QtReports

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

v0.3.0

HTTPS://GITHUB.COM/PO-31/QTREPORTS

Оглавление

Глава 1 Начало работы с QtReports

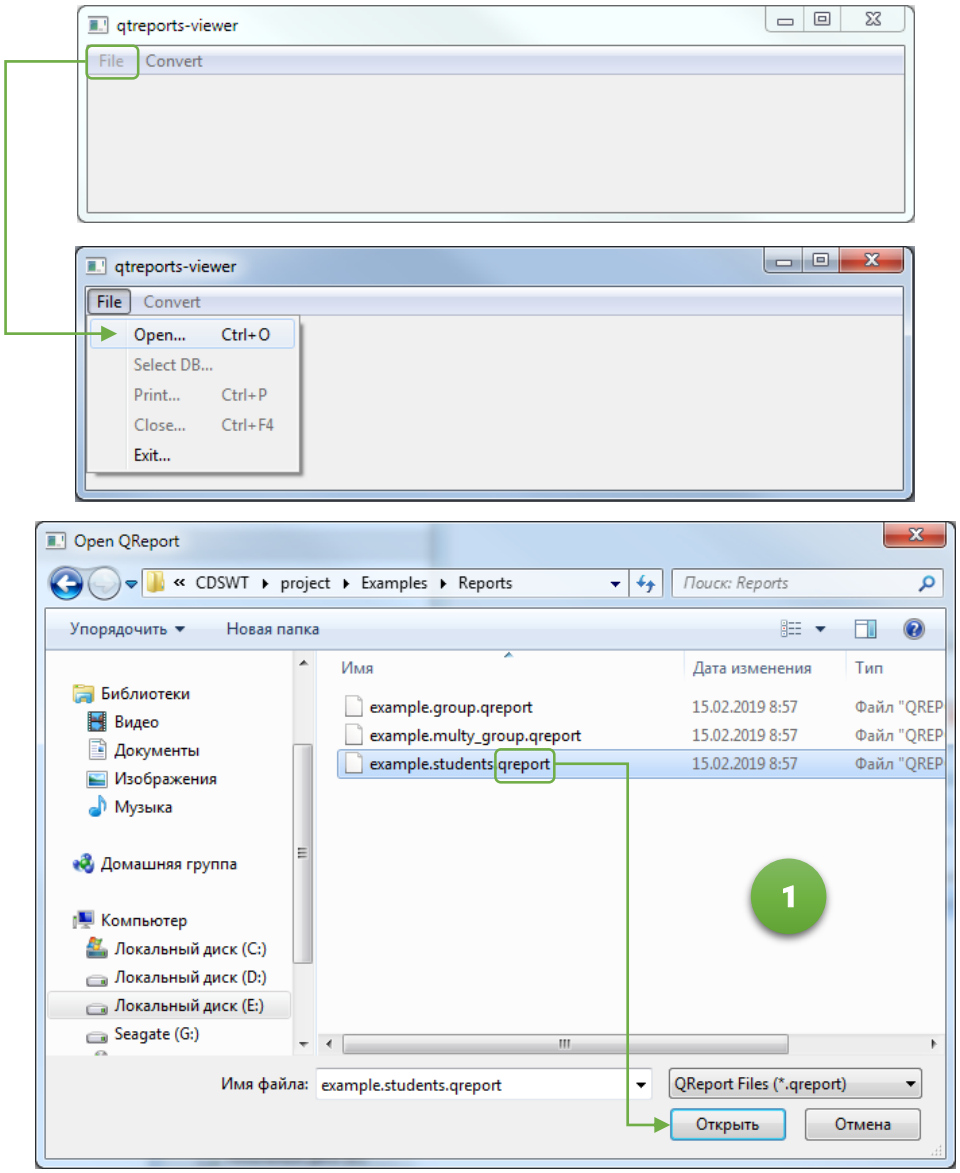
1.1 Введение в QtReports

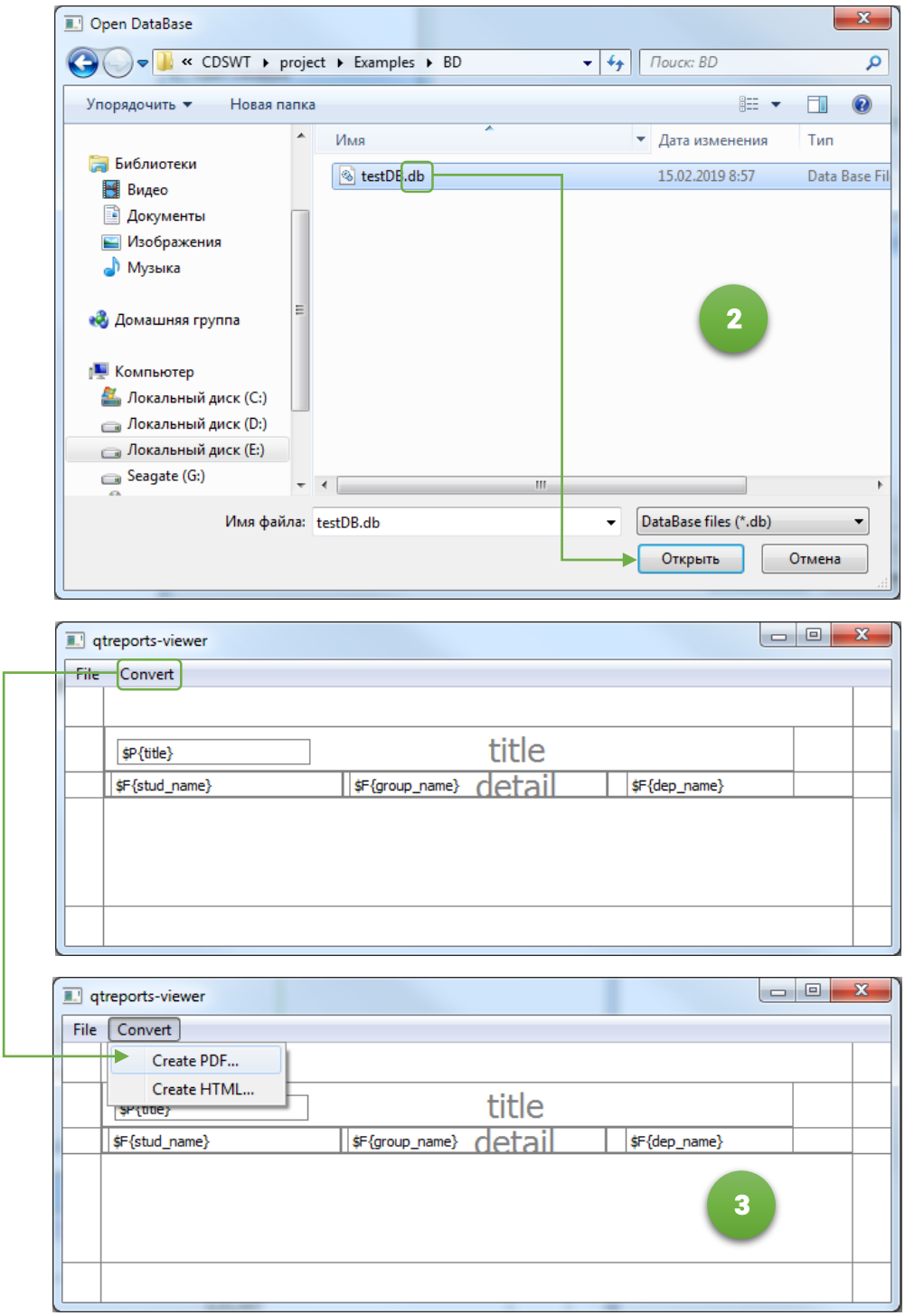
QtReports – это новый генератор отчетов для Qt-приложений, написанный при помощи фреймворка Qt. QtReports позволяет создавать сложные макеты, содержащие диаграммы, изображения, вложенные отчеты, кросс-таблицы и многое другое.

Доступ к данным происходит через стандартный класс Qt – QSqlDatabase, позволяющий работать посредством драйверов с различными СУБД; также доступ к данным может быть получен из XML-файлов. QtReports предоставляет возможность экспорта отчетов в форматы PDF и HTML.

1.2 Пользовательский интерфейс

В текущей версии QtReports (0.3.0) оконный интерфейс имеет только один модуль – QtReports Viewer, который имеет следующий вид:





1. После запуска средства просмотра шаблонов отчета QtReports Viewer необходимо выбрать файл с расширением «.qreport».

2. Далее нужно выбрать базу данных – файл с расширением «.db».

3. Теперь у Вас есть возможность экспорта данных из выбранной базы в соответствии с выбранным шаблоном в один из предоставленных форматов: PDF или HTML.

Дизайнер отчетов в данной версии еще не реализован.

1.3 Требования к аппаратному обеспечению

Для QtReports требуется 64-битный или 32-битный процессор и не менее 50мб свободного места на жестком диске. Рекомендуемый объем оперативной памяти – 2гб.

1.4 Требования к аппаратному обеспечению

Как и любой другой Qt-проект, для сборки QtReports требуется Qt-фреймворк. Для сборки QtReports Вам потребуется Qt Версии не ниже 5.8 с компилятором MinGW 32bit или MSVC 32/64bit.

Текущая версия QtReports поддерживает наиболее распространенные операционные системы:

- Windows 7 с разрядностью 32 или 64 бит
- Linux с разрядностью 32 или 64 бит

1.5 Доступ к исходным текстам

Последняя версия исходного кода QtReports доступна по адресу https://github.com/PO-31/QtReports. Вы можете скачать и скомпилировать этот исходный код, но он может содержать новые, неизданные функции и ошибки, поскольку находится на стадии разработки. Вся информация, необходимая для успешной сборки проекта находится в следующем пункте данного руководства.

1.6 Сборка и установка QtReports

Работа с QtReports начинается с загрузки исходных текстов с официального репозитория:

https://github.com/PO-31/QtReports

1.6.1 Сборка под Qt

Требования:

- Qt 5+
- gcc-5+/clang-3.4+/msvc14+

Шаг 1: Распаковка.

- Клонировать проект или скачать релиз из репозитория https://github.com/PO-31/QtReports/.
- Распаковать архив в папку QtReports. Рекомендуется, чтобы полный путь к папке QtReports содержал только латинские символы.

Шаг 2: Сборка.

Собрать проект можно несколькими способами, но если необходимо указать версию собираемой библиотеки, то необходимо задать переменную среды QTREPORTS_VERSION.

Способ 1. При помощи QtCreator:

- Используя программу, в корневой папке QtReports необходимо открыть файл проекта "QtReports.pro".
- Для сборки со статистикой для Coverage установить значение переменной среды BUILD_COVERAGE на "TRUE".
- Далее нажать кнопку "Собрать" (кнопка Молоток) для сборки всего проекта.
- Если необходимо собрать только определенный компонент, то нужно нажать ПКМ по проекту компонента в обозревателе проектов и выбрать "Собрать [Название_Модуля]".

Способ 2. При помощи Qmake:

- Запускаем консоль/терминал и переходим в каталог необходимого нам модуля
- Набираем qmake [-spec "Используемый_Компилятор"] "CONFIG += Требуемые_Типы_Сборки (release,debug,coverage,etc.)" Имя_Проекта_Модуля.pro
- После генерации Makefile вызываем make [-jКоличествоПотоков]

Шаг 3: Использование.

- После сборки файл библиотеки можно будет найти в папке "build/lib", а заголовочные файлы - "qtreports/include"
- Для подключения собранной библиотеки к Qmake проекту необходимо в файле проекта указать "LIB += -lqtreports" и, если необходимо, "LIB += Путь_До_Библиотеки", "INCLUDEPATH += Путь_До_Заголовочных_Файлов".

1.6.2 Сборка под MS Visual Studio

Требования:

- Microsoft Visial Studio 2015 (далее MVS 2015)
- Qt 5.8.0 с компилятором MVSC 2015 64-bit
- QtPackage или Qt Visual Studio tools для MVS 2015

Шаг 1: Установка Qt Package или Qt Visual Studio tools в MVS 2015.

- Запустить среду разработки MVS 2015 и проверить подключение к сети Интернет.
- Перейти «Сервис – Расширения и обновления…»
- В левой части окна выбрать вкладку «В сети», а в строке поиска (в правом верхнем углу) на брать «QtPackage» или «Qt Visual Studio tools».
- В списке найденных компонентов выбрать нужный и скачать, установка будет произведена автоматически.
- Перезапустить MVS 2015 для применения изменений.

Шаг 2: Настройка решения QtReports в MVS 2015.

- Перейти «Файл – Открыть решение или проект…», указать файл QtReports.sln.
- Перейти «QT5 – Qt Options». Добавить путь к компилятору MSVC 2015 64-bit для Qt.
- ПКМ по решению QtReports в окне «Обозреватель решений», выбрать «Change Solution's Qt Version» и указать компилятор MSVC 2015 64-bit. Для каждого проекта в окне Qt Project Settings переменная Version должна установиться MSVC 2015 64-bit, если нет, то применить вручную.
- Установить конфигурацию решения «Release», а платформу «x64».

Шаг 3: Сборка решения.

- Собирать проекты следует в последовательности: qtreports, qtreports-tests, qtreports-viewer.

1.7 Структура отчета в QtReports

Структура отчета определяется с помощью страницы, поделенной на различные горизонтальные секции, называемые «полосами» (тег band). Когда отчет связывается с данными, из которых генерируется выходной документ, эти секции (полосы) печатаются определенное количество раз, в соответствии с правилами, которые установлены автором отчета. Например, тег «page header» повторяется в начале каждой страницы, в то время как тег «detail» повторяется для каждой записи из результатов запроса.

1.7.1 Жизненный цикл отчета

Жизненный цикл отчета начинается с создания его дизайна. Разработка отчета означает создание своего рода шаблона, представляющего собой страницу с пустым пространством, которое может быть заполнено данными. Некоторые части страницы циклически повторяются, другие растягиваются по ширине страницы и так далее.

Шаблон сохраняется в виде подтипа XML-файла, называющегося QRXML («QR» - QtReports). Он содержит всю основную информацию о макете отчета, включая сложные формулы для выполнения вычислений, необязательный запрос для извлечения данных из источника данных и другие функции, описанные в следующих главах.

Жизненный цикл можно разделить на два этапа:

- Задачи, выполняемые на этапе разработки (разработка и планирование отчета, а также компиляция исходного файла .qreport, .qrxml).
- Задачи, которые должны быть выполнены во время выполнения (загрузка файла qreport, заполнение отчета и экспорт отчета в выбранный формат).

Основная роль QtReports в жизненном цикле отчета заключается в разработке этого отчета и создании связанного файла .qreport, хотя есть возможность предварительно просмотреть результат и экспортировать его во все поддерживаемые форматы. QtReports обеспечивает поддержку широкого спектра источников данных и позволяет пользователям тестировать свои собственные источники данных. Таким образом, QtReports является полной средой для разработки и тестирования отчетов.

При разработке отчета с использованием QtReports вы создаете файл .qrxml, который представляет собой документ XML, содержащий определение макета отчета. Перед запуском генерации отчета, .qrxml должен быть преобразован в файл .qreport. Файлы .qreport - это то, что вам нужно загружать в приложение для запуска отчета.

Выполнение отчета выполняется путем передачи файла .qreport и источника данных в QtReports. Существует несколько типов источников данных: Вы можете заполнить файл .qreport из запроса SQL или из XML файла.

Благодаря qreport-файлу и источнику данных можно сгенерировать окончательный документ в нужном Вам формате.

1.7.2 Тег Band

Тег Band – парный блок, отвечающий за общее описание элементов (разделов) отчета.

Необязательные атрибуты:

- height - высота блока (по умолчанию равно 0).
- isSplitAllow - устаревший, заменяется splitType. Это флаг, указывающий, разрешено ли блоку разрываться при расширении (значения: true, false).
- splitType - задает поведение разрыва блока (значения: stretch, prevent, immediate).

Пример использования:

```
<detail>
    <band height="400">
      <staticText>
          <reportElement x="380" y="0" width="200" height="20" />
          <text><![CDATA[Тест !!]]></text>
      </staticText>
      <textField>
          <reportElement x="220" y="20" width="100" height="200" />
          <textFieldExpression class="QString">
             <![CDATA[$P{title}]]>
          </textFieldExpression>
      </textField>
    </band>
</detail>
```

Дочерние элементы:

- <staticText>
- <textField>
- <graficElement>

1.7.3 Тег CDATA

CDATA - это часть содержания элемента, которая помечена для парсера как содержащая только символьные данные или только передающиеся параметры, а не разметку.

Начинается последовательностью символов <![CDATA[и заканчивается символами]]>

Пример использования:

```
 `<sender>Jonh Smith</sender>`
```

Открывающий и закрывающий теги "sender" будут интерпретированы как разметка. Однако, если записать ее так:

```
`<![CDATA[<sender>John Smith</sender>]]>`
```

то этот код будет интерпретирован так же, как если бы было записано:

```
&lt;sender&gt;John Smith&lt;/sender&gt;
```

Таким образом, теги «sender» будут восприниматься так же, как "John Smith", то есть как текст.

1.7.4 Тег Crosstab

Тег Crosstab – это особый тип элемента отчета, который объединяет данные в двумерную сетку.

Необязательные атрибуты:

- isRepeatColumnHeaders(true/false) – отображение заголовков колонок при переносе на следующую строку.
- isRepeatRowHeaders(true/false) – отображение заголовков строк при переносе на следующую строку.
- columnBreakOffset – указывает вертикальное пространство между разделами кросстаблицы, когда кросстаблица превышает ширину таблицы и два раздела отображаются на одной странице.
- runDirection – указывается направление заполнения кросстаблицы.
- ignoreWigth(true/false) – разрешение таблице бесконечно расширяться/или влево. Кросстаблицы будут отображаться поверх границы страницы. Используется в файлах, где возможно использование динамической ширины.

Пример использования:

```
<crosstab>
     <reportElement x="200" y="0" width="360" height="60"/>
     <rowGroup name="dep_name" width="130">
           <crosstabRowHeader>
                 <cellContents>
                       <textField>
                             <reportElement x="0" y="0" width="130" height="20"/>

     <textFieldExpression><![CDATA[$F{dep_name}]]></textFieldExpression>
                       </textField>
                 </cellContents>
           </crosstabRowHeader>
     </rowGroup>
     <columnGroup name="group_name" height="20">
           <crosstabColumnHeader>
                 <cellContents>
                       <textField>
                             <reportElement x="0" y="0" width="30" height="20"/>

     <textFieldExpression><![CDATA[$F{group_name}]]></textFieldExpression>
                       </textField>
                 </cellContents>
           </crosstabColumnHeader>
     </columnGroup>
     <crosstabCell width="30" height="20">
           <cellContents>
                 <textField>
                       <reportElement x="0" y="0" width="30" height="20"/>

     <textFieldExpression><![CDATA[$F{stud_name}]]></textFieldExpression>
                 </textField>
```

```
            </cellContents>
        </crosstabCell>
</crosstab>
```

Дочерние элементы:

- textField

1.7.5 Тег Detail

Тег detail – парный блок, который является «телом» отчета. В данном блоке содержится основная информация для каждой записи из источника данных. Может содержать несколько блоков <band>.

Примеры использования:

```
<detail>
    <band height="400">
        <staticText>
            <reportElement x="380" y="0" width="200" height="20" />
            <text><![CDATA[Тест !!]]></text>
        </staticText>
        <textField>
            <reportElement x="220" y="20" width="100" height="200" />
            <textFieldExpression class="QString">
                <![CDATA[$P{title}]]>
            </textFieldExpression>
        </textField>
    </band>
</detail>
```

Родительский элемент: <report>

Дочерние элементы: <band>

1.7.6 Тег Ellipse *(TBD)*

Тег Ellipse - Определение объекта эллипса.

1.7.7 Тег Field

Тег <Field> – представляет собой поле данных, которое будет хранить значения, извлеченные из источника данных отчета. Поле отчета представляет собой единственный способ отображения данных из источника в отчете шаблона, и использования этих данных в выражениях отчетов для получения желаемого результата.

При использовании запроса SQL в отчете, необходимо убедиться, что столбец для каждого поля получен после выполнения запроса. Соответствующий столбец должен нести то же самое имя и иметь тот же тип данных, что и поле, которое отображает его.

Обязательные атрибуты:

- name – название поля (тип - QString).
- class – класс значений поля (тип - QString).

- name – атрибут имени элемента является обязательным. Это позволяет ссылаться на поле в отчете по названию.
- class – второй атрибут; определяет имя класса для назначений полей. По умолчанию является QString, но может быть изменен на любой класс доступный во время выполнения.

Необязательные атрибуты:

- fieldDescription – это дополнительный текстовый фрагмент, может оказаться очень полезным при реализации пользовательских данных. Например, вы можете хранить в нем ключ или любую информацию, которая может понадобиться для того, чтобы восстановить значение поля из источника пользовательских данных во время выполнения.

Пример использования:

EmployeeID	LastName	FirstName	HireDate
int 4	varchar 50	varchar 50	datetime 8

Поля отчета следует указать следующим образом:

```
<field name="EmployeeID" class="Integer"/>
<field name="LastName" class="String"/>
<field name="FirstName" class="String"/>
<field name="HireDate" class="Data"/>
<field name="PersonName" class="String" isForPromting="true">
<fieldDescription>PERSON NAME</fieldDescription></field>
```

1.7.8 Тег GraphicElement *(TBD)*

Тег <graphicElement> – тег, отвечающий за графическую составляющую. Включает в себя линии, треугольники и эллипсы.

Пример использования:

```
<rect>
    <reportElement mode="Opaque" x="5" y="60" width="782" height="80"
forecolor="#000000"/>
    <graphicElement pen="None" fill="Solid"/>
</rect>
```

1.7.9 Тег Group

Тег <group> – это гибкий способ организации данных в отчете. Они представляют собой последовательность записей, которые имеют что-то общее, например, значение некоторого поля. В отчете может быть сколь угодно групп. Порядок групп, заявленных в шаблоне отчета важен, потому что группы вложены одна в другую.

Атрибуты:

- name – обязательный. Название однозначно определяет группу и может быть использовано для других атрибутов, когда необходимо сослаться на конкретную группу в отчете. Название группы является обязательным и подчиняется той же схеме именования, что используется для параметров и переменных отчета.
- IsStartNewColumn – разрыв страницы (true| false) default: "false".

- IsStartNewPage – разрыв колонки (столбца) (true| false) default: "false". Иногда требуется вставить разрыв страницы или колонки, когда начинается новая группа. Для того, чтобы engine вставлял разрыв страницы или колонки каждый раз, когда начинается новая группа, необходимо задать атрибуты IsStartNewPage или IsStartNewColumn соответственно.
- IsReprintHeaderOnEachPage (true| false) default: "false".

Примечание:

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

Например, если вы хотите сгруппировать некоторые продукты по стране или городу производителя, engine рассчитывает найти записи в источнике данных уже упорядоченными по стране и городу.

Если это не так, может получиться, что записи, относящихся к конкретной стране или городу, окажутся в разных частях полученного документа, потому что engine не сортирует данные.

Пример использования:

```
<group name="group_name">
    <groupExpression class="QString">
        <![CDATA[$F{group_name}]]>
    </groupExpression>
    <groupHeader>
        <band height="40">
            <textField>
                <reportElement x="0" y="0" width="200" height="40" />
                <textFieldExpression class="QString">
                    <![CDATA[$F{group_name}]]>
                </textFieldExpression>
            </textField>
        </band>
    </groupHeader>
    <groupFooter>
        <band height="40">
            <textField>
                <reportElement x="0" y="0" width="200" height="40" />
                <textFieldExpression class="QString">
                    <![CDATA[$F{group_name}]]>
                </textFieldExpression>
            </textField>
        </band>
    </groupFooter>
</group>
```

Дочерние элементы:

- <groupExpression> – выражение, по которому будет производиться группировка.
- <groupHeader> – заголовок группы - то, что будет напечатано перед первым элементом.
- <groupFooter> – то, что будет напечатано после последнего элемента группы.

1.7.10 Тег GroupExpression

Тег <groupExpression> – дочерний тег тега <group>, устанавливающий поле, по которому будет происходить группировка. Является обязательным тегом при использовании группировки. Внутри с помощью CDATA описывается поле, по которому будет происходить группировка.

Пример использования:

```
    <groupExpression>
        <![CDATA[$F{group_name}]]>
    </groupExpression>
```

1.7.11 Тег GroupFooter

Тег <groupFooter> – дочерний тег тега <group>, в котором находятся элементы, отображаемые в конце группы. Содержимое описывается с помощью тега band.

Пример использования:

```
 <groupFooter>
   <band height="40">
     <textField>
       <reportElement x="0" y="0" width="100" height="30" />
       <textFieldExpression class="QString">
         <![CDATA[$F{group_name}]]>
       </textFieldExpression>
     </textField>
   </band>
 </groupFooter>
```

1.7.12 Тег GroupHeader

Тег <groupHeader> – дочерний тег тега group, в котором находятся элементы, отображаемые в начале группы. Содержимое описывается с помощью тега band.

Пример использования:

```
 <groupHeader>
     <band height="37">
       <textField>
         <reportElement x="33" y="0" width="100" height="30" />
         <textFieldExpression
class="QString"><![CDATA[$F{group_name}]]></textFieldExpression>
       </textField>
     </band>
 </groupHeader>
```

1.7.13 Тег Image

Тег <image> – Может использоваться для вставки растровых изображений (таких как GIF, PNG и JPEG изображения) в отчете.

Атрибуты:

- scaleImage – указывает на то, как должно быть вынесено изображение, если его фактический размер не подходит под размер элемента отчета изображения. Это происходит потому, что много изображений загружается во время выполнения, и нет

никакого способа узнать их точного размера при создании шаблона отчета.
Возможные значение для этого атрибута:

- Отсечение изображения: если размер изображение имеет больший размер чем выделенная область, то изображение не изменит свой размер, а будет отображаться лишь частично (scaleImage = "Clip").
- Принудительный размер изображения: если размеры фактического изображения не соответствуют указанному для элемента image, который отображается его, изображение будет растягиваться так, что оно будет подходить в обозначенную область вывода. Изображение будет искаженно при необходимости (scaleImage = "FillFrame").
- Сохранение пропорций изображения: если фактическое изображение не помещается в элемент image, оно может быть адаптировано к этим размерам, сохраняя при этом свои первоначальные недеформированные пропорции (scaleImage = "RetainShape").
- Растягивание изображения, сохраняя ширину: Изображение может быть вытянуто по вертикали, чтобы соответствовать фактической высоте изображения, в то время как регулирую ширину элемента изображения, чтобы соответствовать фактической ширине изображения (scaleImage = "RealSize").

Примечание:

Если тип scaleImage - "Clip" или "RetainShape", и фактическое изображение меньше его заданного размера в шаблоне отчета или не имеет тех же пропорций, изображение не может занимать все пространство, выделенное ему в шаблоне отчета. В таких случаях, вы можете задать положение изображения внутри заранее заданного пространства отчетов с помощью "Align" и атрибута VALIGN, который определяют выравнивание изображения по горизонтальной оси (Left, Center, Right) и вертикальной (Top, Middle, Bottom). По умолчанию изображение выравнивают по верхней левой внутренней границе.

Все элементы изображения имеют динамическое содержимое, однако изображения в отчете статические и не обязательно исходят от источника данных или из параметров. Как правило, они загружаются из файлов на диске и представляют собой логотипы и другие статические ресурсы. Чтобы отобразить одно изображение несколько раз в отчете (например, логотип появляется в заголовке страниц), можно кэшировать изображение для лучшей производительности. Когда будет установлен атрибут isUsingCache как "TRUE", обработчик отчетности будет пытаться распознать ранее загруженные изображения, используя их указанный источник. Это функция кэширования для элементов изображения, чьи выражения возвращают объекты любого типа в качестве источника изображения. Флаг isUsingCache устанавливается как "TRUE" по умолчанию для изображений, имеющих String выражения, и как "FALSE" для всех остальных типов. Ключ, используемый для кэша, является значением выражения исходного изображения; ключевые сравнения выполняются с использованием стандартного метода EQUALS. Как следствие, для изображений, имеющих источник InputStream с выключенным кэшированием входной поток считывается только один раз, а затем изображение будет браться из кэша. Флаг isUsingCache не следует устанавливать в тех случаях, когда изображение имеет динамический источник (например, изображение загружается из бинарного поля базы данных для каждой строки), потому что изображения будут накапливаться в кэше и заполнение прекратится из-за ошибки, связанной с отсутствием памяти. Очевидно, что флаг не должен также быть установлен, когда один источник используется для

получения различных изображений (например, URL-адрес, который будет возвращать другое изображение каждый раз, когда это доступно).

- isLazy – флаг, который определят, должно ли быть загружено изображение, и обрабатывается при заполнении отчета или во время экспорта, в случае, если изображение не доступно по время заполнения. По умолчанию этот флаг установлен в "false". Если установлено значение "true", изображение сохраняется во время заполнения вместо самого изображения, и в процессе экспортирования изображение будет загружено с места для чтения с пути. По разным причинам изображение может быть недоступно, когда обработчик пытается загрузить его либо при заполнении отчета, либо во время экспорта, особенно, если изображение загружается из какого-то публичного URL. По этой причине вы можете настроить обработчик, обрабатывающий отсутствующие изображения во время создания отчета. Атрибут OnErrorType для изображений позволяет это. Он может принимать следующие значения:
  - Ошибка: Возникает исключение, если обработчик не может загрузить изображение (OnErrorType="Error").
  - Бланк: Любое исключение image-loading игнорируется, и ничего не будет отображаться в созданном документе (OnError Type="Blank").
  - Значок: Если изображение не загружается успешно, то обработчик поставит небольшой значок в документе, чтобы указать, что фактическое изображение отсутствует (OnErrorType="Icon").
- evaluationTime – как и в случае с текстовыми полями, вы можете отложить вычисление выражения изображения, которое по умолчанию выполняется медленно. Это позволяет отображать в документе изображения, которые будут построены или выбраны позднее в процессе заполнения отчета. Атрибут evaluationTime может принимать следующие значения:
  - Непосредственной оценки: выражение изображения вычисляется, когда заполняется текущий диапазон (evaluationTime="Now").
  - Конец отчета оценки: выражение изображения вычисляется при достижении конца отчета (`evaluationTime="Report")
  - Конец страницы отчета: выражение изображения вычисляется при достижении конца текущей страницы (evaluationTime="Page")
  - Конца столбца оценки: по достижении конца текущего столбца вычисляется выражение изображения (`evaluationTime="Column")
  - Конец группы оценки: выражение изображения вычисляется при группе путем изменения атрибута evaluationGroup (evaluatonTime="Group")
  - Автоматическая оценка: каждой переменной, участвующих в выражении изображения оцениваются в то время, соответствующий типу его сброс. В настоящее время оцениваются поля (evaluationTime="Auto").
- evaluationGroup – группа участвует в процессе оценки изображения, когда атрибут оценки времени устанавливается в группе.

1.7.14 Тег Line

Тег <line> - Определение линейного объекта (линия).

Атрибуты:

- direction (направление) – линии рисуются в виде диагоналей прямоугольника, определяемого свойствами элемента отчета. Этот атрибут указывает, какая из двух диагоналей должна быть нарисована.

Пример использования:

```
<line>
    <reportElement x="100" y="0" width="1" height="100" style="Arial_Normal" />
    <textElement textVAlignment="Bottom" textAlignment="Left">
        <font isBold="true"/>
    </textElement>
</line>
```

1.7.15 Тег Parameter *(TBD)*

Параметр – это ссылка на объекты, которые передаются в процессе заполнения отчета к движку генератора отчета. Не являются обязательными элементами отчета.

Атрибуты:

- name - имя параметра;
- class - класс(тип) значений параметра;

Пример использования:

Объявление параметра:

```
<parameter name="name_parameter" class="name_class" />
```

Что бы использовать параметр, используется конструкция:

$P{name_parameter}, где name_parameter - имя объявленного ранее параметра.

Пример обращения к конкретному параметру:

```
<![CDATA[$P{name_parameter}]]>
```

1.7.16 Тег QueryString

Тег <queryString> – шаблон, необходимый для определения SQL-запроса для данных отчета, если эти данные расположены в реляционных базах данных.

Пример использования:

Существует три возможных способа использовать параметры запроса:

1. Первый способ: $P{paramName} – эти параметры используются как обычные параметры, используя следующий синтаксис:

```
<queryString>
    <!CDATA[
    SELECT *FROM Orders WHERE orderID <= $P{MaxOrderID} ORDER BY ShipCountry
    ]]>
</queryString>
```

2. Второй способ *(TBD)*: $P!{paramName} – иногда бывает полезно использовать параметры для динамического изменения части SQL-запроса или для передачи всего SQL-запроса в качестве параметра для отчета наполнения процедур. В таких случаях синтаксис немного отличается, как показано в примере:

```
<queryString>
  <![CDATA[
  SELECT *FROM $P!{MyTable} ORDER BY $P!{OrderByClause}
  ]]>
```

```
</queryString>
```

3. Третий способ (Не реализован): $X{functionName, param1, param2, ...} – есть случаи, когда часть запроса должна быть динамически построена, исходя из значения параметра отчета, с частью запроса, содержащего текст запроса и связывания параметров. Такие сложные элементы запроса вводятся в запрос с использованием синтаксиса $ X{}. Например, если отчет получается в качестве параметра список стран и должен фильтровать заказы, основанные на этом списке, стоит написать запрос следующего вида:

```
<queryString>
  <![CDATA[
  SELECT *FROM Order WHERE $X{IN, ShipCountry, CountryList}
  ]]>
</queryString>
```

1.7.17 Тег Rect

Тег <rect> – определение объекта прямоугольника.

Атрибуты:

- radius - Радиус угла дуги.

1.7.18 Тег Report

Тег <report> – корневой элемент отчета. С него начинается и им заканчивается каждый отчет.

Обязательные атрибуты:

- Name - название отчета.

Необязательные атрибуты:

- PageWidth - ширина отчета, по умолчанию 595
- PageHeight - высота отчета, по умолчанию 842
- Orientation - ориентация листа отчета, (значения: Portrait,Landscape)

Пример использования:

```
<report>
     Тело отчета
</report>
```

Дочерние элементы:

- <Style>
- <Title>
- <Detail>
- <QueryString>
- <Field>

1.7.19 Тег ReportElement

Тег <reportElement> – первый элемент каждого из дочерних элементов тега <band>. Определяет, как данные размещаются для этого конкретного элемента (задает положение и размер элемента перед которым указан).

Обязательные атрибуты:

- x - координата x
- y - координата y
- width - ширина элемента
- height - высота элемента
- style - стиль

Необязательные атрибуты:

- stretchType - указывает, как текущий элемент растягивается, когда содержится в растянутом элементе band. Значения:
  - NoStretch (default): текущий элемент не растягивается.
  - RelativeToTallestObject: текущий элемент будет растянут, приспосабливаясь к высоким объектам в своей группе.
  - RelativeToBand: текущий элемент будет растянут, соответствуя высоте конкретного элемента band.
- positionType - указывается позиция текущего элемента, когда растянут конкретный элемент band. Значения:
  - Float: текущий элемент будет передвигаться в зависимости от размеров окружающих элементов.
  - FixeRelativeToTop (default): текущий элемент будет сохранять фиксированное положение относительно верхней части элемента band.
  - FixRelativeToBottom: Текущий элемент будет сохранять фиксированное положение относительно нижней части элемента band.
- mode - элементы отчета могут быть прозрачными или непрозрачными в зависимости от значения mode(transparent | opaque). Значения по умолчанию для этого атрибута зависит от типа элемента отчета. Графические элементы, такие как прямоугольники (<rectangle>) и линии (<line>) не прозрачны по умолчанию, в то время как изображения (<image>) являются прозрачными. <staticTexts> и <textFields> являются прозрачными по умолчанию, и поэтому подотчет элементы тоже.

Пример использования:

```
<reportElement x="380" y="0" width="200" height="20" />
```

1.7.20 Тег StaticText

Тег <staticText> – постоянный текст, который не зависит от каких-либо источников данных.

Пример использования:

```
<staticText>
    <reportElement x="20" y="20" width="250" height="20" />
        <textElement textAlignment="Right" textVAlignment="Top" />
    <text><![CDATA[$P{i am staticText}]]></text>
</staticText>
```

Родительский элемент: <band>

Дочерние элементы:

- <reportElement/>
- <textElement/>
- <text></text>

1.7.21 Тег Style

Тег <style> – позволяет единожды определить некоторый набор свойств элементов, а затем использовать этот набор в любом блоке отчета. Стиль применяется для элемента <reportElement> путем указания имени стиля в качестве атрибута style="", в противном случае применяется стиль по умолчанию.

Обязательные атрибуты:

- Name - уникальное имя стиля

Необязательные атрибуты:

- IsDefault - этот стиль будет использоваться по умолчанию (значения: true,false; по умолчанию - false)
- FontSize - размер шрифта
- Forecolor - цвет шрифта (значения: black,blue,gray,green,red,yellow,while)
- FontName - название шрифта
- IsBold - определяет, будет ли текст "жирным" (значения: true,false)
- IsItalic - определяет, будет ли текст "курсивом" (значения: true,false)

Пример использования:

Объявление стилей:

```
<style name="Regular" isDefault="true" fontSize="12" />
<style name="Emphasis" fontSize="12" isBold="tru2 />
```

Применение:

```
<reportElement x="180" y="0" width="200" height="20" />
<text><![CDDATA[стиль по умолчанию]]></text>
<reportElement x="180" y="20" width="200" height="20" style="Emphasis" />
<text><![CDATA[стиль "Emphasis"]]></text>
```

Родительский элемент: <report>

1.7.22 Тег Summary

Тег Summary – это один из типов секций. Располагается один раз в конце отчета после секции Detail.

Пример использования:

```
<summary>
    <band height="35">
          <textElement textAlignment="Center" textVAlignment="Middle">
                <staticText>
                      <reportElement x="10" y="10" width="150" height="20" />
```

```
                        <text><![CDATA[$P{summary}]]></text>
                  </staticText>
            </textElement>
            <textField>
                  <textElement textAlignment="Center" textVAlignment="Bottom">
                  <reportElement x="310" y="0" width="140" height="40" />
                  <textFieldExpression
class="QString"><![CDATA[$F{NManual}]]></textFieldExpression>
                  </textElement>
            </textField>
            <rect>
                  <reportElement x="0" y="0" width="480" height="30" />
                  <graphicElement pen="Dotted" />
            </rect>
            <line>
                  <reportElement x="170" y="10" width="20" height="40" />
            </line>
            <line>
                  <reportElement x="200" y="30" width="20" height="-40" />
            </line>
      </band>
</summary>
```

Дочерние элементы:

- textField

1.7.23 Тег Text

Тег <text> – определяет фактический статический текст, отображаемый в отчете.

В приведенном ниже шаблоне мы решили добавить XML CDATA раздел между тегами и </text>. Хотя нет строго необходимо в этом случае, это позволяет нам легко модифицировать текст между этими тегами, чтобы включить текст, который помешал бы XML от синтаксического анализа.

Пример использования:

```
<text><![CDATA[$P{title}]]></text>
```

Родительский элемент: <staticText>

1.7.24 Тег TextElement

Тег <textElement>

Необязательные атрибуты:

- textAlignment - выравнивание текста (значения: Left,Center,Right,Justified)
- textVAlignment - вертикальное выравнивание текста (значения: Top,Middle,Bottom)

Пример использования:

```
<textElement textAlignment="Left" textVAlignment="Middle">
```

1.7.25 Тег TextField *(TBD)*

Тег <textField> – в отличие от статических текстовых элементов, которые не изменяют содержание их текста, текстовые поля имеют ассоциированное выражение, которое вычисляется с каждой итерацией в источнике данных, чтобы получить текстовое содержимое, которое будет отображаться.

Необязательные атрибуты:

- pattern
- isBlankWhenNull - (значения: true,false)
- isStretchWithOverflow (значения: true,false; по умолчанию - false)
- evaluationTime (значения: Now,Report,Page,Column,Group,Band,Auto; по умолчанию - Auto)

Пример использования:

```
<textField>
    <reportElement x="220" y="20" width="100" height="200" />
    <textFieldExpression class="QString">
        <![CDATA[$P{title}]]>
    </textFieldExpression>
</textField>
```

Родительский элемент: <band>

Дочерние элементы:

- <reportElement>
- <textElement>
- <textFieldExpression>

1.7.26 Тег TextFieldExpression

Тег <TextFieldExpression> – текстовое поле может возвращать значения только ограниченного диапазона перечисленных классов следующим образом:

- Boolean
- Byte
- Date
- Timestamp
- Time
- Double
- Float
- Integer
- Long
- Short
- BigDecimal
- Number
- QString (По умолчанию)

Атрибуты: class - класс для значений текстового поля.

Пример использования:

```
<textFieldExpression class="QString">
```

```
    <![CDATA[$P{title}]]>
</textFieldExpression>
```

1.7.27 Тег Title

Тег <title> – название отчета, отображается один раз в начале отчета.

Пример использования:

```
<title>
    <band>
        <staticText>
            <reportElement x="100" y="16" width="100" height="20"/>
            <textElement verticalAlignment="Top" textAlignment="Right"/>
            <text><![CDATA[Title]]></text>
        </staticText>
    </band>
</title>
```

Родительский элемент: <report>

Дочерние элементы: <band>

1.7.28 Тег Variable

Тег <variable> – 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 (или формуле) переменная может использовать другие переменные, поля или параметры. Переменные вычисляются/инкрементируются с каждой записью из источника данных в том порядке, в котором они объявлены.

Атрибуты:

- name - имя переменной. Является обязательным атрибутом и позволяет ссылаться на переменную по этому имени.
- class - тип данных, которому принадлежит значение переменой(string - by default).
- resetType - периодичность установки исходного значения(None|Report|Page|Column|Group). Варианты значений:
    - None - никогда не инициализируется начальным значением. Содержит значение полученное путем вычисления выражения переменной.
    - Report - инициализируется начальным значением (<initialValueExpression>) один раз в начале заполнения отчета (by Default).
    - Page - инициализируется заново в начале каждой страницы.
    - Column - инициализируется заново вначале каждого нового столбца.
    - Group - инициализируется заново каждый раз, когда задается новая группа.
- resetGroup - если resetGroup используется, то он содержит имя группы и работает только в сочетании с атрибутом resetType, значение которого будет resetType=Group.
- incrementType - периодичность приращения переменной (None | Report | Page | Column | Group). По умолчанию переменная инкрементируется с каждой записью из источника данных. Варианты значений:
    - None - переменная инкрементируется с каждой записью (by Default).
    - Report - переменная никогда не инкрементируется.
    - Page - переменная инкрементируется с каждым новой страницей.

    - Column - переменная инкрементируется с каждым новым столбцом.
    - Group - инкрементируется каждый раз, когда задается новая группа.
- incrementGroup - если incrementGroup используется, то он содержит имя группы и работает только в сочетании с атрибутом incrementType, значение которого будет incrementType=Group.
- calculation - агрегатная функция(Count|Sum|Average|Lowest|Highest).

Примечания:

Выражение, задающее значение переменной.

```
<variableExpression><![CDATA[$F{absence}]]></variableExpression>
```

Тег, задающий исходное значение. (может не использоваться)

```
<initialValuesExpression><![CDATA[0]]></initialValuesExpression>
```

Встроенные переменные:

- PAGE_NUMBER - содержит номер текущей страницы. В конце заполнения отчета содержит общее число страниц в документе. Может использоваться для отображения текущей страницы и количества страницы одновременно.
- COLUMN_NUMBER - содержит номер текущей колонки. Переменная подсчитывает количество колонок для каждой страницы.
- REPORT_COUNT - содержит общее число обработанных записей в отчете.
- PAGE_COUNT - содержит количество записей обработанных в процессе заполнения текущей страницы.
- COLUMN_COUNT - число записей, обработанных при генерации текущей колонки.

Пример использования:

```
<variable name="sumAbsenceByDiscipline" class="Integer" resetType="Group"
resetGroup="Discipline" calculation="Sum">
    <variableExpression><![CDATA[$F{absence}]]></variableExpression>
    <initialValueExpression><![CDATA[0]]></initialValueExpression>
</variable>
<textField>
    <reportElement x="396" y="0" width="159" height="34"/>
    <textElement verticalAlignment="Middle"/>
<textFieldExpression><![CDATA[$V{sumAbsenceByDiscipline}]]></textFieldExpression>
</textField>`
```

1.7.29 Настройка параметров отчета *(TBD)*

1.7.30 Свойства столбцов *(TBD)*

1.7.31 Расширенные настройки *(TBD)*

Глава 2 Основные понятия QtReports

Эта глава иллюстрирует базовые концепции QtReports для лучшего понимания того, как работает данная библиотека для генерации отчетов.

API QtReports, синтаксис XML для определения отчетов и все подробности использования библиотеки в Ваших собственных программах описаны в данном руководстве. Другая информация и примеры доступны по адресу: https://github.com/PO-31/QtReports

2.1 Файлы .QREPORT и .QRXML

QtReports определяет отчет с помощью файла XML. Файл .qrxml и файл .qreport состоят из набора разделов, некоторые из которых касаются физических характеристик, таких как размер страницы, расположение полей и высота строк; и некоторые из них относятся к определению логических характеристик, таких как объявление параметров и переменных, а также созданию запросов для выборки данных.

Пример структуры .qrxml файла:

```
<?xml version="1.0" encoding="UTF-8"?>
<report name="sample_report" orientation="Landscape">
  <style name="Arial_Normal" isDefault="true" fontName="Arial"
    fontSize="12" pdfFontName="c:\tahoma.ttf" pdfEncoding="Cp1251"
    isPdfEmbedded="false" />
  <queryString>
    <![CDATA[ select idImg, nameImg, image from images; ]]>
  </queryString>
  <field name="idImg" class="QString" />
  <field name="nameImg" class="QString" />
  <field name="image" class="QString" />
  <group name="nameImg">
    <groupExpression>
      <![CDATA[$F{nameImg}]]>
    </groupExpression>
    <groupHeader>
      <band height="40">
```

```
        <textField>
          <reportElement x="0" y="10" width="50" height="30"/>
          <textFieldExpression
class="QImage"><![CDATA[$F{nameImg}]]></textFieldExpression>
        </textField>
      </band>
    </groupHeader>
    <groupFooter>
      <band height="10">
      </band>
    </groupFooter>
  </group>
  <title>
    <band height="35">
      <staticText>
        <reportElement x="10" y="10" width="150" height="20" />
        <text><![CDATA[$P{title}]]></text>
      </staticText>
      <ellipse>
        <reportElement x="0" y="0" width="535" height="35" />
      </ellipse>
    </band>
  </title>
  <detail>
    <band height="200">
      <rect>
        <reportElement x="0" y="0" width="535" height="200" />
      </rect>
      <textField>
        <reportElement x="5" y="0" width="20" height="200" />
        <textFieldExpression
class="QString"><![CDATA[$F{idImg}]]></textFieldExpression>
      </textField>
      <line>
        <reportElement x="25" y="0" width="1" height="200" />
      </line>
      <textField>
        <reportElement x="30" y="0" width="140" height="200" />
        <textFieldExpression
class="QString"><![CDATA[$F{nameImg}]]></textFieldExpression>
      </textField>
      <line>
        <reportElement x="170" y="0" width="1" height="200" />
      </line>
      <image>
        <reportElement x="170" y="0" width="425" height="200" />
        <imageExpression class="QString"><![CDATA[$F{image}]]></imageExpression>
      </image>
    </band>
  </detail>
</report>
```

Пример структуры .qreport файла:

```
<?xml version="1.0" encoding="UTF-8"?>
<report name="sample_report">
  <style name="Arial_Normal" isDefault="true" fontName="Arial"
    fontSize="12" pdfFontName="c:\tahoma.ttf" pdfEncoding="Cp1251"
```

```
    isPdfEmbedded="false" />
  <queryString>
    <![CDATA[ select group_name, stud_name from groups_t NATURAL JOIN students_t
ORDER BY group_name]]>
  </queryString>
  <field name="group_name" class="QString" />
  <field name="stud_name" class="QString" />
<group name="group_name">
        <groupExpression class="QString">
            <![CDATA[$F{group_name}]]>
        </groupExpression>
  <groupHeader>
      <band height="37">
        <textField>
          <reportElement x="33" y="0" width="100" height="30" />
          <textFieldExpression
class="QString"><![CDATA[$F{group_name}]]></textFieldExpression>
        </textField>
      </band>
  </groupHeader>
  <groupFooter>
    <band height="40">
      <textField>
        <reportElement x="0" y="0" width="100" height="30" />
        <textFieldExpression class="QString">
          <![CDATA[$F{group_name}]]>
        </textFieldExpression>
      </textField>
    </band>
  </groupFooter>
</group>
  <detail>
    <band height="40">
      <staticText>
        <reportElement x="380" y="0" width="200" height="20" />
        <text><![CDATA[Тест !!]]></text>
      </staticText>
      <textField>
        <reportElement x="220" y="20" width="100" height="20" />
        <textFieldExpression class="QString">
          <![CDATA[$P{title}]]>
        </textFieldExpression>
      </textField>
      <textField>
        <reportElement x="51" y="0" width="200" height="20" />
        <textFieldExpression class="QString">
          <![CDATA[$F{group_name}]]>
        </textFieldExpression>
      </textField>
      <textField>
        <reportElement x="252" y="0" width="200" height="20" />
        <textFieldExpression class="QString">
          <![CDATA[$F{stud_name}]]>
        </textFieldExpression>
      </textField>
    </band>
  </detail>
</report>
```

2.2 Источники данных и форматы для печати

QtReports работает со всеми СУБД, поддерживаемыми фреймворком Qt. Для экспорта и печати доступны форматы PDF и HTML.

2.3 Совместимость между версиями

2.4 Выражения

2.4.1 Типы выражений

2.4.3 Использование конструкции if-else в выражениях

2.5 Простейшая программа

В следующем примере кода показано, как использовать QtReports в программах Qt.

.pro file:

```
QT += core gui sql widgets printsupport

TARGET = qtreports-viewer
TEMPLATE = app
CONFIG += c++14
INCLUDEPATH += ../qtreports/include/
LIBS   += -L"$$PWD"/../build/lib/
LIBS   += -lqtreports
DESTDIR = "$$PWD"/../build/viewer/

SOURCES += src/main.cpp

QMAKE_CXXFLAGS += -std=c++14

message("Using LIB: $$LIBS")
message("Using spec: $$QMAKESPEC")
message("Compiler: $$QMAKE_CXX")
```

.cpp file:

```
#include <QApplication>
#include <QMessageBox>
#include <QMainWindow>
#include <QAction>
#include <QMenu>
#include <QMenuBar>
#include <QFileDialog>
#include <QDir>
#include <QSqlDatabase>
#include <QSqlError>
#include <qtreports/engine.hpp>

#ifdef WIN32
#include <Windows.h>
```

```
#endif

static QMainWindow* mainWindow = nullptr;
static QMenuBar* menuBar = nullptr;
static QMenu* fileMenu = nullptr;
static QMenu* convertMenu = nullptr;

static QAction* printAction = nullptr;
static QAction* createPdfAction = nullptr;
static QAction* createHtmlAction = nullptr;
static QAction* selectDatabaseAction = nullptr;
static QAction* closeAction = nullptr;
static QAction* openAction = nullptr;
static QAction* exitAction = nullptr;

static QSharedPointer<QWidget> layout;

static qtreports::Engine* engine = nullptr;

void showError(const QString& text);
void closeReport();
void openReport(const QString& reportPath);
void openReport();
void openDatabase(const QString& dbPath);
void openDatabase();

void showError(const QString& text)
{
    QMessageBox::warning(nullptr, "Error: ", text);
}

void closeReport()
{
    engine->close();
    layout.clear();
    if (mainWindow->centralWidget() != nullptr) {
        delete mainWindow->centralWidget();
    }
    convertMenu->setEnabled(false);
    printAction->setEnabled(false);
    closeAction->setEnabled(false);
    selectDatabaseAction->setEnabled(false);
    mainWindow->setWindowTitle("QtReports viewer");
}

void openReport(const QString& reportPath)
{
    closeReport();

    bool result = engine->open(reportPath);
    closeAction->setEnabled(result);
    selectDatabaseAction->setEnabled(result);
    if (!result) {
        showError(engine->getLastError());
        return;
    }
    QFileInfo fileInfo(reportPath);
    mainWindow->setWindowTitle("QtReports viewer - " + fileInfo.fileName());
    layout = engine->createLayout();
```

```
    if (layout.isNull()) {
        showError("Widget is empty");
        engine->close();
        return;
    }
    mainWindow->setCentralWidget(layout.data());
    mainWindow->resize(layout->size());
}

void openReport()
{
    auto reportPath = QFileDialog::getOpenFileName(
        mainWindow,
        QObject::tr("Open QReport"),
        QString(),
        QObject::tr("QReport Files (*.qreport *.qrxml);;All Files (*.*)")
    );
    if (reportPath.isEmpty()) {
        return;
    }
    openReport(reportPath);
}

void openDatabase(const QString& dbPath)
{
    auto db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName(dbPath);
    if (!db.open()) {
        showError("Can not open database. Database error: " +
db.lastError().text());
        return;
    }
    bool result = engine->setConnection(db);
    convertMenu->setEnabled(result);
    printAction->setEnabled(result);
    if (!result) {
        showError(engine->getLastError());
        return;
    }

    //setting report parameters
    QMap <QString, QVariant> map;
    map["title"] = "TITLE";
    map["idPlan"] = "2";
    map["summary"] = "SUMMARY";

    if (!engine->setParameters(map)) {
        showError(engine->getLastError());
    }
}

void openDatabase()
{
    auto dbPath = QFileDialog::getOpenFileName(
        mainWindow,
        QObject::tr("Open DataBase"),
        QString(),
        QObject::tr("DataBase files (*.db);;All Files (*.*)")
    );
```

```
    if (dbPath.isEmpty()) {
        return;
    }
    openDatabase(dbPath);
}

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

    mainWindow = new QMainWindow();
    mainWindow->setWindowTitle("QtReports viewer");
    mainWindow->resize(800, 600);
    menuBar = new QMenuBar(mainWindow);
    fileMenu = new QMenu("File", menuBar);
    convertMenu = new QMenu("Convert", menuBar);

//For Debug in Windows
#ifdef WIN32
    AllocConsole();
    freopen("CONOUT$", "w", stdout);
    freopen("CONOUT$", "w", stderr);
#endif

    printAction = new QAction(QObject::tr("&Print..."), mainWindow);
    printAction->setShortcuts(QKeySequence::Print);
    printAction->setStatusTip(QObject::tr("Print current report"));
    QObject::connect(printAction, &QAction::triggered, [&]() {
        if (!engine->print()) {
            showError(engine->getLastError());
        }
    });

    createPdfAction = new QAction(QObject::tr("&Create PDF..."), mainWindow);
    createPdfAction->setStatusTip(QObject::tr("Create PDF from current report"));
    QObject::connect(createPdfAction, &QAction::triggered, [&]() {
        auto file = QFileDialog::getSaveFileName(
            mainWindow,
            QObject::tr("Save as PDF"),
            QString(),
            QObject::tr("PDF Files (*.pdf)")
        );
        if (file.isEmpty()) {
            return;
        }
        if (!engine->createPDF(file)) {
            showError(engine->getLastError());
            return;
        }
    });

    createHtmlAction = new QAction(QObject::tr("&Create HTML..."), mainWindow);
    createHtmlAction->setStatusTip( QObject::tr("Create HTML from current
report"));
    QObject::connect(createHtmlAction, &QAction::triggered, [&]() {
        auto file = QFileDialog::getSaveFileName(
            mainWindow,
            QObject::tr("Save as HTML"),
            QString(),
            QObject::tr("HTML Files (*.html *.htm)")
```

```
        );
        if (file.isEmpty()) {
            return;
        }

        bool result = engine->createHTML(file);
        if (!result) {
            showError(engine->getLastError());
            return;
        }
    });

    selectDatabaseAction = new QAction(QObject::tr("&Select database..."),
mainWindow);
    selectDatabaseAction->setStatusTip(QObject::tr("Select database file"));
    QObject::connect(selectDatabaseAction, &QAction::triggered, [&]() {
openDatabase(); });

    closeAction = new QAction(QObject::tr("&Close report"), mainWindow);
    closeAction->setShortcuts(QKeySequence::Close);
    closeAction->setStatusTip(QObject::tr("Close current report"));
    QObject::connect(closeAction, &QAction::triggered, [&]() { closeReport(); });

    openAction = new QAction(QObject::tr("&Open report..."), mainWindow);
    openAction->setShortcuts(QKeySequence::Open);
    openAction->setStatusTip(QObject::tr( "Open an existing report file"));
    QObject::connect(openAction, &QAction::triggered, [ & ]() { openReport(); });

    exitAction = new QAction(QObject::tr("&Exit"), mainWindow);
    exitAction->setShortcuts( QKeySequence::Quit );
    QObject::connect(exitAction, &QAction::triggered, mainWindow,
&QMainWindow::close);

    fileMenu->addActions({ openAction, selectDatabaseAction, printAction,
closeAction, exitAction });
    convertMenu->addActions({ createPdfAction, createHtmlAction });
    menuBar->addMenu(fileMenu);
    menuBar->addMenu(convertMenu);
    mainWindow->setMenuBar(menuBar);

    engine = new qtreports::Engine();
    closeReport();

    if (argc > 1) {
        openReport(argv[1]);
    }
    if (argc > 2) {
        openDatabase(argv[2]);
    }
    mainWindow->show();

    return a.exec();
}
```

Глава 3 Создание простейшего отчета

3.1 Создание нового отчета

3.2 Добавление и удаление элементов отчета

3.2.1 Добавление полей в отчет

3.2.2. Удаление полей

3.2.3 Добавление других элементов

3.3 Предпросмотр отчета

3.4 Создание каталога проекта

Глава 4 Работа с полями

4.1 Распознавание полей

4.2 Регистрация полей из SQL-запроса

4.3 Поля и текстовые поля

Глава 5 Шаблоны отчетов

5.1 Структура шаблона

5.2 Создание и кастомизация шаблонов

5.2.1 Создание нового шаблона

5.2.2 Кастомизация шаблона

5.3 Сохранение шаблонов

5.3.1 Создание директории шаблона

5.3.2 Экспорт шаблона

5.4 Добавление шаблона в QtReports

Глава 6 Использование параметров

6.1 Управление параметрами

6.2 Параметры по умолчанию

6.3 Использование параметров в запросах

6.3.1 Использование параметров в SQL-запросах

В QtReports существует возможность использования параметров запроса ($P{paramName}). Пример использования приведен ниже:

```
<queryString>
    <!CDATA[
    SELECT *FROM Orders WHERE orderID <= $P{MaxOrderID} ORDER BY ShipCountry
    ]]>
</queryString>
```

6.3.2 Использование параметров с null-значениями

6.3.3 Относительные даты

6.3.4 Передача параметров из программы

В QtReports существует возможность передачи параметров из программы при помощи метода setParameters(QMap map). Пример использования приведен ниже:

```
qtreports::Engine engine;
QVERIFY2( engine.open( reportPath ), engine.getLastError().toStdString().c_str()
);
QMap < QString, QVariant > map;
map[ "title" ] = "Best Title in World";
map[ "param1" ] = "Best param1 in World";
qDebug() << endl << "Used map: " << map;
QVERIFY2( engine.setParameters( map ), engine.getLastError().toStdString().c_str()
);
```

Глава 7 Переменные

7.1 Определение новой переменной или изменение существующей

7.2 Основные свойства переменной

7.3 Дополнительные свойства переменной

7.3.1 Время оценки

7.3.2 Вычислительная функция

7.3.3 Тип приращения

7.3.4 Тип сброса

7.4 Встроенные переменные