKTRACE(res);

  if (!(_telnet = new Telnet(_connection, _getLine)))
    return ERROR(MSG_OBJECT_CANNOT_CREATE, "Telnet");
  if (FAILED(res = _telnet->failureCode()))
    return ERROR_BACKTRACE(res);

  if (FAILED(res = _telnet->iac(DO, TELOPT_NAWS, 0, 0, 0, 0)))
    return ERROR_BACKTRACE(res);
  if (FAILED(res = _telnet->iac(DO, TELOPT_TTYPE, 0, 0, 0, 0)))
```

```
    return ERROR_BACKTRACE(res);
  if (FAILED(res = _telnet->iac(WILL, TELOPT_ECHO, 0, 0, 0, 0)))
    return ERROR_BACKTRACE(res);
  if (FAILED(res = _telnet->iac(WILL, TELOPT_SGA, 0, 0, 0, 0)))
    return ERROR_BACKTRACE(res);

  if (FAILED(res = CONNECT(_connection, dataReady, this, FSh, dataReady)))
    return ERROR_BACKTRACE(res);

  if (FAILED(res = CONNECT(_connection, dead, this, FSh, die)))
    return ERROR_BACKTRACE(res);

  if (FAILED(res = CONNECT(colors__on(), set, this, FSh, setColorEnabled)))
    return ERROR_BACKTRACE(res);

  if (FAILED(res = CONNECT(log__errors__on(), set, this, FSh, setLogErrorsEnabled)))
    return ERROR_BACKTRACE(res);
  if (FAILED(res = CONNECT(log__errors__regexp(), set, this, FSh, setLogErrorsRegexp)))
    return ERROR_BACKTRACE(res);

  if (FAILED(res = CONNECT(log__warnings__on(), set, this, FSh, setLogWarningsEnabled)))
    return ERROR_BACKTRACE(res);
  if (FAILED(res = CONNECT(log__warnings__regexp(), set, this, FSh, setLogWarningsRegexp)))
    return ERROR_BACKTRACE(res);

  if (FAILED(res = CONNECT(log__debug__on(), set, this, FSh, setLogDebugEnabled)))
    return ERROR_BACKTRACE(res);
  if (FAILED(res = CONNECT(log__debug__regexp(), set, this, FSh, setLogDebugRegexp)))
    return ERROR_BACKTRACE(res);

  if (FAILED(res = CONNECT(ERR(), error, this, FSh, logError)))
    return ERROR_BACKTRACE(res);
  if (FAILED(res = CONNECT(ERR(), warning, this, FSh, logWarning)))
    return ERROR_BACKTRACE(res);
  if (FAILED(res = CONNECT(ERR(), debug, this, FSh, logDebug)))
    return ERROR_BACKTRACE(res);

  return OK;
}

int FSh::dataReady()
{
  uint8_t ch;
  int res;

  if (FAILED(res = _connection->read((char *)&ch, sizeof(ch), 1))) {
    ERROR_BACKTRACE(res);
    if (FAILED(res = kill()))
      ERROR_BACKTRACE(res);
    return res;
  }

  if (ch == IAC) {
    int readCnt;

    if (FAILED(res = _telnet->parse(&readCnt)))
      return ERROR_BACKTRACE(res);
  } else
      if (FAILED(res = _getLine->pushChar(ch)))
        return ERROR_BACKTRACE(res);

  return OK;
}

int FSh::die()
{
  int res;
  if (FAILED(res = CORE()->kill(this)))
    return ERROR_BACKTRACE(res);
  return OK;
}

int FSh::setColorEnabled(const bool &yes)
{
  int res;
  if (FAILED(res = String::setColorEnabled(yes)))
    return ERROR_BACKTRACE(res);

  return OK;
}

int FSh::setLogErrorsEnabled(const bool &yes)
{
  _logErrors = yes;
  return OK;
}

int FSh::setLogErrorsRegexp(const String &re)
```

```
{
  if (_logErrorsRe.pattern() != re)
    _logErrorsRe = re;
  return OK;
}

int FSh::setLogWarningsEnabled(const bool &yes)
{
  _logWarnings = yes;
  return OK;
}

int FSh::setLogWarningsRegexp(const String &re)
{
  if (_logWarningsRe.pattern() != re)
    _logWarningsRe = re;
  return OK;
}

int FSh::setLogDebugEnabled(const bool &yes)
{
  _logDebug = yes;
  return OK;
}

int FSh::setLogDebugRegexp(const String &re)
{
  if (_logDebugRe.pattern() != re)
    _logDebugRe = re;
  return OK;
}

int FSh::logError(const String &timestamp, const String &fileName,
                  const unsigned int lineNum, const String &functionName, const String &msg)
{
  if (!_logErrors)
    return OK;
#ifdef DEBUG_MODE
  String str;
  str.sprintf("%s [%s] [file: %s, line: %u, function: %s]: %s%s",
              CSTRING(COLOR_ERROR()), CSTRING(timestamp),
              CSTRING(fileName), lineNum, CSTRING(functionName), CSTRING(msg),
              String::eol());
#else
  String str;
  str.sprintf("%s: %s%s",
              CSTRING(COLOR_ERROR()), CSTRING(msg),
              String::eol());
#endif
  int res;
  if (_logErrorsRe.PartialMatch(str))
    if (FAILED(res = FORTH()->printf(CSTRING(str))))
      return ERROR_BACKTRACE(res);
  return OK;
}

int FSh::logWarning(const String &timestamp, const String &fileName,
                    const unsigned int lineNum, const String &functionName, const String &msg)
{
  if (!_logWarnings)
    return OK;
#ifdef DEBUG_MODE
  String str;
  str.sprintf("%s [%s] [file: %s, line: %u, function: %s]: %s%s",
              CSTRING(COLOR_WARNING()), CSTRING(timestamp),
              CSTRING(fileName), lineNum, CSTRING(functionName), CSTRING(msg),
              String::eol());
#else
  String str;
  str.sprintf("%s: %s%s",
              CSTRING(COLOR_WARNING()), CSTRING(msg),
              String::eol());
#endif
  int res;
  if (_logWarningsRe.PartialMatch(str))
    if (FAILED(res = FORTH()->printf(CSTRING(str))))
      return ERROR_BACKTRACE(res);
  return OK;
}

int FSh::logDebug(const String &timestamp, const String &fileName,
                  const unsigned int lineNum, const String &functionName, const String &msg)
{
  if (!_logDebug)
    return OK;
#ifdef DEBUG_MODE
  String str;
  str.sprintf("%s [%s] [file: %s, line: %u, function: %s]: %s%s",
```

```
                CSTRING(COLOR_DEBUG()), CSTRING(timestamp),
                CSTRING(fileName), lineNum, CSTRING(functionName), CSTRING(msg),
                String::eol());
  #else
    String str;
    str.sprintf("%s: %s%s",
                CSTRING(COLOR_DEBUG()), CSTRING(msg),
                String::eol());
  #endif
    int res;
    if (_logDebugRe.PartialMatch(str))
      if (FAILED(res = FORTH()->printf(CSTRING(str))))
        return ERROR_BACKTRACE(res);
    return OK;
  }
```

Listing 2.8: Реализация слота в классе `Fsh` (файл *../../../src/srv/fsh/src/FSh.cpp*)

Листинг 2.8 также содержит пример использования макроса `CONNECT`, “подключающего” слот к сигналу.

```
CONNECT(signalObjectPtr, signalName, slotObjectPtr, slotClassName, slotMethodName);
```

Listing 2.9: Подключение слота к сигналу

Здесь

- `signalObjectPtr` — Указатель на объект, которому принадлежит сигнал.
- `signalName` — Имя сигнала.
- `slotObjectPtr` — Указатель на объект, которому принадлежит слот.
- `slotClassName` — Имя класса объекта, которому принадлежит слот.
- `slotMethodName` — Имя слота.

Замечание 2.2.0.3. *В отличие от Qt* PYLON *позволяет подключать только сигнал к слоту. То есть нет, например, возможности подключить сигнал к другому сигналу, слот к слоту и т.п.*

Определение 2.2.0.1. *Класс, в котором определен сигнал (слот), будем называть владельцем этого сигнала (слота), а о сигнале (слоте) будем говорить, что он принадлежит этому классу.*

2.3 Сериализация и десериализация

Сериализация и десериализация типов используется PYLON'ом для реализации реестра. Здесь используются два класса — `Value<>` и `S11n<>`.

Класс `Value<>` является контейнером значений элементарных типов и типов, определенных программистом. Этот класс похож на класс `QVariant` из Qt [4]. `Value<>` — это шаблонный класс, параметром для которого является хранимый им тип.

В случае, если программисту необходимо определить элементы реестра (см. раздел 2.4) неэлементарного типа, ему необходимо определить специализацию класса `S11n<>` для этого типа[1]. Эта специализация будет использоваться для сериализации и десериализации объектов этого хранимого типа. В следующем примере приведена специализация класса `S11n<>` для хранения значения типа `InetdItem`.

```
#ifndef __S11NINETDITEM_H__
#define __S11NINETDITEM_H__

/*

$Header$

*/

#include "S11n.h"
#include "InetdItem.h"

static const char *const INETD_ITEM_DELIMITER = ":";

// Tags.

static const char *const XML_TAG_SERVICE  = "service";
```

[1] S11n — это сокращение от слова “Serialization”, в котором между буквами ’S’ и ’n’ содержатся 11 букв.

```
static const char *const XML_TAG_PORT     = "port";
static const char *const XML_TAG_LOGIN    = "login";
static const char *const XML_TAG_PASSWORD = "password";
static const char *const XML_TAG_ALLOW    = "allow";
static const char *const XML_TAG_DENY     = "deny";

template<>
int S11n<InetdItem>::toString(const InetdItem &value, String &str);

template<>
int S11n<InetdItem>::fromString(const String &str, InetdItem &value);

template<>
int S11n<InetdItem>::toXml(const InetdItem &value, XmlElement *element);

template<>
int S11n<InetdItem>::fromXml(const XmlElement *element, InetdItem &value);

#endif
```

Listing 2.10: Декларация специализации S11n<InetdItem> (файл *../../../src/srv/inetd/include/S11nInetdItem.h*)

```
/*

$Header$

*/

#include "Parser.h"
#include "Config.h"

#include "S11nInetdItem.h"

template<>
int S11n<InetdItem>::toString(const InetdItem &value, String &str)
{
  str = value.service + INETD_ITEM_DELIMITER +
        String::number(value.port) + INETD_ITEM_DELIMITER +
        value.login + INETD_ITEM_DELIMITER +
        value.password + INETD_ITEM_DELIMITER +
        value.allow + INETD_ITEM_DELIMITER +
        value.deny + INETD_ITEM_DELIMITER;
  return OK;
}

template<>
int S11n<InetdItem>::fromString(const String &str, InetdItem &value)
{
  StringList list;
  unsigned int i, size;
  int res;

  if (FAILED(res = Parser::split(str, list, 0, INETD_ITEM_DELIMITER)))
    return ERROR_BACKTRACE(res);
  if ((size = list.size()) < 2)
    return ERROR(MSG_CONFIG_WRONG_FORMAT, CSTRING(str));

  for (i = 0; i < size; ++i)
    switch (i) {
      case 0:
        value.service = list[i];
        break;
      case 1:
        if (FAILED(res = list[i].toUint(value.port)))
          return ERROR_BACKTRACE(res);
        break;
      case 2:
        value.login = list[i];
        break;
      case 3:
        value.password = list[i];
        break;
      case 4:
        value.allow = list[i];
        break;
      case 5:
        value.deny = list[i];
        break;
    }
  return OK;
}

template<>
int S11n<InetdItem>::toXml(const InetdItem &value, XmlElement *element)
{
```

```
  CHECK_POINTER(element);

  int res;

  if (FAILED(res = element->removeChildren()))
    return ERROR_BACKTRACE(res);

  XmlElement *root =
    new XmlElement(XML_TAG_SERVICE, new XmlAttribute(XML_ATTR_VALUE, value.service), "",
      new XmlElement(XML_TAG_PORT,     NULL, String::number(value.port), NULL,
      new XmlElement(XML_TAG_LOGIN,    NULL, value.login, NULL,
      new XmlElement(XML_TAG_PASSWORD, NULL, value.password, NULL,
      new XmlElement(XML_TAG_ALLOW,    NULL, value.allow, NULL,
      new XmlElement(XML_TAG_DENY,     NULL, value.deny)))))
    );
  if (FAILED(res = element->appendChild(root)))
    return ERROR_BACKTRACE(res);

  return OK;
}

template<>
int S11n<InetdItem>::fromXml(const XmlElement *element, InetdItem &value)
{
  CHECK_POINTER(element);

  XmlElementCollection *children = element->children();
  unsigned int i, size = children->size();
  XmlElement *e;
  int res;

  value.clear();

  for (i = 0; i < size; ++i) {
    e = children->at(i);
    if (e->tag().equals(XML_TAG_SERVICE, true)) {
      if (FAILED(res = e->attribute(XML_ATTR_NAME, value.service)))
        return ERROR_BACKTRACE(res);

      XmlElementCollection *children = e->children();
      unsigned int i, size = children->size();
      XmlElement *el;

      for (i = 0; i < size; ++i) {
        el = children->at(i);
        if (el->tag().equals(XML_TAG_PORT, true)) {
          if (FAILED(res = el->innerData().toUint(value.port)))
            return ERROR_BACKTRACE(res);
        } else
            if (el->tag().equals(XML_TAG_LOGIN, true))
              value.login = el->innerData();
            else
              if (el->tag().equals(XML_TAG_PASSWORD, true))
                value.password = el->innerData();
              else
                if (el->tag().equals(XML_TAG_ALLOW, true))
                  value.allow = el->innerData();
                else
                  if (el->tag().equals(XML_TAG_DENY, true))
                    value.deny = el->innerData();
      }
    }
  }

  return OK;
}
```

Listing 2.11: Реализация специализации `S11n<InetdItem>` (файл *../../../src/srv/inetd/src/S11nInetdItem.cpp*)

Из листингов видно, что значения типа `InetdItem` сериализируются при помощи строки, содержащей значения полей этой структуры, разделенные двоеточием.

Определены специализации класса `S11n<>` для следующих хранимых типов.

Хранимый тип	Описание	Описание сериализации
`bool`	Элементарный тип.	Строка “true” для истинного значения и строка “false” для ложного.
`char`	Элементарный тип.	Строка, содержащая символ.
`double`	Элементарный тип.	Строка, содержащая значение.
`int`	Элементарный тип.	Строка, содержащая значение.
`unsigned int`	Элементарный тип.	Строка, содержащая значение.
`std::string`	Строка STL.	Используется само значение.
`void *`	Указатель.	Строка, содержащая целое значение указателя.
`Object *`	Указатель на объект PYLON’а.	Строка, содержащая Id объекта.
`std::vector<>`	Вектор STL.	Строка, содержащая сериализации элементов вектора, разделенные вертикальной чертой.

2.4 Реестр

Определение 2.4.0.2. *Каждый класс, являющийся потомком* `Object`*, может “публиковать” некоторые свои члены-поля таким образом, чтобы к этим членам могли получить доступ другие классы. Такие поля мы будем называть* **элементами реестра** *класса. Класс, в котором определены эти поля будем называть владельцем этих элементов реестра, а о самих элементах будем говорить, что они принадлежат классу.*

Определение 2.4.0.3. *Набор всех элементов реестра класса будем называть* **реестром** *класса.*

Элементы реестра могут быть двух типов — **общие** и **частные**. Об общих элементах можно домать как о статических членах класса — все объекты этого класса разделяют однин и тот же экземпляр члена. Частные элементы — это обычные члены класса — каждый объект имеет свой экземпляр такого члена. Общий элемент реестра декларируется при помощи макроса `REGISTRY`, частный — при помощи макроса `REGISTRY_PRIVATE`. Оба эти макроса имееют одинаковые аргументы, поэтому рассмотрим только один.

```
REGISTRY(valueType, entryName, defaultValue [, flags]);
```

Listing 2.12: Объявление элемента реестра

Здесь

- `valueType` — Тип значения, хранимого элементом реестра.
- `entryName` — Имя элемента реестра.
- `defaultValue` — Значение по умолчанию типа `valueType`.
- `flags` — Флаги.

Для элемента реестра могут быть установлены следующие флаги.

Флаг	Описание	Установлен по умолчанию
REG_PERSISTENCY_YES	Объявляет персистентный элемент реестра.	Да
REG_PERSISTENCY_NO	Объявляет неперсистентный элемент реестра.	Нет
REG_ACCESS_READ_ONLY	Элемент только для чтения.	Нет
REG_ACCESS_WRITE_ONLY	Элемент только для записи.	Нет
REG_ACCESS_FULL	Разрешен любой доступ к элементу.	Да
REG_VISIBILITY_PRIVATE	Доступ к элементу разрешен только объекту класса, которому принадлежит этот элемент.	Нет
REG_VISIBILITY_PROTECTED	Доступ к элементу разрешен всем объектам класса, которому принадлежит этот элемент.	Нет
REG_VISIBILITY_PUBLIC	Доступ к элементу разрешен всем классам процесса.	Нет
REG_VISIBILITY_PUBLISHED	Доступ к элементу разрешен всем процессам.	Да

Замечание 2.4.0.4. *В случае общего элемента реестра флаги* REG_VISIBILITY_PRIVATE *и* REG_VISIBILITY_PROTECTED *имеют один и тот же смысл.*

```
#ifndef __FSH_H__
#define __FSH_H__

/*

$Header$

*/

#include "pcrecpp.h"
#include "Service.h"
#include "TcpConnection.h"
#include "FShGetLine.h"
#include "Telnet.h"
#include "Registry.h"

static const char *const NAME_FSH = "fsh";
static const char *const MAN_FSH  = "\
Service <service>fsh</service> is a <keyword>Forth Shell</keyword>. That is what you are \
working with right now. To see all the available <keyword>Forth</keyword> words say <word>words</word>. \
To discover what a cool thing <keyword>Forth</keyword> is go to <url>http://www.forth.org.ru</url>. \
";

// Registry manuals.

static const char *const MAN_FSH_COLORS_ON = "\
<registry>bool fsh.colors.on(true)</registry>\n\n\
Turn on and off colored output in the Forth shell (<service>fsh</service> service). \
It is a global property. It affects all Forth shells in the system.\
";

static const char *const MAN_FSH_LOG = "\
<registry>bool fsh.log.errors.on(false)</registry>\n\
<registry>String fsh.log.errors.regexp()</registry>\n\
<registry>bool fsh.log.warnings.on(false)</registry>\n\
<registry>String fsh.log.warnings.regexp()</registry>\n\
<registry>bool fsh.log.debug.on(false)</registry>\n\
<registry>String fsh.log.debug.regexp()</registry>\n\n\
Entries <registry>fsh.log.*.on</registry> turn on and off \
logs printing to a Forth shell. <registry>fsh.log.*.regexp</registry> \
allow to filter those logs. See man page for word <word>GREP</word> \
for <keyword>regular expressions</keyword> explanation. \
These are private properties. They affect only the Forth shell they belong to.\
";

using namespace pcrecpp;

class FSh : public Service<FSh>
{

friend class Service<FSh>;
friend class FShGetLine;

  PYLON_OBJECT
```

```
  public:

    inline
    static String NAME();

    inline
    static String MANUAL();

  protected:

    FSh(const ValueList &params);
    virtual ~FSh();

  private:

    int start();

    SLOT(dataReady);
    SLOT(die);

    SLOT(setColorEnabled, const bool &yes);

    SLOT(setLogErrorsEnabled, const bool &yes = true);
    SLOT(setLogErrorsRegexp, const String &re);

    SLOT(setLogWarningsEnabled, const bool &yes = true);
    SLOT(setLogWarningsRegexp, const String &re);

    SLOT(setLogDebugEnabled, const bool &yes = true);
    SLOT(setLogDebugRegexp, const String &re);

    SLOT(logError, const String &timestamp, const String &fileName,
                   const unsigned int lineNum, const String &functionName, const String &msg);
    SLOT(logWarning, const String &timestamp, const String &fileName,
                     const unsigned int lineNum, const String &functionName, const String &msg);
    SLOT(logDebug, const String &timestamp, const String &fileName,
                   const unsigned int lineNum, const String &functionName, const String &msg);

    REGISTRY(bool, colors__on, true, MAN_FSH_COLORS_ON);

    REGISTRY_PRIVATE(bool,   log__errors__on,       true,  MAN_FSH_LOG);
    REGISTRY_PRIVATE(String, log__errors__regexp,   "",    MAN_FSH_LOG);
    REGISTRY_PRIVATE(bool,   log__warnings__on,     false, MAN_FSH_LOG);
    REGISTRY_PRIVATE(String, log__warnings__regexp, "",    MAN_FSH_LOG);
    REGISTRY_PRIVATE(bool,   log__debug__on,        false, MAN_FSH_LOG);
    REGISTRY_PRIVATE(String, log__debug__regexp,    "",    MAN_FSH_LOG);

    TcpConnection *_connection;
    FShGetLine *_getLine;
    Telnet *_telnet;

    bool _logErrors;
    RE _logErrorsRe;

    bool _logWarnings;
    RE _logWarningsRe;

    bool _logDebug;
    RE _logDebugRe;
  };

  inline
  String FSh::NAME()
  {
    return NAME_FSH;
  }

  inline
  String FSh::MANUAL()
  {
    return MAN_FSH;
  }

  #endif
```

Listing 2.13: Декларация общего элемента реестра в класса `FSh` (файл *../../../src/srv/fsh/include/FSh.h*)

Каждый элемент реестра является классом-потомком класса `Object`, в котором определены два сигнала: `set` и `got`. Первый срабатывает, после того, как элементу реестра присвоено новое значение, второй — до того, как значение прочитано из элемента реестра.

З а м е ч а н и е 2.4.0.5. *Даже если в слоте, подключенном к сигналу* `got`*, изменить значение элемента реестра, запросившему значение этого элемента будет возвращено старое значение. При этом,*

конечно, значение элемента будет изменено. Именно поэтому имена сигналов элемента реестра — глаголы в прошедшем времени.

В отличие от обычных членов-полей, к элементам реестра нужно обращаться специальным образом, используя макросы REGISTRY_GET и REGISTRY_SET.

```
REGISTRY_GET(name [, owner])
REGISTRY_SET(name, value [, owner]);
```

Listing 2.14: Объявление элемента реестра

Здесь

- name — Имя элемента реестра (см. Замечание 2.4.0.6.).
- value — Новое значение.
- owner — Указатель на объект-владелец элемента реестра. Должен быть указан только, если элемент является частным.

З а м е ч а н и е 2.4.0.6. *Имя элемента реестра при использовании макросов* REGISTRY_GET *и* REGISTRY_SET *является строкой, которая формируется следующим образом. Берется имя класса-владельца элемента реестра, приведенное к нижнему регистру, к нему справа прибавляется символ '.' (точка), а затем справа прибавляется имя элемента реестра (значение параметра* entryName *из объявления 2.12), приведенное к нижнему регистру, в котором все символы подчеркивания заменены на точки. Наепример, для обращения к элементу реестра, объявленному в листинге 2.13, должно использоваться имя* fsh.colors.on*. Такое соглашение об именовании позволяет строить древовидную иерархию имен элементов реестра.*

2.5 Сервисы

Глава 3

Справочник по классам

Литература

[1] Учебники Форта // http://wiki.forth.org.ru/%D3%F7%E5%E1%ED%E8%EA%E8%D4%EE%F0%F2%E0

[2] К вопросу о вопросах по C++ // http://singalen.livejournal.com/49410.html.

[3] Qt Signals and Slots // http://doc.trolltech.com/4.3/signalsandslots.html.

[4] QVariant Class Reference // http://doc.trolltech.com/4.3/qvariant.html.

[5] Страуструп Б. Язык программирования C++, спец. изд. / Пер. с англ. — М.; СПб.: “Издательство БИНОМ” — “Невский Диалект”, 2001 г. — 1099 с.