Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»

Факультет вечернего, заочного и дистанционного обучения
Кафедра электронных вычислительных машин

Контрольная работа
по дисциплине
«Технология проектирования программного обеспечения ЭВМ»
на тему «Текстовый редактор с подсветкой синтаксиса»
студента 4 курса 500503 учебной группы
Авсеева С.П.

Минск 2009

Содержание

1. Постановка задачи

Реализовать текстовый редактор. В редакторе осуществить поддержку создание, отрытие, и сохраниение файлов в тектовом формате. Так же реализовать подсветку синтаксиса.

Минимальные требования к компьютеру:

1. Unix-подобная операционная система (тестирование производилось на Ubuntu Linux 9.04 jaunty);

2. библиотека GTK2+ (возможно операционная система из семейства Windows с установленным GTK2+);

3. интерпретатор языка ruby (http://ruby-lang.org), тестирование производилось с версией 1.8.7, приложение должно работать с любой версией ветки 1.8.x;

4. привязки GTK2+ для руби: пакет libgtk2-ruby (для Ubuntu Linux можно найти в репозитории)

2. Обзор методов и алгоритмов решения задачи

Выбор платформы, для которой разрабатывалось приложение обусловлен в первую очередь тем, что Ubuntu Linux — свободная операционная система, доступная любому пользователю, в отличие от других систем требующих платной лицензии. Во-вторых под этой системой проще настроить окружение для разработки, чем под MS Windows.

Далее необходимо было выбрать язык, на котором будет реализовано приложение. Был выбран ruby как полностью объектно-ориентированный (в отличие от C++), динамический язык широкого профиля.

Среди доступных для ruby графических библиотек наиболее развиты:

1. libgtk2-ruby, привязка для GTK2+, графической подсистеме GNOME;

2. qt4-qtruby, привязка для QT, графической подсистеме KDE;

3. fxruby, привязка к Fox Toolkit, кросплатформенной блиблиотеке, для разработки графических интерфейсов.

Для разработки приложения была выбрана библиотека libgtk2, так как на моем компьютере оконная система — GNOME и данный интерфейс для неё — «родной».

Все вышеперечисленные пакеты, кроссплатформенные и, учитывая кроссплатформенность интерпретатора ruby, можно утверждать что разработанное приложение так же будет кроссплатформенным и будет запускаться на всех системах, для которых доступно вышеуказанное окружение.

Основная проблема, возникшая при разработке — техника подсветки синтаксиса. После изучения элементов управления, предоставленных в GTK было решено использовать класс Gtk::TextView для отображения текста и Gtk::TextBuffer для управления буфером, в котором хранится текст.

Процесс подсветки можно разбить на следующие стадии:

1. настройка тегов, для кусков текста, имеющих специальное значение (например строки, зарезервированнные слова и т.д.);

2. разбор текста, который пользователь вводит, или загружает из файла;

3. пометка кусков текста, распознанных как специальные, соответствующими тегами;

4. отображение текста для пользователя

Для разбора текста были использованы регулярные выражения, описывающие текст, который нужно подсветить.

3. UML модели программы

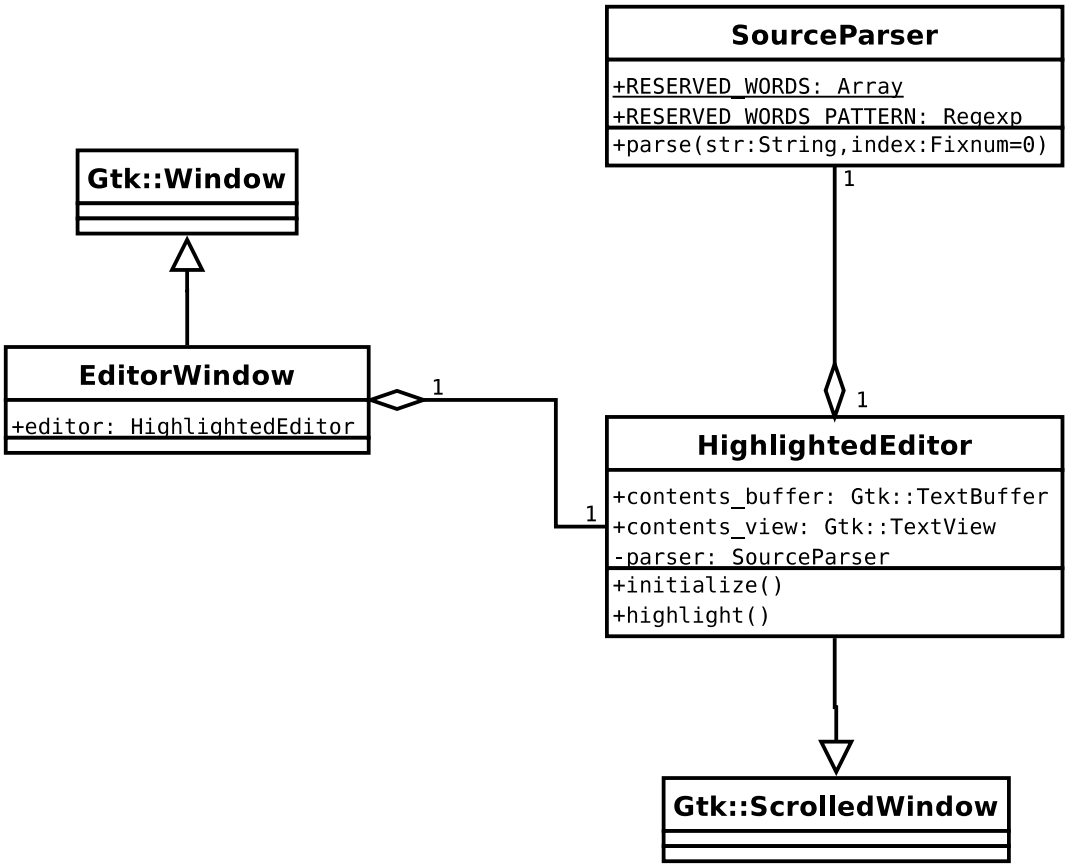

Рис. 3.1. Диаграмма классов

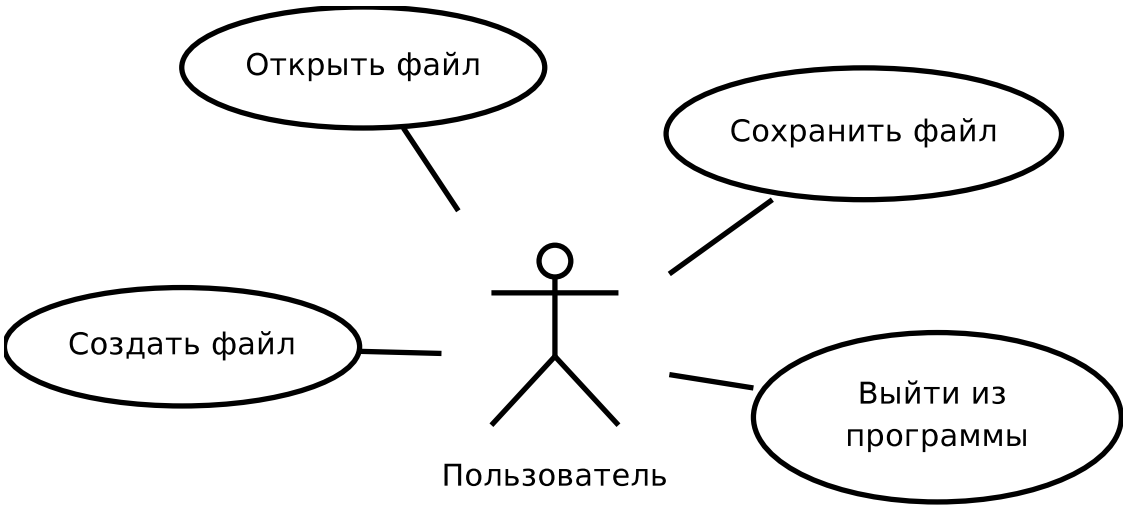

Рис. 3.2. Диаграмма прецендентов

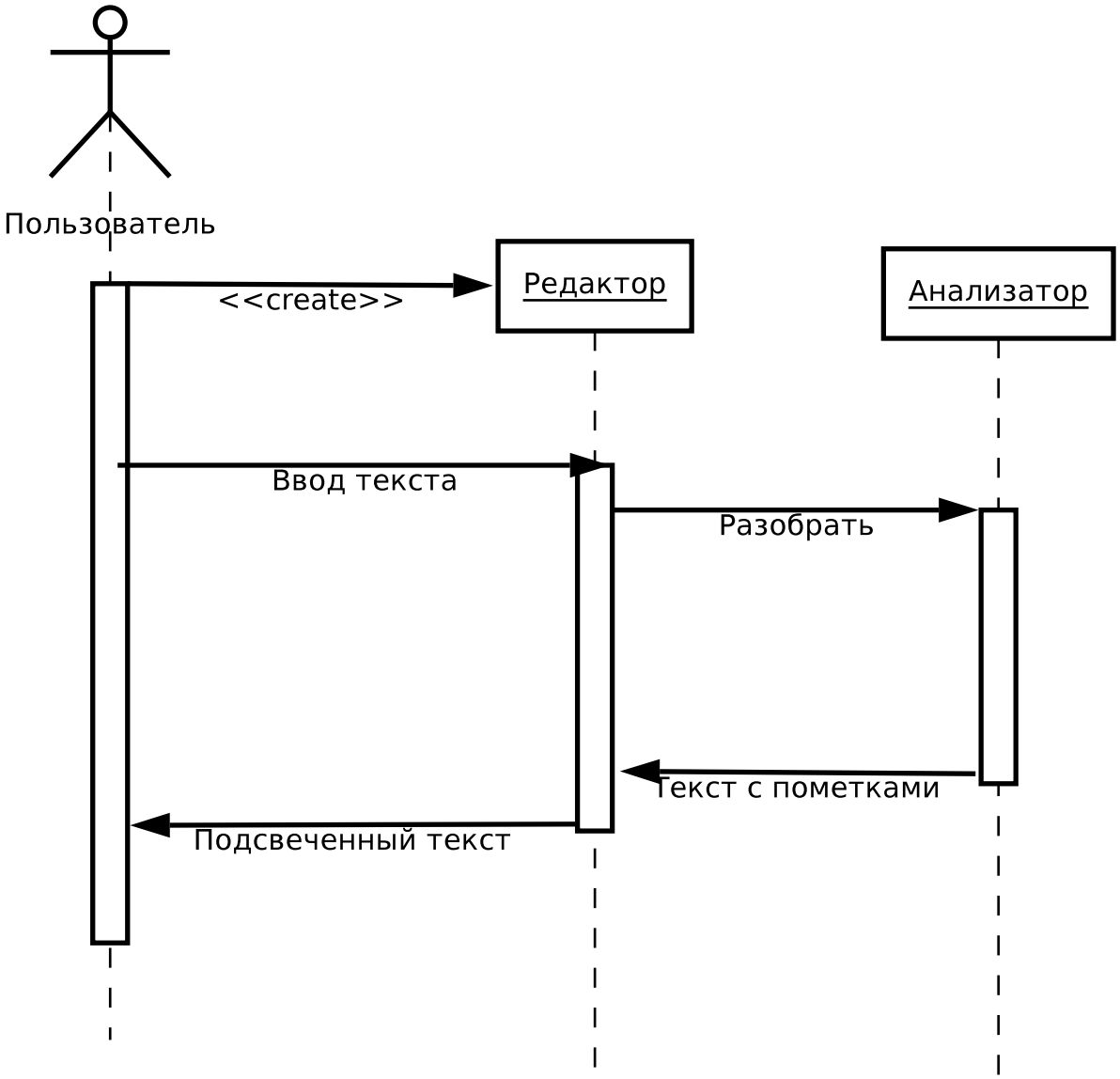

Рис. 3.3. Диаграмма последовательности

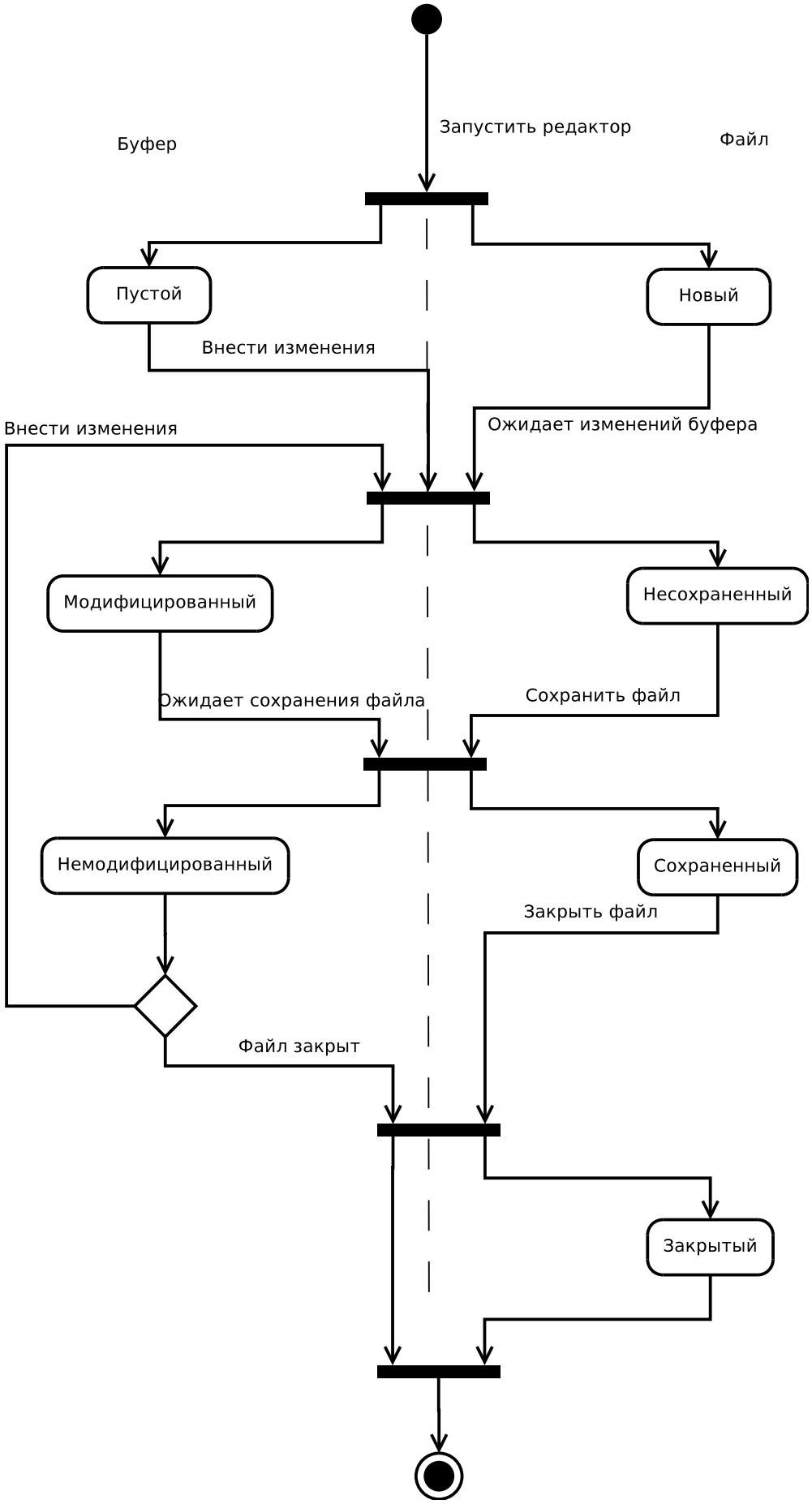

Рис. 3.4. Диаграмма взаимодействия

4. Описание структуры программы

Приложение представляет собой набор классов, которые выполняют всю работу:

EditorWindow: класс для представления главного окна приложения, создает и содержит элементы управления, а так же навешивает обработчики на события;

HighlightedEditor: элемент управления, служащий для работы с текстом, поддерживающий подсветку синтаксиса;

SourceParser: класс для анализа текста и вычленения из него кусков, имеющих специальное значение и которые можно подсветить.

Рассмотрим работу приложения по-подробнее. Во время инициализации объекта HighlightedEditor мы определяем набор тегов для элементов текста, подлежащий подсветке: строка, комментарий, константа и зарезервированное слово. Для каждого тега устанавливаются определенный цвет и стиль начертания. Далее создаем пару связанных объектов классов Gtk::TextBuffer и Gtk::TextView. Первый занимается хранением и управлением текста, а второй его отображением. Регистрируем обработчик события «changed» для буфера, в котором мы вызываем метод highlight, чтобы просканировать и обновить подсветку. В этом методе вызывается метод parse для объекта класса SourceParser, которому передается текст и блок, который он в свою очередь вызывает, когда определяет значимый кусок текста. В блок передается тип куска (строка, комментарий, константа или зарезервированное слово) и границы текста. Имея ту информацию мы можем применить теги. Перед началом разбора все предыдущие теги очищаются.

Рассмотрим работу анализатора. В этом классе определен набор регулярных выражений для каждого элемента исходного кода. Метод parse сопоставляет переданный текст этим выражениям и вызывает блок, если сопоставление удачно. В блок передается имя группы и границы текста, соответствующего найденной группе.

5. Исходный код программы

5.1. Главный класс программы

Создает экземпляр главного окна и отображает его.

```
require 'gtk2'
require 'editor_window'

main = EditorWindow.new
main.set_default_size(600, 400)
main.show_all

Gtk.main
```

5.2. Класс окна

Создает окно с панелью управления и меню. Так же содержит область редактирования.

```
require 'gtk2'
require 'highlighted_editor'

class EditorWindow < Gtk::Window
  attr_accessor :editor

  def initialize
    super(Gtk::Window::TOPLEVEL)
    set_title("GTK Editor")
    signal_connect("destroy"){Gtk.main_quit}
    @editor = HighlightedEditor.new

    callback_quit = Proc.new { Gtk.main_quit }

    callback_open = Proc.new do
      dialog =  Gtk::FileChooserDialog.new("Open File...", nil,
        Gtk::FileChooser::ACTION_OPEN,
        "gnome-vfs",
        [Gtk::Stock::OPEN, Gtk::Dialog::RESPONSE_ACCEPT],
        [Gtk::Stock::CANCEL, Gtk::Dialog::RESPONSE_CANCEL]
      )

      filter_rb = Gtk::FileFilter.new
      filter_rb.name = "Ruby Scripts"
      filter_rb.add_pattern("*.rb")
      filter_rb.add_pattern("*.rbw")
      dialog.add_filter(filter_rb)

      if dialog.run == Gtk::Dialog::RESPONSE_ACCEPT
```

```
    File.open(dialog.filename,  "r") do |f|
      @editor.contents_buffer.set_text(f.readlines.join)
    end
  end
  dialog.destroy
end

callback_save = Proc.new do
  dialog =  Gtk::FileChooserDialog.new("Save As...", nil,
    Gtk::FileChooser::ACTION_SAVE,
    "gnome-vfs",
    [Gtk::Stock::SAVE, Gtk::Dialog::RESPONSE_ACCEPT],
    [Gtk::Stock::CANCEL, Gtk::Dialog::RESPONSE_CANCEL]
  )

  filter_rb = Gtk::FileFilter.new
  filter_rb.name = "Ruby Scripts"
  filter_rb.add_pattern("*.rb")
  filter_rb.add_pattern("*.rbw")
  dialog.add_filter(filter_rb)

  if dialog.run == Gtk::Dialog::RESPONSE_ACCEPT
    File.open(dialog.filename,  "w+") do |f|
      f.write @editor.contents_buffer.get_text
    end
  end
  dialog.destroy
end

callback_about = Proc.new do
  Gtk::AboutDialog.set_url_hook {|about, link|}

  Gtk::AboutDialog.show(self,
    :program_name => "GTK+ Editor",
    :version => "1.0",
    :website => "http://avsej.net",
    :comments => "Контрольная работа по курсу ТППО.",
    :authors => ["Сергей Авсеев"])
end


actions = [
  ["FileMenu", nil, "_File"],
  ["HelpMenu", nil, "_Help"],
  ["New", Gtk::Stock::NEW, "_New", "<control>N",
    "Create a new file", Proc.new {@editor.contents_buffer.set_text("")}],
  ["Open", Gtk::Stock::OPEN,
    "_Open...", "<control>O", "Open a file", callback_open],
  ["SaveAs", Gtk::Stock::SAVE,
    "Save _As...", "<control>S", "Save to a file", callback_save],
  ["Quit", Gtk::Stock::QUIT,
    "_Quit", "<control>Q", "Quit", callback_quit],
```

```
      ["About", Gtk::Stock::ABOUT,
        "_About", nil, "About", callback_about],
    ]

    ui_info = %Q[
<ui>
  <menubar name='MenuBar'>
    <menu action='FileMenu'>
      <menuitem action='New'/>
      <menuitem action='Open'/>
      <menuitem action='SaveAs'/>
      <separator/>
      <menuitem action='Quit'/>
    </menu>
    <menu action='HelpMenu'>
      <menuitem action='About'/>
    </menu>
  </menubar>
  <toolbar  name='ToolBar'>
    <toolitem action='New'/>
    <toolitem action='Open'/>
    <toolitem action='SaveAs'/>
    <toolitem action='Quit'/>
    <separator action='Sep1'/>
    <toolitem action='About'/>
  </toolbar>
</ui>]

    actiongroup = Gtk::ActionGroup.new("Actions")
    actiongroup.add_actions(actions)

    uimanager = Gtk::UIManager.new
    uimanager.insert_action_group(actiongroup, 0)
    add_accel_group(uimanager.accel_group)

    vbox = Gtk::VBox.new
    uimanager.add_ui(ui_info)
    vbox.pack_start(uimanager["/MenuBar"], false, false)
    vbox.pack_start(uimanager["/ToolBar"], false, false)
    add(vbox)

    frame = Gtk::Frame.new
    frame.shadow_type = Gtk::SHADOW_NONE
    vbox.pack_start(frame, true, true, 0)

    frame.add(@editor)
  end
end
```

5.3. Класс область редактирования с подстветкой

Элемент управления, отвечающий за редактирование и подсветку кода.

```
require 'gtk2'
require 'source_parser'

class HighlightedEditor < Gtk::ScrolledWindow
  attr_accessor :contents_buffer
  attr_accessor :contents_view

  def initialize
    super
    self.set_policy(Gtk::POLICY_AUTOMATIC, Gtk::POLICY_AUTOMATIC)
    self.shadow_type = Gtk::SHADOW_IN
    @parser = SourceParser.new
    @contents_buffer = Gtk::TextBuffer.new
    @contents_buffer.create_tag('comment', {'foreground' => 'gray'})
    @contents_buffer.create_tag('const', {'foreground' => 'blue'})
    @contents_buffer.create_tag('string', {'foreground' => 'ForestGreen' })
    @contents_buffer.create_tag('reserved',
      {'foreground' => 'black',
      'weight' => Pango::FontDescription::WEIGHT_BOLD})

    @contents_buffer.signal_connect('changed') do |buffer|
      highlight
    end

    @contents_view = Gtk::TextView.new
    @contents_view.set_buffer(@contents_buffer)
    @contents_view.modify_font(Pango::FontDescription.new('Monospace 8'))
    @contents_view.set_wrap_mode(Gtk::TextTag::WRAP_NONE)

    self.add(@contents_view)
  end

  def highlight
    start_iter = @contents_buffer.start_iter
    end_iter = @contents_buffer.end_iter

    @contents_buffer.remove_all_tags(start_iter, end_iter)
    str = @contents_buffer.get_text(start_iter, end_iter, true)

    @parser.parse(str, start_iter.offset) do |tag, start, last|
      @contents_buffer.apply_tag(
        tag.to_s,
        @contents_buffer.get_iter_at_offset(start),
        @contents_buffer.get_iter_at_offset(last))
    end
  end
end
```

5.4. Анализатор исходного текста

Класс, анализирующий текст и определяющий в нем специальные конструкции, которые можно выделить, например строки, константы и т.д.

```
class SourceParser
  RESERVED_WORDS = %w(
    alias   and     BEGIN   begin   break   case    class   def     while
    do      else    elsif   END     end     ensure  false   for     if
    in      module  next    nil     not     or      redo    rescue  retry
    return  self    super   then    true    undef   unless  until   when
    yield)
  RESERVED_WORDS_PATTERN =
    Regexp.compile(/(^|\s+)(#{RESERVED_WORDS.collect { |pat|
      Regexp.quote(pat)
    }.join('|')})(\s+|$)/)

  def parse(str, index = 0)
    until str.empty?
      tag = nil

      case str
      when /[^'"\/]#.*$/
        tag = :comment
      when /".+?"/, /'.+?'/
        tag = :string
      when RESERVED_WORDS_PATTERN
        tag = :reserved
      when /[A-Z][A-Za-z0-9_]+/
        tag = :const
      end

      if tag
        parse($~.pre_match, index) do |*args|
          yield(*args)
        end
        yield(tag, index + $~.begin(0), index + $~.end(0))
        index += (str.length - $~.post_match.length)
        str = $~.post_match
      else
        index += str.length
        str = ''
      end
    end
  end
end
```

6. Литература

1. Ruby-GNOME2 Project Website
   (http://ruby-gnome2.sourceforge.jp/hiki.cgi)

2. Ruby Standard Library Documentation
   (http://www.ruby-doc.org/stdlib)

3. Fox Toolkit (http://www.fox-toolkit.org)

4. FXRuby (http://www.fxruby.org)