**министерство образования и науки российской федерации**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Тихоокеанский государственный университет»**

Факультет Факультет компьютерный и фундаментальных наук

Кафедра Кафедра программного обеспечения вычислительной техники и автоматизированных

Направление 09.03.04 Программная инженерия

(шифр, наименование)

Профиль Разработка программно-информационных систем

ДОПУСТИТЬ К ЗАЩИТЕ

Завкафедрой \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись ФИО

\_\_\_\_\_\_\_\_\_\_\_

дата

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

Тема Написание генератора отчетов с использованием библиотек Qt 5

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись ФИО

\_\_\_\_\_\_\_\_\_\_\_

дата

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись ФИО

\_\_\_\_\_\_\_\_\_\_\_

дата

Нормоконтролёр \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись ФИО

\_\_\_\_\_\_\_\_\_\_\_

дата

**Консультанты:**

По \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись ФИО

\_\_\_\_\_\_\_\_\_\_\_

дата

По \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись ФИО

\_\_\_\_\_\_\_\_\_\_\_

дата

По \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись ФИО

\_\_\_\_\_\_\_\_\_\_\_

дата

По \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись ФИО

\_\_\_\_\_\_\_\_\_\_\_

дата

**Хабаровск – 2016 г.**

РЕФЕРАТ

Выпускная квалификационная работа содержит 72 страницы текстового документа формата А4, включающего 13 рисунков, 1 приложение и 26 источников информации.

ОТЧЕТ, ГЕНЕРАТОР ОТЧЕТОВ, QT, CUTEREPORT, LIMEREPORT, ТЕГ, EXARO, ЭКСПОРТ.

Целью работы являлось создать генератор отчетов на языке С++ с использованием библиотек Qt версии 5.

Объект исследования – автоматизация процесса генерации отчетов.

С помощью разработанного приложения можно автоматизировать процесс генерации отчетов и их экспорта в различные форматы.

Содержание

[Введение 5](#_Toc453966980)

[1 Постановка задачи 6](#_Toc453966981)

[1.1 Система тегов 6](#_Toc453966982)

[2 Описание программного средства 16](#_Toc453966983)

[2.1 Диаграмма классов 16](#_Toc453966984)

[2.2 Описание классов 17](#_Toc453966985)

[3 Технология эксплуатации 42](#_Toc453966986)

[3.1 Руководство для администратора 42](#_Toc453966987)

[3.2 Руководство для программиста 44](#_Toc453966988)

[4 Примеры использования 46](#_Toc453966989)

[4.1 Текст и данные из базы данных 46](#_Toc453966990)

[4.2 Заголовок и изображения 47](#_Toc453966991)

[4.3 Группировка данных 49](#_Toc453966992)

[4.4 Форматирование текста 51](#_Toc453966993)

[Заключение 53](#_Toc453966994)

[Список использованных источников 54](#_Toc453966995)

[Приложения А 57](#_Toc453966996)

Введение

На рынке представлено много генераторов отчетов, среди них есть такие как: eXaro [1, 2], LimeReport [5, 6], CuteReport [3, 4] и т.д. Если брать каждый по отдельности, то мы столкнемся с рядом сложностей при выборе конкретного. На наш выбор повлияют различные факторы такие как: цена, лицензия, функционал, язык, актуальность.

eXaro – несомненно хороший генератор отчетов, но главным недостатком является его отсталость. Его разработка закончена несколько лет назад и можно сказать что он устарел, не смотря на его хороший функционал. Генератор написан на языке С++ с использованием уже устаревших версии библиотеки Qt(Qt4) [7].

LimeReport – проект в стадии разработки, к его недостаткам можно отнести его не завершённость, отсутствие хорошей документации. Язык разработки С++ и используются актуальные библиотеки фреймворка Qt.

CuteReport – мощный генератор отчетов, написан на языке С++ и использует актуальные библиотеки фреймворка Qt. Отсутствует хорошая документация, сложность в интеграции и настройке.

С учетом того что на рынке есть программные продукты, но возникают сложности в их использовании, настройке, интеграции. Возникает необходимость в программном решении, которое совмещало в себе критерии: бесплатность, документацию, функционал, легкость интеграции и модификации. По этой причине была выбрана данная тема выпускной квалификационной работы.

1. Постановка задачи

Задачей выпускной квалификационной работы было создать генератор отчетов с возможность использовать в отчетах текстовые и графические данные при необходимости группировать их, так же использовать переменные и параметры при генерации отчетов. Организовать экспорт отчетов в различные форматы. Структура должна отчета должна задаваться тегами, на вход должен подаваться XML [9, 10, 11] документ.

* 1. Система тегов

Структура тегов представлена рисунке 1.1

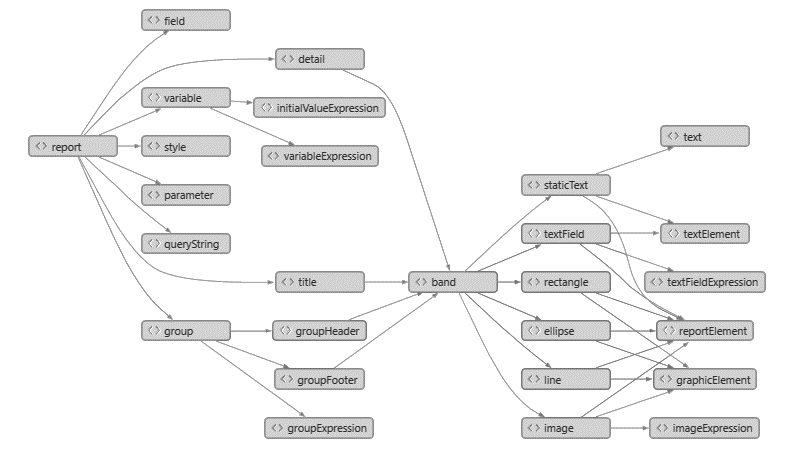


Рисунок 1.1 – Структура тегов

Теги разрабатывались на основе книги [8].

Теги field, initValueExpression, variableExpression, parameter, queryString, groupExpression, text, textFieldExpression и imageExpression используют CDATA разметку.

CDATA- это часть содержания элемента, которая помечена для парсера как содержащая только символьные данные, а не разметку. Начинается последовательностью символов <![CDATA[ заканчивается символами]]>.

Подробное описание каждого тега приведено ниже. Тип данных указывается после двоеточия, используются следующие типы:

- String – строковый тип;

- Integer – 32-х битовое знаковое целое число;

- Float – 32-х битовое знаковое число с плавающей точкой;

- Double – 64-х битовое число с плавающей точкой.

* + 1. Тег «report»

Тег «report» - это основной, корневой элемент отчета. Является контейнером для всех тегов и имеет обязательный атрибут name: String – имя отчета.

Необязательные атрибуты:

- pageWidth:Integer – ширина страницы, по умолчанию 595;

- pageHeight:Integer - высота страницы, по умолчанию 842;

- orientation:String – ориентация страницы, возможные значения: «Книжная», «Альбомная» и по умолчанию «Книжная».

* + 1. Тег «field»

Тег «field» описывает поле отчета. Представляет собой единственный способ отображения данных из источников данных в отчете шаблона, и использовать эти данные в выражениях отчетов, чтобы получать желаемый результат.

При использовании запросов SQL в отчете, необходимо убедиться, что столбец для каждого поля, полученный после выполнения запроса, имеет тоже самое имя и тот же тип данных, что и поле, которое отображает его.

Имеет два обязательны атрибута:

- name:String – имя атрибута, позволяет ссылаться на поле в отчете по имени;

- class:String – имя класса поля, принимает значения: String и Integer. По умолчанию - String.

* + 1. Тег «variable»

Тег «variable» является контейнером для variableExpression и initValueExpression. Переменные упрощают шаблон отчета путем выделения в одной части выражения, которое широко используется во всем шаблоне отчета. Они могут выполнять расчеты, основанные на соответствующей формуле(выражении). В своем выражении(формуле) переменная может использовать другие переменные, поля или параметры. Переменные изменяются при обработке записи из источника данных в том порядке, в котором они объявлены.

Атрибуты:

- name:String – имя переменной, является обязательным атрибутом и позволяет ссылаться на переменную по этому имени;

- class:String – тип данных, которому принадлежит значение переменной. Принимает значения: String, Integer, Float. По умолчанию - String;

- resetType:String – периодичность установки исходного значения. Возможные значения: None, Report, Page, Column, Groupю. По умолчанию – Report;

- None – никогда не инициализируется начальным значением. Содержит значение, полученное путем вычисления выражения переменной;

- Report – инициализируется начальным значением (initialValueExpression) один раз в начале заполнения отчета;

- Page – инициализируется заново в начале каждой страницы;

- Column – инициализируется заново в начале каждого нового столбца;

- Group – инициализируется заново каждый раз, когда задается новая группа;

- resetGroup:String – содержит имя группы и работает только в сочетании с атрибутом resetType, значение которого будет Group;

- incrementType:String – периодичность приращения переменной. Принимает значения: None, Report, Page, Column, Group.По умолчанию – None;

- None – переменная инкрементируется с каждой записью;

- Report – переменная никогда не изменяется;

- Page – инкрементируется с каждой новой страницей;

- Column – инкрементируется с каждым новым столбцом;

- Group – инкрементируется каждый раз, когда задается новая группа;

- incrementGroup:String – содержит имя группы и работает только в сочетании с атрибутом incrementType, значение которого будет Group;

- calculation:String - агрегатная функция, принимает значения: Count, Sum, Average, Lowest, Highest;

- Count – количество;

- Sum – сумма;

- Average – среднеарифметическое;

- Lowest – минимальный элемент;

- Highest – максимальный элемент.

* + 1. Тег «variableExpression»

Тег «variableExpression»:String – задает значение переменной.

* + 1. Тег «initValueExpression»

Тег «initValueExpression»:String – задает начальное значение переменной.

* + 1. Тег «style»

Тег «style» определят стиль. Позволяет единожды определить некоторый набор свойств элементов, а затем использовать этот набор в любом блоке отчета. Стиль применяются для элемента reportElement путем указания имени стиля в качестве атрибута style=”name”, в противном случае применяется стиль по умолчанию.

Имеет обязательный атрибут name:String – имя стиля.

Необязательные атрибуты:

- isDefault:String – будет ли этот стиль использоваться как стиль по умолчанию, принимает значения: true, false. По умолчанию - false;

- fontSize:Integer – размер шрифта;

- fontColor:String – цвет шрифта, принимает значения: black, blue, gray, green, red, yeallow, while. По умолчанию – black;

- fontName:String – имя шрифта;

- isBold:String – будет ли текст «жирным», принимает значения: true, false. По умолчанию - false;

- isItalic;String – будет ли текст «курсивным», принимает значения:true, false. По умолчанию - false;

* + 1. Тег «parameter»

Тег «parameter» определяет параметр. Это ссылка на объекты, которые передаются при процессе заполнения отчета к движку генератора отчета.

Атрибуты:

- name:String - имя параметра;

- class:String – тип значения параметра, принимает значения: String, Integer.

* + 1. Тег «queryString»

Тег «queryString» описывает SQL запросы. Используется в качестве источника данных в отчетах.

* + 1. Тег «group»

Тег «group» является контейнером для groupFooter, groupHeader и groupExpression. Группа – это гибкий способ организации данных в отчете. Они представляют собой последовательность записей, имеющие общее значения в заданных полях.

В отчете может быть несколько групп. Порядок групп, заявленный в шаблоне отчета важен, так как группы содержат друг друга.

Для группы должно быть задано значение, которое является конструктором данной группы. Когда в процессе обработке записей из источника данных значения выражения изменяется, происходит вставка соответствующих секций groupFooter и groupHeader в результирующий документ.

Механизм группировки довольно прост – для каждой записи из источника данных вычисляется groupExpression и сравнивается с groupExpression предыдущей записи. В случае разных значений, закрывается прошлая группа и открывается следующая.

Атрибут name:String – имя, является обязательным. Название однозначно определяет группу и может быть использовано для других атрибутов, когда необходимо сослаться на конкретную группу в отчете.

Группировка данных работает, как задумано только тогда, когда записи в источнике данных уже упорядочены в соответствии с групповым выражением, используемым в отчете.

* + 1. Тег «groupExpression»

Тег «groupExpression» описывает выражение, по которому будет производиться группировка.

* + 1. Тег «groupHeader»

Тег «groupHeader» является контейнером для тега band. Заголовок группы, то что будет напечатано перед первым элементом группы. Этот раздел отмечает начало новой группы в итоговом документе. Он вставляется в документ каждый раз, когда значение groupExpression изменяется во время обработки записей из источника.

* + 1. Тег «groupFooter»

Тег «groupFooter» является контейнером для тега band. Подвал группы, то что будет напечатано после последнего элемента группы, отмечает конец группы в итоговом документе. Вставляется при изменении значения groupExpression.

* + 1. Тег «title»

Тег «title» является контейнером тега band. Название отчета, отображается один раз в начале отчета.

* + 1. Тег «detail»

Тег «detail» является контейнером тега band. Ключевая и самая важная часть отчета, можно назвать блок «телом» отчета. В блоке detail содержится основная информация, для каждой записи в источнике данных. Может содержать несколько полос элементов band.

* + 1. Тег «band»

Тег «band» является контейнером тегов: staticText, textField, rectangle, ellipse, line, image. Блок, в котором перечисляются элементы отчета.

Атрибут height:Integer – высота полосы, не является обязательным, по умолчанию равен нулю.

* + 1. Тег «staticText»

Тег «staticText» является контейнером тегов: text, textElement и reportElement. Постоянный текст, который не зависит от каких-либо источников данных.

* + 1. Тег «textElement»

Тег «textElement» определяет выравнивание текста, если стоит перед text.

Атрибуты:

- textAligment:String – горизонтальное выравнивание текста, принимает значения: Left, Center, Right. Является обязательным атрибутом;

- textVAligment:String – вертикальное выравнивание текста. Не обязателен, принимает значения: Top, Middle, Bottom.

* + 1. Тег «text»

Тег «text» содержит текст, символы для отображения.

* + 1. Тег «textField»

Тег «textField» является контейнером тегов: textElement, textFieldExpression, reportElement. В отличии от статических текстовых элементов, которые не изменяют содержание их текста, текстовые поля имею ассоциированное выражение. Которое вычисляется с каждым обращением к полю в источнике данных и отображается.

* + 1. Тег «rectangle»

Тег «rectangle» является контейнером тегов: reportElement, graphicElement. Прямоугольник, является простейшим элементом отчета.

* + 1. Тег «line»

Тег «line» является контейнером тегов: reportElement, graphicElement. Линия – при отображении рисует одну из двух сторон прямоугольника. Рисование вертикальных или горизонтальных линий выбирается установкой width и height.

Атрибут:

- direction: String определяет, какую из двух сторон прямоугольника стоит отобразить, принимает значения: TopDown, BottomUp. По умолчанию - TopDown.

Для вертикальных линий, направление не важно.

* + 1. Тег «ellipse»

Тег «ellipse» является контейнером тегов: reportElement, graphicElement. Эллипсы - основной графический элемент, рисуется в теги reportElement вписываясь в него.

* + 1. Тег «image»

Тег «image» является контейнером тегов: reportElement, graphicElement, imageExpression. Отображает изображения из источника данных в итоговом документе.

* + 1. Тег «imageExpression»

Тег «imageExpression» возвращает значение которое является источником для отображаемого изображения. Возвращает значения: Sting.

* + 1. Тег «graphicElement»

Тег «graphicElement» расширяет функционал тегов, предоставляя новые возможности.

Атрибуты:

- stretchType:String – элементы получают возможность адаптировать свою высоту в зависимости от высоты других связанных с ними элементов черед элемент группировки. Возможные значения: NoStretch, RelativeToTallestObject, RelativeToBandHeight.

- pen:String – указывает тип границы вокруг графического элемента. По умолчанию граница зависит от типа элемента. Возможные значения: None, Thin, 1Point, 2Point, 3Point, 4Point, Dotted;

- fill:String - определяет стиль фона графических элементов. В настоящее время поддерживает только твердый стиль заливки, значение: solid.

* + 1. Тег «reportElement»

Тег «reportElement» является первым элементом каждого из подтегов тега band. Определяет положение и размер элемента перед которым указан. Если не указать элемент может не отображаться.

Атрибуты:

Обязательные атрибуты задают положение, ширину и высоту.

- х:Integer – координата по х, задает начало блока;

- у:Integer – координата по у, задает начало блока;

- width:Integer – ширина элемента;

- height:Integer – высота элемента.

Не обязательный атрибут: style:String – стиль.

* + 1. Тег «textFieldExpression»

Тег «textFieldExpression» содержит возвращаемый элемент текстового поля с описанием класса элемента.

Обязательный атрибут class – тип класса, принимает значения: String, Float, Integer, Double. По умолчанию – String.

1. описание программного средства
   1. Диаграмма классов

Диаграмма классов программы представлена на рисунке 2.1



Рисунок 2.1 – Диаграмма классов

При разработке использовалась справочная информация [12], использовались подход ООП [13, 14].

* 1. Описание классов
     1. Класс Engine

Класс Engine – движок генератора отчетов. Он отвечает за связь программного продукта и самой библиотеки. Является главным файлом, отвечающим за работу и преобразование отчета. Все параметры и методы публичны.

Методы:

* bool open( const QString & path ) - загружает макет отчета из файла по пути path и запускает парсер. Возвращает true если файл открыт и false в противном случае;
* void close() - Выгружает из оперативной памяти данные об отчете;
* bool setParameters( const QMap< QString, QVariant > & map ) - передает карту параметров в отчет, для успешной передачи параметров не обязательно загружать их описание с макетом отчета, в случае успеха возвращает true и false при ошибке. map карта параметров в формате (<имя параметра>, <значение параметра>);
* bool setConnection( const QSqlDatabase & connection ) - устанавливает источник данных для отчета из БД в случае успеха возвращает true, false при ошибке. connection соединение с БД;
* bool setDataSource( const QMap <QString, QVector <QVariant> > & columnsSet) - устанавливает источник данных для отчета из таблицы полей. columnsSet набор данных в формате (<имя столбца>, <вектор значений столбца>). Возвращает true при успехе и false при ошибке;
* bool setQuery( const QString & query ) - устанавливает запрос - query, по которому будут заполнятся поля отчета.Возвращает true при успехе и false при неудаче;
* bool addScript( const QString & script ) - добавить скрипт - script для отчета. Возвращает true при успехе и false при ошибке;
* bool setDataModel( const QAbstractItemModel & model ) - выбирается модель данных - model. Возвращает true при успехе и false при ошибке;
* bool createPDF( const QString & path ) - создает PDF документ отчета по указанному пути path с указанием имени сохраняемого файла. Возвращает true при успехе и false при ошибке;
* bool createHTML( const QString & path ) - создает HTML документ отчета по указанному пути path с указанием имени сохраняемого файла. Возвращает true при успехе и false при ошибке;
* QWidgetPtr createWidget() - создает виджет отчета для вывода на экран. Возвращает умный указатель на виджет;
* QWidgetPtr createLayout() - создает слой отчета для вывода на экран. Возвращает умный указатель на слой;
* bool print() - создает виджет отчета для вывода на экран. Возвращает true при успехе и false при ошибке;
* bool isOpened() const - возвращает статус m\_isOpened;
* ReportPtr getReport() const - возвращает умный указатель на структуру отчета;
* const QString getLastError() const - возвращает ошибку в строковом формате;
* void drawPreview( QPrinter \* printer ) - создает окно предпросмотра. printer указатель на отрисовщик;
* bool prepareDB() - выбирается база данных, возвращает true при успешном соединении;
* bool prepareDataSource( const QMap< QString, QVector< QVariant > > & source ) – извлекает карту с данными из source;
* void executeQueries( const QStringList & queries ) - извлекает queries и передает данные в prepareDataSource.

Параметры:

* bool m\_isOpened – статус открытия макета отчета;
* QString m\_lastError, m\_compiledPath - строковые данные содержат ошибку и полный путь к макету отчета;
* ReportPtr m\_report – умный указатель на структуру отчета;
* QVector< QString > m\_scripts - вектор содержащий скрипты отчета;
* QSqlDatabase m\_dbConnection - экземпляр класса, для подключения к базе данных.
  + 1. Класс ConverterToQWidget

Класс ConverterToQWidget отвечает за предварительное отображение макета отчёта. И предварительный просмотр сгенерированного отчета.

Публичные методы:

* bool convert( WidgetType type = WidgetType::Report ) – конвертирует m\_report в виджет по типу type;
* bool isReport() const – возвращает m\_type равным отчет;
* bool isLayout() const – возвращает m\_type равным макет;
* WidgetType getType() const – возвращает текущий m\_type;
* const QWidgetPtr getQWidget() const - возвращает текущий m\_qwidget;
* const QWidgetPtr getPage( int i ) const – возвращает умный указатель на страницу по значению i;
* const QVector< QWidgetPtr > getPages() const – возвращает вектор с умными указателями на страницы;
* const QString getLastError() const – возвращает строку, содержащую описание последней ошибки.

Закрытые методы:

* void addVerticalBorder( QBoxLayout \* parent, const QMargins & margins, int height ) – создается вертикальный блок. parent – указатель на объект макета, margins – отступы, height – высота;
* void addEmptySection( QBoxLayout \* parent, const QMargins & margins ) – создает пустую секцию. parent – указатель на объект макета, margins – отступы, height – высота;
* QFrame \* addSectionLayout( QBoxLayout \* parent, const QMargins & margins, int height ) – добавляет секцию на макет используя addVerticalBorder и addEmptySection. parent – указатель на объект макета, margins – отступы, height – высота;
* void addPage() – добавляет новую страницу;
* bool createQWidget( const ReportPtr & report ) – создает виджет по умному указателю структуры report, возвращает true в случаи успеха, при ошибке false;
* bool createSection( QWidget \* parent, const SectionPtr & section, int i ) – создает секции на макете. parent – указатель на объект макета, section – умный указатель на секцию, i – индекс секции;
* bool createBands( QWidget \* parent, const SectionPtr & section ) – создается блок. parent – указатель на обьект макета, section – умный указатель на секцию;

Закрытые параметры:

* ReportPtr m\_report – умный указатель на структуру отчета;
* QString m\_lastError – строковый тип содержит описание ошибки;
* QWidgetPtr m\_qwidget – умный указатель на виджет;
* WidgetType m\_type – тип виджета;
* QVector< QWidgetPtr > m\_pages – вектор хранящий умные указатели на страницы;
* int m\_currentHeight – числовой, содержит значение высоты.
  + 1. Класс ConverterToPDF

Класс ConverterToPDF экспортирует отчет в формат PDF [15, 16, 17]. Все функции данного класса, возвращающие bool, возвращают true в случае успеха либо false при ошибке. Поддерживает установку количества точек да дюйм (dpi).

Публичные методы:

* bool convert( const QString & path ) – запускает процесс конвертации по пути path;
* void setDPI( int dpi ) – устанавливает значение dpi;
* int getDPI() const – возвращает текущее значение dpi;
* const QString getLastError() const – возвращает строку, содержащую описание последней ошибки.

Закрытые методы:

* bool create( const QString & path ) – создает документы по пути path.

Закрытые параметры:

* ReportPtr m\_report – умный указатель на структуру отчета;
* QString m\_lastError – строковый тип, содержит описание ошибки;
* int m\_dpi – числовой содержит значение dpi.
  + 1. Класс ConverterToHTML

Класс ConverterToHTML экспортирует отчет в HTML [18, 19] формат.

Публичные методы:

* bool convert() – создает новый документ HTML, возвращает true при успехе и false при ошибке;
* bool convert( const QString & path ) – запускает процесс конвертации по пути path, возвращает true при успехе и false при неудаче;
* const QString getLastError() const - возвращает строку, содержащую описание последней ошибки;
* const QString getHTML() const – возвращает текущее значение HTML;

Закрытые методы:

* bool createHTML() – наполняет содержимым созданный документ по пути path, возвращает true при успехе и false при ошибке;
* void drawShapes(QSharedPointer< Band > band, QString &elementStr, int index) – рисует графические элементы в документе. band – тип элемента, elementStr – строка содержащее текущую разметку документа HTML, index – индекс изображения для загрузки.

Закрытые параметры:

* ReportPtr m\_report - умный указатель на структуру отчета;
* QString m\_lastError – строковый тип, содержит описание ошибки;
* QString m\_html – строковый тип, содержит разметку HTML документа.
  + 1. Класс Report

Класс Report реализует тег report. Один из основных классов.

Публичные методы:

* void setDefaultStyle( const StylePtr & style ) – устанавливает style по умолчанию. StylePtr – умный указатель на стиль;
* const StylePtr getDefaultStyle() const – возвращает умный указатель на стиль по умолчанию;
* void addStyle( const QString & name, const StylePtr & style ) – добавляет стиль. name - имя стиля, StylePtr указатель на стиль;
* const StylePtr getStyle( const QString & name ) const – возвращает умный указатель на стиль по имени, name – имя стиля;
* const QMap< QString, StylePtr > getStyles() const – возвращает карту указателей на стили;
* void addGroup( const QString & name, const GroupPtr & group ) – добавляет группу. name - имя группы, group умный указатель на группу;
* const GroupPtr getGroup( const QString & name ) const - возвращает умный указатель на группу по имени, name – имя группы;
* const QMap< QString, GroupPtr > getGroups() const – возвращает карту указателей на группы;
* void setQuery( const QString & query ) – устаналивает queryString, query значение queryString;
* const QString getQuery() const – возвращает queryString в строковом формате;
* void addVariable(const QString & name, const VariablePtr & variable) - добавляет переменную. name имя переменной, VariablePtr умный указатель на переменную;
* const VariablePtr getVariable(const QString & name) const - возвращает умный указатель на переменную по имени, name – имя переменной;
* const QMap< QString, VariablePtr > getVariables() const - возвращает карту указателей на переменные;
* void setField( const QString & name, const FieldPtr & field ) - устаналивает field, FieldPtr умный указатель на обьект field; name – имя field;
* const FieldPtr getField( const QString & name ) const - возвращает умный указатель на field по имени – name;
* const QMap< QString, FieldPtr > getFields() const - возвращает карту указателей на field;
* void setFieldData( const QString & name, const QVector< QVariant > & data ) – устаналивает данные с field. name имя field, data вектор данных field;
* void setTitle( const TitlePtr & title ) – устаналивает заголовок отчёта. TitlePtr – умный указатель на обьект title;
* const TitlePtr getTitle() const – возвращает умный указатель на обьект title;
* void setDetail( const DetailPtr & detail ) - устаналивает detail. TitlePtr – умный указатель на обьект detail;
* const DetailPtr getDetail() const - возвращает умный указатель на обьект detail;
* void setParameter( const QString & name, const QVariant & value ) – устанавливает значение параметра. name имя, value значение;
* void setParameters( const QMap< QString, QVariant > & parameters ) – устанавливает карту параметров.
* const QVariant getParameter( const QString & name ) const – возвращает значение параметра по имени name;
* const QMap< QString, QVariant > getParameters() const – возвращает карту параметров с именем и значениями;
* int getRowCount() const – возвращает количество строк;
* void setOrientation( QPrinter::Orientation orientation ) – устанавливает ориентацию отчета orientation – книжная/альбомная;
* QPrinter::Orientation getOrientation() const – возвращает ориентацию отчета;
* void setSize( const QSize & size ) – устанавливает размер отчета. size – размер;
* const QSize getSize() const – возвращает размер отчета;
* void setWidth( int width ) – устанавливает ширину отчета. width – ширина;
* int getWidth() const – возвращает ширину;
* void setHeight( int height ) – устанавливает высоту отчета. height – высота;
* int getHeight() const – возвращает высоту;
* void setLeftMargin( int left ); - устанавливает значение отступа слева. left – значение отступа;
* int getLeftMargin() const – возвращает значение отступа слева;
* void setTopMargin( int top ) - устанавливает значение отступа сверху. top – значение отступа;
* int getTopMargin() const – возвращает значение отступа сверху;
* void setRightMargin( int right ) - устанавливает значение отступа справа. right – значение отступа;
* int getRightMargin() const – возвращает значение отступа справа;
* void setBottomMargin( int bottom ) - устанавливает значение отступа снизу. bottom – значение отступа;
* int getBottomMargin() const – возвращает значение отступа снизу;
* void setMargins( int left, int top, int right, int bottom ) – устанавливает значения отступа от краев. left слева, top сверху, right справа, bottom снизу;
* void setMargins( const QMargins & margins ) – устанавливает карту отступов. margins отступы;
* const QMargins getMargins() const – возвращает карту отступов;
* bool isDetailHasGroupHeader(int detailNum, const QString &groupField) – определяет нужно ли открывать GroupHeader. detailNum номер строки, groupField – значение field;
* bool isDetailHasGroupFooter(int detailNum, const QString &groupField) - определяет нужно ли открывать GroupFooder. detailNum номер строки, groupField – значение field;
* void reorderByGroups() – перебирает группы, если их несколько;
* const QString getFieldFromGroupExpression(const QString & expression) – возвращает значение field по значению expression;
* const QVector<bool> getGroupVec() – возвращает вектор строк со значениями группы;
* int getGroupIndexFromField(const QString & field) – получение значение индекса группы по значению field;
* GroupPtr getGroupByIndex(int index) – возвращает умный указатель на группу по индексу index;

Закрытые методы:

* void swapRows(int row1, int row2, QVector<bool> & vec) – меняет строки местами в таблице; row1 и row2 строки;

Закрытые параметры:

* QPrinter::Orientation m\_orientation – ориентация;
* QSize m\_size – значение размера отчета;
* QMargins m\_margins – карта отступов;
* StylePtr m\_defaultStyle – умный указатель на стиль по умолчанию;
* QMap< QString, StylePtr > m\_styles – карта стилей;
* QString m\_query – содержит значение query;
* QMap< QString, FieldPtr > m\_fields – карта значений field;
* QMap< QString, GroupPtr > m\_groups – карта значений групп;
* TitlePtr m\_title – умный указатель на заголовок отчета;
* DetailPtr m\_detail – умный указатель на блок detail;
* QMap< QString, QVariant > m\_parameters – карта значений параметров;
* QVector<bool> m\_group\_vec – вектор значений групп;
  + 1. Класс ParserFromXML

Класс ParserFromXML считывает (парсит) отчет и создает из него внутреннюю структуру данных report.

Публичные методы:

* bool parse( const QString & path ) – парсит отчет по пути path;
* const ReportPtr getReport() const – возвращает указатель на внутреннюю структуру отчета. Указатель может быть пустым.
* const QString getLastError() const – возвращает строковое описание ошибки;
* const QString getLog() const - возвращает процесс лог процесса парсинга, используется для отладки.

Закрытые методы:

* bool getValue( QXmlStreamReader & reader, QString & data ) – считывает xml элемент в data. reader обьект QXmlStreamReader, data считанные данные;
* bool getAttribute( QXmlStreamReader & reader, const QString & name, QString & data, AttributeOption option ) – записывает в data значения атрибута с именем name текущего тега, reader обьект QXmlStreamReader, option указывает обязателен ли атрибут, data считанные данные;
* bool getRequiredAttribute( QXmlStreamReader & reader, const QString & name, QString & data ) – вызывает функцию getAttribute с переданными параметрами и с параметром option («обязательный атрибут»), reader обьект QXmlStreamReader;
* bool getOptionalAttribute( QXmlStreamReader & reader, const QString & name, QString & data ) - вызывает функцию getAttribute с переданными параметрами и с параметром option («опциональный атрибут»), reader обьект QXmlStreamReader;
* bool goToElementEnd( QXmlStreamReader & reader ) – устанавливает курсор reader к началу следующего тега или к концу документа, reader обьект QXmlStreamReader;
* bool parseChilds( QXmlStreamReader & reader, const ObjectPtr & object ) – парсит дочерние теги объекта object, object – родительский обьект отчета, reader обьект QXmlStreamReader;
* bool parseDocument( const QString & text ) – начинает парсить документ по имении name, по пути содержащейся в text;
* bool parseReport( QXmlStreamReader & reader, const ReportPtr & report ) – парсит тег report, report – указатель на обьект отчета, reader обьект QXmlStreamReader;
* bool parseStyle( QXmlStreamReader & reader, const ReportPtr & report ) – парсит тег style, report – указатель на обьект отчета, reader обьект QXmlStreamReader;
* bool parseQueryString( QXmlStreamReader & reader, const ReportPtr & report )- парсит тег qureySting, report – указатель на обьект отчета, reader обьект QXmlStreamReader;
* bool parseField( QXmlStreamReader & reader, const ReportPtr & report ) - парсит тег field, report – указатель на обьект отчета, reader обьект QXmlStreamReader;
* bool parseVariable(QXmlStreamReader & reader, const ReportPtr & report) - парсит тег variable, report – указатель на обьект отчета, reader обьект QXmlStreamReader;
* bool parseVariableExpression(QXmlStreamReader & reader, const GroupPtr & variable) - парсит тег style, variable – указатель на переменную, reader обьект QXmlStreamReader;
* bool parseGroup( QXmlStreamReader & reader, const ReportPtr & report ) - парсит тег group, report – указатель на обьект отчета, reader обьект QXmlStreamReader;
* bool parseGroupExpression( QXmlStreamReader & reader, const GroupPtr & group ) - парсит тег groupExpression, group – указатель на группу, reader обьект QXmlStreamReader;
* bool parseGroupHeader( QXmlStreamReader & reader, const GroupPtr & group ) - парсит тег groupHeader, group – указатель на группу, reader обьект QXmlStreamReader;
* bool parseGroupFooter( QXmlStreamReader & reader, const GroupPtr & group ) - парсит тег groupFooter, group – указатель на группу, reader обьект QXmlStreamReader;
* bool parseTitle( QXmlStreamReader & reader, const ReportPtr & report ) - парсит тег title, report – указатель на обьект отчета, reader обьект QXmlStreamReader;
* bool parseDetail( QXmlStreamReader & reader, const ReportPtr & report ) - парсит тег detail, report – указатель на обьект отчета, reader обьект QXmlStreamReader;
* bool parseBand( QXmlStreamReader & reader, const SectionPtr & section ) - парсит тег band, section секция (объект, содержащий band(ы), detail, title, groupHeader и т.д.), которой принадлежит band, reader обьект QXmlStreamReader;
* bool parseStaticText( QXmlStreamReader & reader, const BandPtr & band ) - парсит тег staticText, band Строка, которой принадлежит объект, reader обьект QXmlStreamReader;
* bool parseTextField( QXmlStreamReader & reader, const BandPtr & band ) ) - парсит тег textField, band Строка, которой принадлежит объект, reader обьект QXmlStreamReader;
* bool parseLine( QXmlStreamReader & reader, const BandPtr & band ) - парсит тег line, band Строка, которой принадлежит объект, reader обьект QXmlStreamReader;
* bool parseRect( QXmlStreamReader & reader, const BandPtr & band ) - парсит тег rectangle, band Строка, которой принадлежит объект, reader обьект QXmlStreamReader;
* bool parseEllipse( QXmlStreamReader & reader, const BandPtr & band ) - парсит тег ellipse, band Строка, которой принадлежит объект, reader обьект QXmlStreamReader;
* bool parseImage( QXmlStreamReader & reader, const BandPtr & band ) - парсит тег image, band Строка, которой принадлежит объект, reader обьект QXmlStreamReader;
* bool parseReportElement( QXmlStreamReader & reader, const WidgetPtr & widget ) – парсит тег reportElement – общие данные о элементе отчета ( положение, размеры и т.д.). widget общий класс для объектов, подлежащих отображению. Для него загружается текущий reportElement, reader обьект QXmlStreamReader;
* bool parseTextElement( QXmlStreamReader & reader, const WidgetPtr & widget ) - парсит тэг textElement - общие данные о текстовом элементе (выравнивание текста и т.д.). widget общий класс для объектов, подлежащих отображению. Для него загружается текущий textElement, reader обьект QXmlStreamReader;
* bool parseFont( QXmlStreamReader & reader, const WidgetPtr & widget ) – парсит тег font – данные о шрифте текста. widget общий класс для объектов, подлежащих отображению. Для него загружается текущий font, reader обьект QXmlStreamReader;
* bool parseText( QXmlStreamReader & reader, const StaticTextPtr & text ) – парсит текст для staticText. text staticText, текущий объект, reader обьект QXmlStreamReader;
* bool parseTextFieldExpression( QXmlStreamReader & reader, const TextFieldPtr & text ) - парсит текст для textField. text textField, текущий объект, reader обьект QXmlStreamReader;
* bool parseImageExpression( QXmlStreamReader & reader, const ImagePtr & image ) - парсит текст (imageExpression) для image, reader обьект QXmlStreamReader;

Закрытые параметры:

* ReportPtr m\_report – умный указатель на элемент структуры отчета;
* QString m\_lastError – содержит последнюю ошибку;
* QTextStream m\_log – лог парсера, используется при отладки;
* QMap< QString, ParseFunc > m\_functions – карта загружаемых функций.
  + 1. Класс Object

Класс Object является базовым классов для любого тега.

Публичные методы:

* Object( const QString & name ) – конструктор, создает обьект с именем. name – имя обьекта;
* void setName( const QString & name ) – устанавливает значение атрибута name;
* const QString getName() const – возвращает значение атрибута name;
* void setTagName( const QString & name ) – устанавливает название тега. name – имя тега;
* const QString getTagName() const – возвращает название тега;
* const QString getLastError() const – возвращает описание ошибки.

Защищенные параметры:

* QString m\_name – имя обьекта;
* QString m\_tagName – имя тега;
* QString m\_lastError – описание ошибки.
  + 1. Класс Style

Класс Style реализует тег style.

Публичные методы:

* bool isDefault() const – возвращает, является ли данный стиль стилем по умолчанию;
* void setAsDefault( bool flag ) – устанавливает, является ли текущий стиль стилем по умолчанию. flag новое значение;
* const QString getFontName() const – возвращает название шрифта;
* void setFontName( const QString & name ) – устанавливает название шрифта name;
* int getFontSize() const – возвращает размер шрифта;
* void setFontSize( int size ) – устанавливает размер шрифта size;
* const QColor getFontColor() const – возвращает цвет шрифта;
* void setFontColor( const QColor & color ) – устанавливает цвет шрифта color;
* bool isBold() const – возвращает, является ли шрифт жирным или нет;
* void setBold( bool flag ) – устанавливает, является ли шрифт жирным или нет. flag новое значение;
* bool isItalic() const – устанавливает, является ли шрифт курсивом или нет;
* void setItalic( bool flag ) – устанавливает, является ли шрифт курсивом или нет. flag новое значение;
* bool isUnderline() const – возвращает, является ли шрифт подчеркнутым или нет;
* void setUnderline( bool flag ) – устанавливает, является ли шрифт подчеркнутым или нет. flag новое значение;
* bool isStrikeThrough() const – возвращает, является ил шрифт перечеркнутым или нет;
* void setStrikeThrough( bool flag ) – устанавливает, является ли шрифт перечеркнутым или нет. flag новое значение;
* const QString getPDFFontName() const – возвращает название шрифта в PDF;
* void setPDFFontName( const QString & name ) – устанавливает название шрифта name в PDF;
* const QString getPDFEncoding() const – возвращает название кодировки в PDF;
* void setPDFEncoding( const QString & encoding ) – устанавливает название кодировки encoding в PDF;
* bool isPDFEmbedded() const – возвращает Embedded в PDF;
* void setPDFEmbedded( bool isEmbedded ) – устанавливает новое значение isEmbedded в PDF;

Закрытые параметры:

* bool m\_isDefault – является ли стилем по умолчанию;
* bool m\_isBold – является ли шрифт жирным;
* bool m\_isItalic – является ли шрифт курсивом;
* bool m\_isUnderline – является ли шрифт подчеркнутым;
* bool m\_isStrikeThrough – является ил шрифт перечеркнутым;
* int m\_fontSize – размер шрифта;
* bool m\_isPDFEmbedded;
* QColor m\_fontColor – цвет шрифта;
* QString m\_fontName – название шрифта;
* QString m\_pdfFontName – название шрифта PDF;
* QString m\_pdfEncoding – название кодировки PDF;
  + 1. Класс Field

Класс Field реализует тег field.

Публичные методы:

* void setClassName( const QString & name ) – устанавливает атрибут className;
* const QString getClassName() const – возвращает атрибут className;
* void setData( const QVector< QVariant > & data ) – устанавливает содержимое field. data – вектор, содержащий данные field;
* QString getData( int row ) – возвращает содержимое field по номеру строки row;
* template< typename T1 > - шаблон типа;
* const T1 getData( int row ) - возвращает содержимое <field>. row номер строки;
* const QVariant getDataVar(int row) - возвращает содержимое <field>. row номер строки;
* void setDataVar(int row, const QVariant & value) – устанавливает значение value в векторе m\_data по номеру строки row;
* int getRowCount() – получить количество строк в field;

Закрытые параметры:

* QString m\_className – содержит значение className;
* QVector< QVariant > m\_data – вектор, содержит значение value;
  + 1. Класс Variable

Класс Variable реализует тег variable.

Публичные методы:

* void setExpression(const QString & text) – устанавливает значение <variableExpression>. text значение <variableExpression>;
* const QString getExpression() const – возвращает значение <variableExpression>;
* void setClassName( const QString & name ) - устанавливает атрибут claaName. name название класса;
* const QString getClassName() const – возвращает атрибут className;
* void setResetType( const QString & resetType) – устанавливает когда производить сброс переменной. resetType – значение сброса переменной;
* const QString getResetType() – возвращает атрибут resetType;
* void setResetGroup(const QString & resetGroup) – если resetType=Group, то на какой группе производит сброс. resetGroup имя группы для сброса;
* const QString getResetGroup() – возвращает атрибуте resetGroup;
* void setIncrementType( const QString & incrementType) – устанавливает когда происходит обновление переменной. incrementType значения обновления переменной;
* const QString getIncrementType() const – возвращает атрибут incrementType;
* void setIncrementGroup(const QString & incrementGroup) – если incrementType=Group, то на какой группе. incrementGroup имя группы;
* const QString getIncrementGroup() const - возвращает атрибут incrementGroup;
* void setCalculation( const QVariant & calculation) – устанавливает функцию вычисления переменной. calculation – название функции;

Закрытые параметры:

* QString m\_className – имя класса переменной;
* QString m\_resetType – тип сброса переменной;
* QString m\_resetGroup – имя группы для сброса переменной;
* QString m\_incrementType – тип для обновления переменной;
* QString m\_incrementGroup – имя группы для обновления переменной;
* QString m\_expression – значение variableExpression переменной;
* QString m\_calcularion – имя функции для вычисления переменной;
  + 1. Класс QueryString

Класс реализует класс queryString.

Публичные параметры:

* void setText( const QString & text ) – устанавливает текст queryString, text – строка;
* const QString getText() const – возвращает значение queryString;

Закрытые параметры:

* QString m\_text – текст queryString;
  + 1. Класс Title

Класс Title реализует тег title. Создается обьект класса, его обрабатывает класс Section;

* + 1. Класс Group

Класс Group реализует тег group.

Публичные методы:

* void setExpression( const QString & text ) – устанавливает groupExpression. text – значение groupExpression;
* const QString getExpression() const – возвращает значение groupExpression;
* void setHeader( const SectionPtr & header ) – устанавливает groupHeader. header указатель на groupHeader;
* const SectionPtr getHeader() const – возвращает указатель на groupHeader;
* void setFooter( const SectionPtr & footer ) – устанавливает groupFooter. footer указатель на groupFooter;
* const SectionPtr getFooter() const – возвращает указатель на groupFooter.

Закрытые параметры:

* QString m\_expression – значение groupExpression;
* SectionPtr m\_header – указатель на groupHeader;
* SectionPtr m\_footer – указатель на groupFooter.
  + 1. Класс Detail

Класс Deatail реализует тег detail. Создается обьект класса, его обрабатывает класс Section;

* + 1. Класс Section

Класс Section вспомогательный класс для класса Band.

Публичные методы:

* void setWidth( int width ) – устанавливает ширину width секции;
* int getWidth() const – возвращает значение ширины секции;
* int getHeight() const – получить высоту секции;
* void addBand( const BandPtr & band ) – добавить секцию band. band – указатель на секцию band;
* const BandPtr getBand( int index ) const – получить указатель на секцию band по индексу index;
* int getBandsSize() const – получить размер секции band;
* const QVector< BandPtr > getBands() const – возвращает вектор с указателями на секции band.

Защищённые параметры:

* QVector< BandPtr > m\_bands – вектор указателей на секции band;
* int m\_width – ширина секции;
* int m\_height – высота секции.
  + 1. Класс Rectangle

Класс Rectangle реализует тег rectangle. Создается обьект класса, его обрабатывает класс Band;

* + 1. Класс Line

Класс Line реализует тег line. Создается обьект класса, его обрабатывает класс Band;

* + 1. Класс Ellipse

Класс Ellipse реализует тег ellipse. Создается обьект класса, его обрабатывает класс Band;

* + 1. Класс TextWidget

Класс TextWidget отображает название тега при отображении на макете отчёта.

Публичные методы:

* void setText( const QString & text ) – устанавливает текст тега - text. Параметры не будут заменены;
* const QString getText() const – возвращает текст тега;
* void setOriginalText( const QString & text ) – устанавливает оригинальный текст тега - text. Все параметры будут заменены на реальные значения;
* const QString getOriginalText() const – возвращает оригинальный текст тега;

Защищенные параметры:

* QString m\_text – текс тега;
* QString m\_originalText – оригинальный текс тега;
  + 1. Класс Widget

Класс Widget вспомогательный класс для отображения тегов в виде виджетов на макете отчета.

Публичные методы:

* void setPosition( const QPoint & pos ) - устанавливает положение объекта. Аргумент pos - неотрицательные координаты объекта, с центром координат в верхнем левом углу. Координаты задаются относительно внешнего объекта. pos положение объекта;
* void setX( int x ) – устанавливается значение координаты х;
* void setY( int y ) – устанавливается значение координаты у;
* void setSize( const QSize & size ) – устанавливается размер size объекта;
* void setWidth( int width ) – устанавливается ширина width объекта;
* void setHeight( int height ) – устанавливается ширина height объекта;
* void setRect( const QRect & rect ) – устанавливается высота rect обьекта;
* void setStyle( const QString & style ) – устанавливается стиль syle обьекта;
* void setAlignment( Qt::Alignment alignment ) – устанавливается выравнивание обьета. alignment – значение выравнивания;
* Qt::Alignment getAlignment() – получить значение выравнивания;
* const QPoint getPos() const – устанавливается позиция объекта. Возвращается позиция объекта;
* int getX() const – устанавливает значение координаты x. Возвращает значение координаты х;
* int getY() const – устанавливает значение координаты у. Возвращает значение координаты у;
* const QSize getSize() const – устанавливает и возвращает размер объекта;
* int getWidth() const – возвращает текущую ширину объекта;
* const QRect getRect() const – возвращает координаты и размер объекта;
* const QString getStyle() const – получает и возвращает индекс стиля.

Защищенные параметры:

* QRect m\_rect – хранит размеры объекта;
* Qt::Alignment m\_alignment – значение выравнивания;
* QString m\_style – значение стиля.
  + 1. Класс Band

Класс Band реализует тег band.

Публичные методы:

* void addStaticText( const StaticTextPtr & staticText ) – добавляет <staticText> в <band>. staticText - указатель на <staticText>;
* const QVector< StaticTextPtr > getStaticTexts() const – возвращает вектор указателей на <staticText>;
* const StaticTextPtr getStaticText( int index ) const – возвращает указатель на <staticText> по индексу index;
* int getStaticTextsSize() const – возвращает количество <staticText> в текущем <band>.
* void addTextField( const TextFieldPtr & textField ) – добавляет <textField> в <band>. textField указатель на <textField>;
* const QVector< TextFieldPtr > getTextFields() const - возвращает вектор указателей на <textField>;
* const TextFieldPtr getTextField( int index ) const – возвращает указатель на <textField> по индексу index;
* int getTextFieldsSize() const – возвращает количество <textField> в текущем <band>;
* int getTextWidgetsSize() const - возвращает количество текстовых виджетов( <staticText>, <textField> );
* QVector< TextWidgetPtr > getTextWidgets() const - возвращает вектор текстовых виджетов( <staticText>, <textField> );
* void addLine( const LinePtr & line ) – добавляет <line> в <band>. line указатель на <line>
* const LinePtr getLine( int index ) const – возвращает указатель на <line> по индексу index;
* int getLinesSize() const - возвращает количество <line> в текущем <band>;
* const QVector< LinePtr > getLines() const - возвращает вектор указателей на <line>;
* void addRect( const RectPtr & rect ) - Добавляем <rectangle> в <band>. rect указатель на <rectangle>
* const RectPtr getRect( int index ) const - возвращает указатель на <rect> по индексу index;
* int getRectsSize() const - возвращает количество <rectangle> в текущем <band>;
* const QVector< RectPtr > getRects() const – возвращает вектор указателей на <rectangle>;
* void addEllipse( const EllipsePtr & ellipse ) – добавляет <ellipse> в <band>. ellipse указатель на <ellipse>;
* const EllipsePtr getEllipse( int index ) const - возвращает указатель на <ellipse> по индексу index;
* int getEllipsesSize() const - возвращает количество <ellipse> в текущем <band>;
* const QVector< EllipsePtr > getEllipses() const - возвращает вектор указателей на <ellipse>;
* void addImage( const ImagePtr & image ) - добавляет <image> в <band>. image указатель на <image>;
* const ImagePtr getImage( int index ) const - Возвращает указатель на <image> по индексу index;
* int getImagesSize() const - возвращает количество <image> в текущем <band>;
* const QVector< ImagePtr > getImages() const - возвращает вектор указателей на <image>.

Закрытые параметры:

* QVector< StaticTextPtr > \_staticTexts – вектор StaticText;
* QVector< TextFieldPtr > m\_textFields – вектор TextField;
* QVector< TextWidgetPtr > m\_textWidgets – вектор TextWidge;
* QVector< LinePtr > m\_lines – вектор Line;
* QVector< RectPtr > m\_rects – вектор Rect;
* QVector< EllipsePtr > m\_ellipses – вектор Ellipse;
* QVector< ImagePtr > m\_images – вектор Image.
  + 1. Класс Image

Класс Image реализует тег image.

Публичные методы:

* void setImage( const QImage & image ) – устанавливает изображение image;
* const QImage getImage() – возвращает изображение;

Закрытые параметры:

* QImage m\_image – имя изображения;
  + 1. Класс StaticText

Класс StaticText реализует тег staticText.

* + 1. Класс TextField

Класс TextField реализует тег textField.

Публичные методы:

* void setClassName( const QString & name ) – устанавливает имя класса - name. Данное значение будет использоваться при вычислении содержимого;
* const QString getClassName() const – возвращает текущее имя класса.

Защищенные параметры:

* QString m\_className – имя параметра.

1. Технология эксплуатации
   1. Руководство для администратора

Для сборки и установки проекта необходимо скачать архив с исходными файлами с сайта проекта по адресу [20]. Для успешной сборки проекта необходимы библиотеки Qt версии не ниже пятой. И GCC [21] версии не ниже 4.8 для Linux. Для Windows необходим установленный CMake [22, 23] версии не ниже 3.5 и MinGW msvc2015\_64 [24, 25].

После скачивания, разархивирования и при выполненных требованиях, можно приступать к сборке с помощью утилиты CMake. Для этого необходимо перейдите в корень разархивированного проекта. Во избежания ошибок при сборке желательно избегать пробелов и русских бук в пути к исходным тестам и папке сборки. Находясь в корне вызовем командную строку и выполним команду:

- Windows: cmake -G "MinGW Makefiles" -B./bin -H./

- Unix: cmake -G "Unix Makefiles" -B./bin -H./

В случаи успеха можно компилировать проект, для этого в командной строке выполнить команду:

- Windows: mingw32-make -C./bin

- Linux: make -C ./bin

При отсутствии ошибок при сборке, в папке qtreportlib появятся файлы библиотеки, после чего библиотеку можно будет использовать при разработке программных продуктов.

При сборке библиотек на Windows возможны следующие ошибки:

- При появлении сообщения: "cmake не является внутренней или внешней командой, исполняемой программой или исполняемым файлом", проверить правильность установки CMake, при необходимости внести пути к папке bin утилиты CMake в системную переменную PATH [26].

- При появлении сообщения: "CMake Error: CMake was unable to find a build program corresponding to "MinGW Makefiles". CMAKE\_MAKE\_PROGRAM s not set. "

Проверьте правильность указания пути к mingw-make в PATH.

Пример пути: C:\Qt\Qt5.5.1\Tools\mingw492\_32\bin

- При появлении сообщения: "By not providing "FindQt5Widgets.cmake" in CMAKE\_MODULE\_PATH this project has asked CMake to find a package configuration file provided by "Qt5Widgets", but CMake did not find one."

Необходимо добавить в команду блок [CMAKE\_PREFIX\_PATH="путь к qt"].

Пример команды:

cmake -DCMAKE\_PREFIX\_PATH="C:\Qt\Qt5.5.1\5.5\mingw492\_32" -G "MinGW Makefiles" -B./bin -H./

При установке на Linux проверьте правильность установки Qt5 пройдете в /usr/lib/x86\_64-linux-gnu/qt5/bin . При отсутствии в папке qt5 папки bin, то откройте терминал и ведите sudo apt-get install qt5-default.

Ошибки на Linux:

- При появлении сообщения: "By not providing "FindQt5Widgets.cmake" in CMAKE\_MODULE\_PATH this project has asked CMake to find a package configuration file provided by "Qt5Widgets", but CMake did not find one."

Необходимо добавить в команду блок [CMAKE\_PREFIX\_PATH="путь к qt"].

Пример:

cmake CMAKE\_PREFIX\_PATH="/usr/lib/x86\_64-linux-gnu/qt5" -G "Unix Makefiles" -B./bin -H./

* 1. Руководство для программиста

Для использования библиотеки скопируйте сгенерированные файлы и папку qtreportslib в папку с проектом. В файле проекта с расширением .pro убедитесь в наличии следующих строк (см. рисунок 3.1).

QT += sql printsupport widgets

CONFIG +=c++11

INCLUDEPATH +="путь до\qtreportslib"

LIBS += "путь до\libqtreports.dll"

Рисунок 3.1 – Содержание .pro файла

Далее необходимо подключить заголовочный файл: #include "engine.hpp".

Теперь необходимо создать обьект класса Engine (движка отчета): qtreports::Engine engine.

После чего необходимо указать путь и имя файла отчета: engine.open( "path/file.xml" ).

В случае использования в качестве источника данных базы данных, необходимо создать подключение, выбрать базу данных и открыть ее   
(см. рисунок 3.2).

QSqlDatabase db = QSqlDatabase::addDatabase( "QSQLITE" );

db.setDatabaseName( "DB" );

db.open();

Рисунок 3.2 – Использование базы данных

Теперь нужно передать движку отчета соединение с базой данных: engine.setConnection( db ).

После чего экспортировать отчет в необходимый формат (см. рисунок 3.3).

engine.createPDF("test.pdf");

engine.createHTML("test.html");

Рисунок 3.3 – Экспорт

В конце необходимо закрыть соединение с базой данных: db.close().

В случаях использования абстрактных типов данных необходимо обьявить и заполнить карту: QMap<QString, QVector < QVariant > map.

И передать генератору отчета данные: engine.setDataSource( map ).

После чего мы может экспортировать в нужный формат.

Если результат не совпадает с запланированным или программа работает не корректно, для отладки можно воспользоваться функцией engine.getLastError().

1. примеры использования

В примерах использовалась база данных с таблицами:

- group1 (name varchar (80), gr varchar (20), year integer);

- images (idImg integer, nameImg varchar (80), image blob).

* 1. Текст и данные из базы данных

На рисунке 4.1 приведен результат работы библиотеки с использованием базы данных и тегов: querySrting, field, staticText, textField, reportElement, text, textFieldExpression, style.



Рисунок 4.1 – Текст и данные из базы данных

На рисунке 4.2 приведена структура XML документа, использованные теги и параметры.



Рисунок 4.2 – Структура XML документа

* 1. Заголовок и изображения

На рисунке 4.3 приведен результат работы библиотеки с использованием тегов: querySrting, field, staticText, textField, reportElement, text, textFieldExpression, style, title, line, rectangle (rect), ellipse, image, imageExpression.

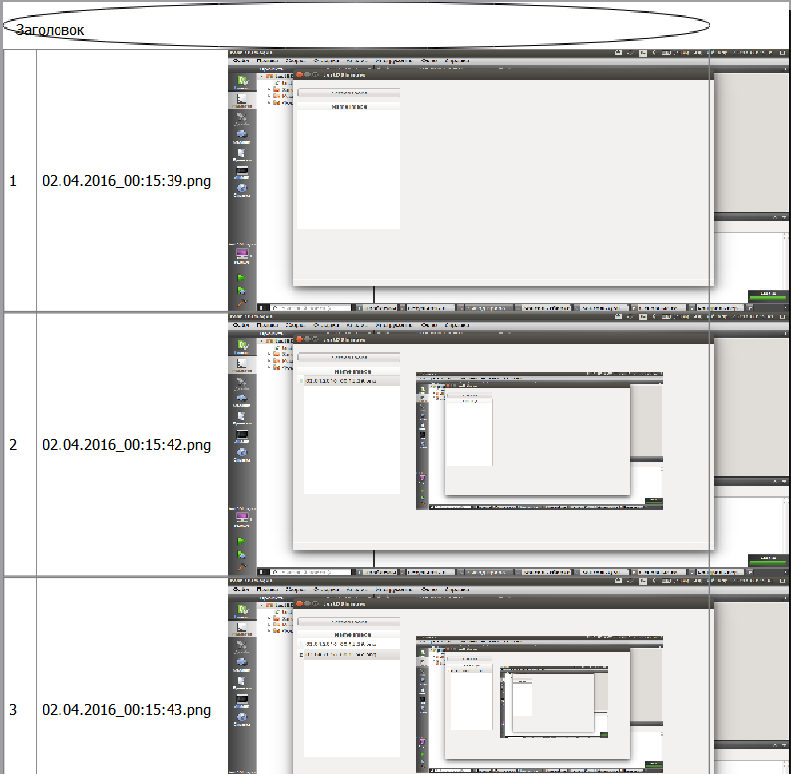


Рисунок 4.3 – Заголовок и изображения

На рисунке 4.4 приведена структура XML документа, использованные теги и параметры.



Рисунок 4.4 – XML документ

* 1. Группировка данных

На рисунке 4.5 приведен результат работы библиотеки с использованием тегов: querySrting, field, staticText, textField, reportElement, text, textFieldExpression, style, group, groupExpression, groupHeader, groupFooter.

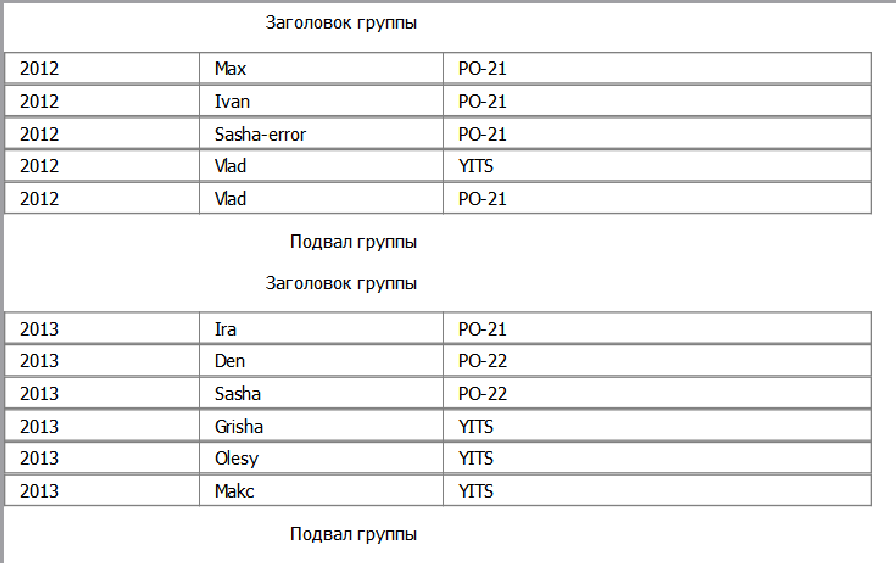


Рисунок 4.5 – Группировка данных

На рисунке 4.6 приведен фрагмент XML документа, с использованными тегами и параметрами.



Рисунок 4.6 – Фрагмент XML документа

* 1. Форматирование текста

На рисунке 4.7 приведен результат работы библиотеки с использованием тегов: querySrting, field, staticText, textField, reportElement, text, textFieldExpression, style, title, line, rectangle (rect), textElement.

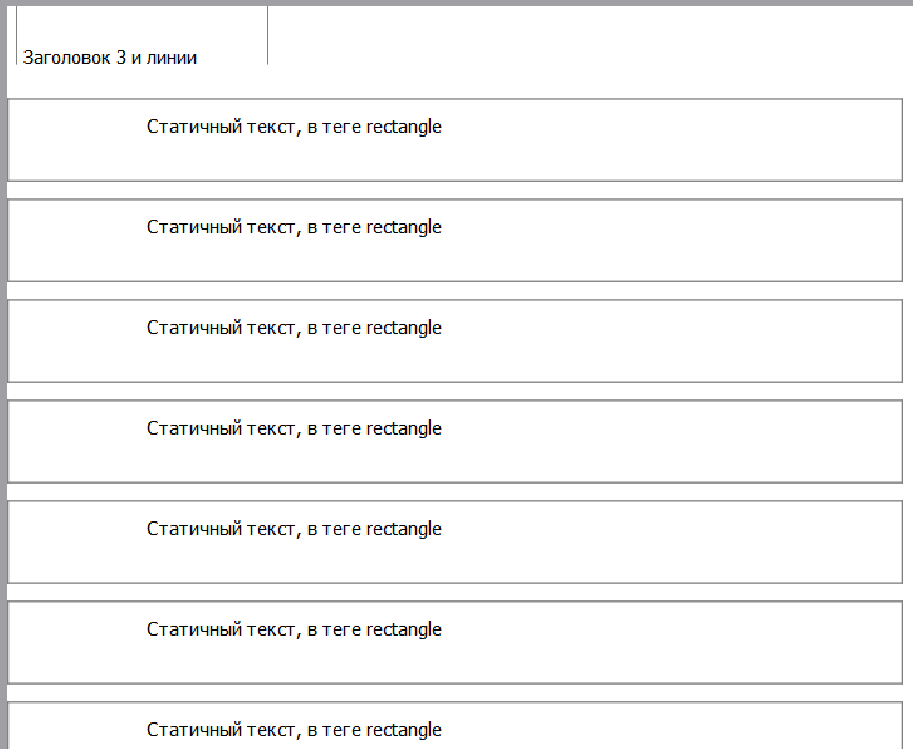


Рисунок 4.7 – Форматирование текста

На рисунке 4.8 представлен фрагмент XML документа, с использованными тегами и параметрами.



Рисунок 4.8 – Фрагмент XML документа

Заключение

В данной выпускной квалификационной работе был создан генератор отчетов с использованием библиотек Qt5.

В первой части работы были рассмотрены теги. Их иерархия, параметры приведены описания.

Во второй части работы рассмотрена структура генератора отчетов на программном уровне. Приведено взаимодействие классов, их описание и перечисление параметров.

В третье части ВКР предоставлена технология эксплуатации для администратора и программиста. Приведены описание возможных ошибок при сборке и способах их решения.

В четвертой части представлены примеры использования генератора отчетов.

Разработанная программа могла бы стать полноценным инструментом генерации отчетов. Функциональная доработка, оптимизация и исправления программного кода увеличили бы производительность, а также спектр возможностей данного программного продукта.

Список использованных источников

1. Сайт проекта eXaro [Электронный ресурс] : sourceforge.net - Режим доступа : <http://exaro.sourceforge.net/> (дата обращения 15.03.2016).
2. Ресурс с исходными файлами проекта eXaro [Электронный ресурс] : sourceforge.net - Режим доступа : <https://sourceforge.net/projects/exaro/> (дата обращения 15.03.2016).
3. Сайт проекта CuteReport [Электронный ресурс] : <cute-report.com> – Режим доступа : <https://cute-report.com/ru> (дата обращения 15.03.2016).
4. Ресурс с исходными файлами проекта CuteReport [Электронный ресурс] : sourceforge.net - Режим доступа <https://sourceforge.net/projects/qreport/> (дата обращения 15.03.2016).
5. Сайт проекта LimeReport [Электронный ресурс] : limereport.ru -Режим доступа : <http://limereport.ru/ru/index.php> (дата обращения 15.03.2016).
6. Ресурс с исходными файлами проекта LimeReport [Электронный ресурс] : sourceforge.net - Режим доступа <https://sourceforge.net/projects/limereport/> (дата обращения 15.03.2016).
7. Сайт проекта Qt [Электронный ресурс] : qt.io – Режим доступа : <https://www.qt.io/ru/download-open-source/#section-2> (дата обращения 13.03.2016).
8. Jaspersoft The JasperReports Ultimate Guide, Third Edition : Jaspersoft Corporation, 2011. – 321 p.
9. Структура XML документа [Электронный ресурс] : Сайт Кунегина С.В. – Режим доступа : <http://kunegin.narod.ru/ref2/xml/go21.htm> (дата обращения 14.04.2016).
10. Язык XML [Электронный ресурс] : Сайт со статьями посвященными программированию - Режим доступа : <http://www.codenet.ru/webmast/xml/part2.php> (дата обращения 15.04.2016).
11. XML [Электронный ресурс] : Википедия – свободная энциклопедия – Режим доступа : <https://ru.wikipedia.org/wiki/XML> (дата обращения 10.04.2016).
12. Справочная документация по Qt [Электронный ресурс] : qt.io – Режим доступа : <http://doc.qt.io/qt-5/classes.html> (дата обращения 14.03.2016).
13. Объектно-ориентированное программирование [Электронный ресурс] : Документация Microsoft – Режим доступа : <https://msdn.microsoft.com/ru-ru/library/dd460654.aspx> (дата обращения 11.03.2016).
14. Свойства ООП [Электронный ресурс] : Сайт Андрей Вольберг – Режим доступа : <http://avolberg.ru/theory/oop/encapsulation> (дата обращения 05.03.2016).
15. PDF – глазами программиста [Электронный ресурс] : Сайт с публикацией статей – Режим доступа : <https://habrahabr.ru/company/abbyy/blog/108459/> (дата обращения 14.05.2016).
16. Сайт PDF библиотеки [Электронный ресурс] : pdflib.com – Режим доступа : <http://www.pdflib.com/> (дата обращения 25.04.2016).
17. Сайт PDF библиотеки [Электронный ресурс] : sourceforge.net – Режим доступа : <http://libharu.sourceforge.net/> (дата обращения 28.04.2016).
18. Справочник по HTML [Электронный ресурс] : htmlbook.ru – Режим доступа : <http://htmlbook.ru/> (дата обращения 03.04.2016).
19. Справочник по HTML [Электронный ресурс] : html.manual.ru. – Режим доступа : <http://html.manual.ru/> (дата обращения 09.04.2016).
20. Репозиторий с проектом [Электронный ресурс] : github.com –Режим доступа : <https://github.com/PO-21/QtReports> (дата обращения 09.02.2016).
21. Сайт проекта GCC [Электронный ресурс] : gccюgnu.org – Режим доступа : <https://gcc.gnu.org/> (дата обращения 01.03.2016).
22. Сайт проекта CMake [Электронный ресурс] : cmake.org – Режим доступа : <https://cmake.org/> (дата обращения 23.03.2016).
23. Введение с CMake [Электронный ресурс] : Сайт с публикацией статей – Режим доступа : <https://habrahabr.ru/post/155467/> (дата обращения 27.03.2016).
24. Сайт проекта MinGW [Электронный ресурс] : mingw.org – Режим доступа : <http://www.mingw.org/> (дата обращения 29.03.2016).
25. Сайт проекта MinGW-w64 [Электронный ресурс] : mingw-w64.org – Режим доступа : <http://mingw-w64.org/doku.php> (дата обращения 29.03.2016).
26. Добавления местоположения программ в переменную среды PATH [Электронный ресурс] : microsoft.com – Режим доступа: <https://msdn.microsoft.com/ru-ru/library/office/ee537574(v=office.14).aspx> (дата обращения 30.03.2016).

Приложения А

(необязательное)

Содержимое файла parsetfromxml.cs

#include <QFile>

#include <QMessageBox>

#include <QDebug>

#include "parserfromxml.hpp"

namespace qtreports

{

namespace detail

{

ParserFromXML::ParseFunc bindParseFunc( ParserFromXML \* obj, ParserFromXML::ParseMethodPtr method )

{

using namespace std::placeholders;

auto func = std::bind( method, obj, \_1, \_2 );

return func;

}

template < typename T1 >

ParserFromXML::ParseFunc toParseFunc( ParserFromXML \* obj, bool( ParserFromXML::\*method )( QXmlStreamReader &, const T1 & ) )

{

auto parseMethodPtr = reinterpret\_cast< ParserFromXML::ParseMethodPtr >( method );

return bindParseFunc( obj, parseMethodPtr );

}

bool toBool( const QString & string )

{

return isEquals( string, "true" ) || isEquals( string, "1" );

}

ParserFromXML::ParserFromXML() : m\_log( new QString() )

{

m\_functions[ "report" ] = toParseFunc( this, &ParserFromXML::parseReport );

m\_functions[ "style" ] = toParseFunc( this, &ParserFromXML::parseStyle );

m\_functions[ "queryString" ] = toParseFunc( this, &ParserFromXML::parseQueryString );

m\_functions[ "field" ] = toParseFunc( this, &ParserFromXML::parseField );

m\_functions[ "variable" ] = toParseFunc(this, &ParserFromXML::parseVariable);

m\_functions[ "variableExpression" ] = toParseFunc(this, &ParserFromXML::parseVariableExpression);

m\_functions[ "group" ] = toParseFunc( this, &ParserFromXML::parseGroup );

m\_functions[ "groupExpression" ] = toParseFunc( this, &ParserFromXML::parseGroupExpression );

m\_functions[ "groupHeader" ] = toParseFunc( this, &ParserFromXML::parseGroupHeader );

m\_functions[ "groupFooter" ] = toParseFunc( this, &ParserFromXML::parseGroupFooter );

m\_functions[ "title" ] = toParseFunc( this, &ParserFromXML::parseTitle );

m\_functions[ "detail" ] = toParseFunc( this, &ParserFromXML::parseDetail );

m\_functions[ "band" ] = toParseFunc( this, &ParserFromXML::parseBand );

Продолжение приложения А

m\_functions[ "staticText" ] = toParseFunc( this, &ParserFromXML::parseStaticText );

m\_functions[ "textField" ] = toParseFunc( this, &ParserFromXML::parseTextField );

m\_functions[ "line" ] = toParseFunc( this, &ParserFromXML::parseLine );

m\_functions[ "rect" ] = toParseFunc( this, &ParserFromXML::parseRect );

m\_functions[ "rectangle" ] = toParseFunc(this, &ParserFromXML::parseRect);

m\_functions[ "ellipse" ] = toParseFunc( this, &ParserFromXML::parseEllipse );

m\_functions[ "image" ] = toParseFunc( this, &ParserFromXML::parseImage );

m\_functions[ "imageExpression" ] = toParseFunc( this, &ParserFromXML::parseImageExpression );

m\_functions[ "reportElement" ] = toParseFunc( this, &ParserFromXML::parseReportElement );

m\_functions[ "textElement" ] = toParseFunc( this, &ParserFromXML::parseTextElement );

m\_functions[ "font" ] = toParseFunc( this, &ParserFromXML::parseFont );

m\_functions[ "text" ] = toParseFunc( this, &ParserFromXML::parseText );

m\_functions[ "textFieldExpression" ] = toParseFunc( this, &ParserFromXML::parseTextFieldExpression );

}

ParserFromXML::~ParserFromXML() {}

bool ParserFromXML::parse( const QString & path )

{

m\_report.clear();

m\_lastError = "";

m\_log.setString( new QString() );

if( !QFile::exists( path ) )

{

m\_lastError = "The file not exists";

return false;

}

QFile file( path );

file.open( QIODevice::OpenModeFlag::ReadOnly | QIODevice::Text );

if( !file.isOpen() )

{

m\_lastError = "The file can not be opened";

return false;

}

return parseDocument( file.readAll() );

}

bool ParserFromXML::getValue( QXmlStreamReader & reader, QString & data )

{

m\_log << "getValue():\tstart" << endl;

while( !reader.atEnd() && !reader.isEndElement() )

{

data += reader.text().toString();

reader.readNext();

}

if( reader.hasError() )

{

m\_log << "getValue():\terror" << endl;

m\_lastError = reader.errorString();

return false;

}

m\_log << "getValue():\tend. data: " << data << endl;

return true;

}

Продолжение приложения А

bool ParserFromXML::getAttribute( QXmlStreamReader & reader, const QString & name, QString & data, AttributeOption option )

{

m\_log << "getAttribute():\tstart. name: " << name << endl;

auto && attributes = reader.attributes();

if( !attributes.hasAttribute( name ) )

{

m\_log << "getAttribute():\tnot have attribute: " + name << endl;

if( option == AttributeOption::Optional )

{

return true;

}

m\_log << "getAttribute():\terror" << endl;

auto elementName = reader.name().toString();

m\_lastError = "Element \"" + reader.name().toString() +

"\" not have attribute: " + name;

return false;

}

data = attributes.value( name ).toString();

m\_log << "getAttribute():\tend. name: " << name << ",\t data: " << data << endl;

return true;

}

bool ParserFromXML::getRequiredAttribute( QXmlStreamReader & reader, const QString & name, QString & data )

{

return getAttribute( reader, name, data, AttributeOption::Required );

}

bool ParserFromXML::getOptionalAttribute( QXmlStreamReader & reader, const QString & name, QString & data )

{

return getAttribute( reader, name, data, AttributeOption::Optional );

}

bool ParserFromXML::goToElementEnd( QXmlStreamReader & reader )

{

m\_log << "goToEnd():\tstart" << endl;

int level = 0;

while( !reader.atEnd() )

{

reader.readNext();

if( reader.isEndElement() )

{

if( level <= 0 )

{

break;

}

--level;

}

if( reader.isStartElement() )

{

++level;

}

}

if( reader.hasError() )

Продолжение приложения А

{

m\_log << "goToEnd():\terror" << endl;

m\_lastError = reader.errorString();

return false;

}

m\_log << "goToEnd():\tend" << endl;

return true;

}

bool ParserFromXML::parseChilds( QXmlStreamReader & reader, const ObjectPtr & object )

{

m\_log << "parseChilds():\tstart" << endl;

while( !reader.atEnd() )

{

reader.readNext();

if( reader.isEndElement() )

{

break;

}

if( !reader.isStartElement() )

{

continue;

}

auto name = reader.name().toString();

m\_log << "parseChilds():\tcurrent tag: " << name << endl;

if( m\_functions.contains( name ) )

{

m\_log << "parseChilds():\tuse func for: " << name << endl;

auto func = m\_functions[ name ];

if( !func( reader, object ) )

{

return false;

}

}

else

{

m\_log << "parseChilds():\tgoToElementEnd: " << name << endl;

if( !goToElementEnd( reader ) )

{

return false;

}

}

}

if( reader.hasError() )

{

m\_log << "parseChilds():\terror" << endl;

m\_lastError = reader.errorString();

return false;

}

m\_log << "parseChilds():\tend" << endl;

return true;

}

bool ParserFromXML::parseDocument( const QString & text )

Продолжение приложения А

{

QXmlStreamReader reader( text );

m\_report = ReportPtr( new Report() );

if( !parseChilds( reader, m\_report ) )

{

return false;

}

return !reader.hasError();

}

bool ParserFromXML::parseReport( QXmlStreamReader & reader, const ReportPtr & report )

{

QString name;

if( !getRequiredAttribute( reader, "name", name ) )

{

return false;

}

QString leftMargin;

if( !getOptionalAttribute( reader, "leftMargin", leftMargin ) )

{

return false;

}

QString rightMargin;

if( !getOptionalAttribute( reader, "rightMargin", rightMargin ) )

{

return false;

}

QString topMargin;

if( !getOptionalAttribute( reader, "topMargin", topMargin ) )

{

return false;

}

QString bottomMargin;

if( !getOptionalAttribute( reader, "bottomMargin", bottomMargin ) )

{

return false;

}

QString orientationString;

if( !getOptionalAttribute( reader, "orientation", orientationString ) )

{

return false;

}

QString pageWidthString;

if( !getOptionalAttribute( reader, "pageWidth", pageWidthString ) )

{

return false;

}

QString pageHeightString;

if( !getOptionalAttribute( reader, "pageHeight", pageHeightString ) )

{

return false;

}

Продолжение приложения А

if( !parseChilds( reader, report ) )

{

return false;

}

report->setTagName( "report" );

report->setName( name );

if( !leftMargin.isEmpty() )

{

report->setLeftMargin( leftMargin.toInt() );

}

if( !topMargin.isEmpty() )

{

report->setTopMargin( topMargin.toInt() );

}

if( !rightMargin.isEmpty() )

{

report->setRightMargin( rightMargin.toInt() );

}

if( !bottomMargin.isEmpty() )

{

report->setBottomMargin( bottomMargin.toInt() );

}

if( !orientationString.isEmpty() )

{

auto orientation = isEquals( orientationString, "portrait" ) ?

QPrinter::Orientation::Portrait :

QPrinter::Orientation::Landscape;

report->setOrientation( orientation );

}

if( !pageWidthString.isEmpty() )

{

report->setWidth( pageWidthString.toInt() );

}

if( !pageHeightString.isEmpty() )

{

report->setHeight( pageHeightString.toInt() );

}

return !reader.hasError();

}

bool ParserFromXML::parseStyle( QXmlStreamReader & reader, const ReportPtr & report )

{

QString nameString;

if( !getRequiredAttribute( reader, "name", nameString ) )

{

return false;

}

Продолжение приложения А

QString isDefaultString;

if( !getOptionalAttribute( reader, "isDefault", isDefaultString ) )

{

return false;

}

QString fontNameString;

if( !getOptionalAttribute( reader, "fontName", fontNameString ) )

{

return false;

}

QString fontSizeString;

if( !getOptionalAttribute( reader, "fontSize", fontSizeString ) )

{

return false;

}

QString fontColorString;

if( !getOptionalAttribute( reader, "fontColor", fontColorString ) )

{

return false;

}

QString isBoldString;

if( !getOptionalAttribute( reader, "isBold", isBoldString ) )

{

return false;

}

QString isItalicString;

if( !getOptionalAttribute( reader, "isItalic", isItalicString ) )

{

return false;

}

QString isUnderlineString;

if( !getOptionalAttribute( reader, "isUnderline", isUnderlineString ) )

{

return false;

}

QString isStrikeThroughString;

if( !getOptionalAttribute( reader, "isStrikeThrough", isStrikeThroughString ) )

{

return false;

}

QString pdfFontNameString;

if( !getOptionalAttribute( reader, "pdfFontName", pdfFontNameString ) )

{

return false;

}

QString pdfEncodingString;

if( !getOptionalAttribute( reader, "pdfEncoding", pdfEncodingString ) )

{

return false;

}

Продолжение приложения А

QString isPdfEmbeddedString;

if( !getOptionalAttribute( reader, "isPdfEmbedded", isPdfEmbeddedString ) )

{

return false;

}

while( !reader.atEnd() && !reader.isEndElement() )

{

reader.readNext();

}

if( reader.hasError() )

{

m\_lastError = reader.errorString();

return false;

}

StylePtr style( new Style() );

style->setTagName( "style" );

style->setName( nameString );

if( !isDefaultString.isEmpty() )

{

bool isDefault = toBool( isDefaultString );

style->setAsDefault( isDefault );

if( isDefault )

{

report->setDefaultStyle( style );

}

}

if( !fontNameString.isEmpty() )

{

style->setFontName( fontNameString );

}

if( !fontColorString.isEmpty() )

{

style->setFontColor( QColor( fontColorString ) );

}

if( !fontSizeString.isEmpty() )

{

style->setFontSize( fontSizeString.toInt() );

}

if( !isBoldString.isEmpty() )

{

style->setBold( toBool( isBoldString ) );

}

if( !isItalicString.isEmpty() )

{

style->setItalic( toBool( isItalicString ) );

}

if( !isUnderlineString.isEmpty() )

{

style->setUnderline( toBool( isUnderlineString ) ); }

Продолжение приложения А

if( !isStrikeThroughString.isEmpty() )

{

style->setStrikeThrough( toBool( isStrikeThroughString ) );

}

if( !pdfFontNameString.isEmpty() )

{

style->setPDFFontName( pdfFontNameString );

}

if( !pdfEncodingString.isEmpty() )

{

style->setPDFEncoding( pdfEncodingString );

}

if( !isPdfEmbeddedString.isEmpty() )

{

style->setPDFEmbedded( toBool( isPdfEmbeddedString ) );

}

report->addStyle( nameString, style );

return !reader.hasError();

}

bool ParserFromXML::parseField( QXmlStreamReader & reader, const ReportPtr & report )

{

QString name;

if( !getRequiredAttribute( reader, "name", name ) )

{

return false;

}

QString className;

if( !getRequiredAttribute( reader, "class", className ) )

{

return false;

}

FieldPtr field( new Field() );

if( !parseChilds( reader, field ) )

{

return false;

}

field->setTagName( "field" );

field->setName( name );

field->setClassName( className );

report->setField( name, field );

return !reader.hasError();

}

bool ParserFromXML::parseVariable(QXmlStreamReader & reader, const ReportPtr & report)

{

QString nameString;

if (!getRequiredAttribute(reader, "name", nameString))

{

return false;

}

Продолжение приложения А

VariablePtr variable(new Variable());

variable->setTagName("variable");

variable->setName(nameString);

if (!parseChilds(reader, variable))

{

return false;

}

report->addVariable(nameString, variable);

return !reader.hasError();

}

bool ParserFromXML::parseVariableExpression(QXmlStreamReader & reader, const GroupPtr & variable)

{

QString text;

if (!getValue(reader, text))

{

return false;

}

variable->setExpression(text);

return !reader.hasError();

}

bool ParserFromXML::parseGroup( QXmlStreamReader & reader, const ReportPtr & report )

{

QString nameString;

if( !getRequiredAttribute( reader, "name", nameString ) )

{

return false;

}

GroupPtr group( new Group() );

group->setTagName( "group" );

group->setName( nameString );

if( !parseChilds( reader, group ) )

{

return false;

}

report->addGroup( nameString, group );

return !reader.hasError();

}

bool ParserFromXML::parseGroupExpression( QXmlStreamReader & reader, const GroupPtr & group )

{

QString text;

if( !getValue( reader, text ) )

{

return false;

}

group->setExpression( text );

return !reader.hasError();

}

Продолжение приложения А

bool ParserFromXML::parseGroupHeader( QXmlStreamReader & reader, const GroupPtr & group )

{

SectionPtr header( new Section() );

header->setTagName( "groupHeader" );

if( !parseChilds( reader, header ) )

{

return false;

}

group->setHeader( header );

return !reader.hasError();

}

bool ParserFromXML::parseGroupFooter( QXmlStreamReader & reader, const GroupPtr & group )

{

SectionPtr footer( new Section() );

footer->setTagName( "groupFooter" );

if( !parseChilds( reader, footer ) )

{

return false;

}

group->setFooter( footer );

return !reader.hasError();

}

bool ParserFromXML::parseTitle( QXmlStreamReader & reader, const ReportPtr & report )

{

TitlePtr title( new Title() );

if( !parseChilds( reader, title ) )

{

return false;

}

title->setTagName( "title" );

report->setTitle( title );

return !reader.hasError();

}

bool ParserFromXML::parseDetail( QXmlStreamReader & reader, const ReportPtr & report )

{

DetailPtr detail( new Detail() );

if( !parseChilds( reader, detail ) )

{

return false;

}

detail->setTagName( "detail" );

report->setDetail( detail );

return !reader.hasError();

}

Продолжение приложения А

bool ParserFromXML::parseBand( QXmlStreamReader & reader, const SectionPtr & section )

{

QString height;

if( !getRequiredAttribute( reader, "height", height ) )

{

return false;

}

BandPtr band( new Band() );

if( !parseChilds( reader, band ) )

{

return false;

}

band->setTagName( "band" );

band->setHeight( height.toInt() );

section->addBand( band );

return !reader.hasError();

}

bool ParserFromXML::parseStaticText( QXmlStreamReader & reader, const BandPtr & band )

{

StaticTextPtr staticText( new StaticText() );

if( !parseChilds( reader, staticText ) )

{

return false;

}

staticText->setTagName( "staticText" );

band->addStaticText( staticText );

return !reader.hasError();

}

bool ParserFromXML::parseTextField( QXmlStreamReader & reader, const BandPtr & band )

{

TextFieldPtr textField( new TextField() );

if( !parseChilds( reader, textField ) )

{

return false;

}

textField->setTagName( "textField" );

band->addTextField( textField );

return !reader.hasError();

}

bool ParserFromXML::parseLine( QXmlStreamReader & reader, const BandPtr & band )

{

LinePtr line( new Line() );

if( !parseChilds( reader, line ) )

{

return false;

}

line->setTagName( "line" );

band->addLine( line );

return !reader.hasError();

}

Продолжение приложения А

bool ParserFromXML::parseRect( QXmlStreamReader & reader, const BandPtr & band )

{

RectPtr rect( new Rect() );

if( !parseChilds( reader, rect ) )

{

return false;

}

rect->setTagName( "rect" );

band->addRect( rect );

return !reader.hasError();

}

bool ParserFromXML::parseEllipse( QXmlStreamReader & reader, const BandPtr & band )

{

EllipsePtr ellipse( new Ellipse() );

if( !parseChilds( reader, ellipse ) )

{

return false;

}

ellipse->setTagName( "ellipse" );

band->addEllipse( ellipse );

return !reader.hasError();

}

bool ParserFromXML::parseImage( QXmlStreamReader & reader, const BandPtr & band )

{

ImagePtr image( new Image() );

if( !parseChilds( reader, image ) )

{

return false;

}

image->setTagName( "image" );

band->addImage( image );

return !reader.hasError();

}

bool ParserFromXML::parseReportElement( QXmlStreamReader & reader, const WidgetPtr & widget )

{

QString xString;

if( !getRequiredAttribute( reader, "x", xString ) )

{

return false;

}

QString yString;

if( !getRequiredAttribute( reader, "y", yString ) )

{

return false;

}

QString widthString;

if( !getRequiredAttribute( reader, "width", widthString ) )

Продолжение приложения А

{

return false;

}

QString heightString;

if( !getRequiredAttribute( reader, "height", heightString ) )

{

return false;

}

QString styleString;

if( !getOptionalAttribute( reader, "style", styleString ) )

{

return false;

}

if( !goToElementEnd( reader ) )

{

return false;

}

if( !xString.isEmpty() )

{

widget->setX( xString.toInt() );

}

if( !yString.isEmpty() )

{

widget->setY( yString.toInt() );

}

if( !widthString.isEmpty() )

{

auto width = widthString.toInt();

widget->setWidth( width );

}

if( !heightString.isEmpty() )

{

widget->setHeight( heightString.toInt() );

}

if( !styleString.isEmpty() )

{

widget->setStyle( styleString );

}

return !reader.hasError();

}

bool ParserFromXML::parseTextElement( QXmlStreamReader & reader, const WidgetPtr & widget )

{

QString textAlignment;

if( !getRequiredAttribute( reader, "textAlignment", textAlignment ) )

{

return false;

}

Продолжение приложения А

QString textVAlignment;

if( !getOptionalAttribute( reader, "textVAlignment", textVAlignment ) )

{

return false;

}

if( !parseChilds( reader, widget ) )

{

return false;

}

auto isCenter = isEquals( textAlignment, "Center" );

auto isRight = isEquals( textAlignment, "Right" );

auto isVTop = isEquals( textVAlignment, "Top" );

auto isVBottom = isEquals( textVAlignment, "Bottom" );

auto hFlag = isCenter ? Qt::AlignCenter : isRight ? Qt::AlignRight : Qt::AlignLeft;

auto vFlag = isVTop ? Qt::AlignTop : isVBottom ? Qt::AlignBottom : Qt::AlignVCenter;

widget->setAlignment( hFlag | vFlag );

return !reader.hasError();

}

bool ParserFromXML::parseFont( QXmlStreamReader & reader, const WidgetPtr & widget )

{

QString isBold;

if( !getOptionalAttribute( reader, "isBold", isBold ) )

{

return false;

}

if( !goToElementEnd( reader ) )

{

return false;

}

if( !isBold.isEmpty() )

{

widget->setBold( toBool( isBold ) );

}

return !reader.hasError();

}

bool ParserFromXML::parseText( QXmlStreamReader & reader, const StaticTextPtr & staticText )

{

QString text;

if( !getValue( reader, text ) )

{

return false;

}

staticText->setOriginalText( text );

return !reader.hasError();

}

Продолжение приложения А

bool ParserFromXML::parseTextFieldExpression( QXmlStreamReader & reader, const TextFieldPtr & textField )

{

QString className;

if( !getRequiredAttribute( reader, "class", className ) )

{

return false;

}

QString text;

if( !getValue( reader, text ) )

{

return false;

}

textField->setOriginalText( text );

textField->setClassName( className );

return !reader.hasError();

}

bool ParserFromXML::parseImageExpression( QXmlStreamReader & reader, const ImagePtr & image )

{

QString text;

if( !getValue( reader, text ) )

{

return false;

}

image->setOriginalText( text );

return !reader.hasError();

}

bool ParserFromXML::parseQueryString( QXmlStreamReader & reader, const ReportPtr & report )

{

QString text;

if( !getValue( reader, text ) )

{

return false;

}

report->setQuery( text );

report->setTagName( "queryString" );

return !reader.hasError();

}

const ReportPtr ParserFromXML::getReport() const

{

return m\_report;

}

const QString ParserFromXML::getLastError() const

{

return m\_lastError;

}

const QString ParserFromXML::getLog() const

{

return \*m\_log.string();

}

}

}