Оглавление

[QT – ВВЕДЕНИЕ 28](#_Toc382058122)

[СРЕДСТВА РАЗРАБОТКИ 28](#_Toc382058123)

[QT CREATOR 28](#_Toc382058124)

[QT DESIGNER 29](#_Toc382058125)

[DESIGNER MODES 30](#_Toc382058126)

[МОДА РЕДАКТИРОВАНИЯ 30](#_Toc382058127)

[МОДА РЕДАКТИРОВАНИЯ СИГНАЛОВ И СЛОТОВ 31](#_Toc382058128)

[МОДА ОПРЕДЕЛЕНИЯ ПАРТНЁРОВ 31](#_Toc382058129)

[МОДА ПОРЯДКА ТАБУЛЯЦИИ 32](#_Toc382058130)

[ШАБЛОН ГЛАВНОГО ОКНА 32](#_Toc382058131)

[РЕДАКТИРОВАНИЕ РЕСУРСОВ 33](#_Toc382058132)

[РЕДАКТОР СТИЛЕЙ 33](#_Toc382058133)

[UI ФАЙЛЫ. СПОСОБЫ ИСПОЛЬЗОВАНИЯ ФОРМ ДИЗАЙНЕРА. АВТОМАТИЧЕСКОЕ И РУЧНОЕ СОЕДИНЕНИЕ СИГНАЛОВ И СЛОТОВ 33](#_Toc382058134)

[ЧАСТНЫЕ ВИДЖЕТЫ В ДИЗАЙНЕРЕ. ПРОДВИЖЕНИЕ ВИДЖЕТОВ. 35](#_Toc382058135)

[СОЗДАНИЕ ЧАСТНЫХ ВИДЖЕТОВ ДЛЯ QT ДИЗАЙНЕРА ПРИ ПОМОЩИ ПЛАГИНОВ 36](#_Toc382058136)

[РАСШИРЕНИЯ ДЛЯ ЧАСТНЫХ ВИДЖЕТОВ 37](#_Toc382058137)

[XML-СХЕМА ДЛЯ ФОРМАТА UI ФАЙЛА 38](#_Toc382058138)

[МАКЕТЫ В ДИЗАЙНЕРЕ 38](#_Toc382058139)

[СОХРАНЕНИЕ, ПРОСМОТР И ПЕЧАТЬ ФОРМ В QT ДИЗАЙНЕРЕ 39](#_Toc382058140)

[QMAKE 40](#_Toc382058141)

[ОБЗОР QMAKE 40](#_Toc382058142)

[НЕКОТОРЫЕ ОСОБЕННОСТИ QMAKE 41](#_Toc382058143)

[СОЗДАНИЕ ФАЙЛОВ ПРОЕКТА 41](#_Toc382058144)

[ПОСТРОЕНИЕ ПРОЕКТОВ ОБЩИХ ТИПОВ 44](#_Toc382058145)

[ВЫПОЛНЕНИЕ QMAKE В КОМАНДНОЙ СТРОКЕ. МОДЫ И НАСТРОЙКИ. 45](#_Toc382058146)

[СПЕЦИФИЧЕСКИЕ ПАРАМЕТРЫ ДЛЯ ПЛАТФОРМ 47](#_Toc382058147)

[ЯЗЫК QMAKE 47](#_Toc382058148)

[ДОБАВЛЕНИЕ НОВЫХ ОСОБЕННОСТЕЙ КОНФИГУРАЦИИ 48](#_Toc382058149)

[ИСПОЛЬЗОВАНИЕ ПРЕДКОМПИЛИРОВАННЫХ ЗАГОЛОВКОВ 51](#_Toc382058150)

[КОНФИГУРИРОВАНИЕ QMAKE 51](#_Toc382058151)

[ССЫЛКИ 52](#_Toc382058152)

[ПЕРЕМЕННЫЕ 53](#_Toc382058153)

[ФУНКЦИИ ЗАМЕЩЕНИЯ 55](#_Toc382058154)

[ТЕСТОВЫЕ ФУНКЦИИ 55](#_Toc382058155)

[QT LINGUIST 55](#_Toc382058156)

[ОБЗОР ПРОЦЕССА ПЕРЕВОДА 55](#_Toc382058157)

[РУКОВОДСТВО ПО ЛИНГВИСТУ ДЛЯ МЕНЕДЖЕРОВ 56](#_Toc382058158)

[РУКОВОДСТВО ПО ЛИНГВИСТУ ДЛЯ ПЕРЕВОДЧИКОВ 57](#_Toc382058159)

[РУКОВОДСТВО ПО ЛИНГВИСТУ ДЛЯ РАЗРАБОТЧИКОВ 58](#_Toc382058160)

[TS ФОРМАТ ФАЙЛОВ 59](#_Toc382058161)

[ПЕРЕВОДЫ, ОСНОВАННЫЕ НА ТЕКСТОВЫХ ИДЕНТИФИКАТОРАХ 60](#_Toc382058162)

[ИНТЕРНАЦИОНАЛИЗАЦИЯ И ЛОКАЛИЗАЦИЯ 60](#_Toc382058163)

[НАПИСАНИЕ ИСХОДНОГО КОДА ДЛЯ ПЕРЕВОДА 62](#_Toc382058164)

[МЕТА-ОБЪЕКТНЫЙ КОМПИЛЯТОР 64](#_Toc382058165)

[КОМПИЛЯТОР ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ (UIC) 65](#_Toc382058166)

[КОМПИЛЯТОР РЕСУРСОВ (RCC) 66](#_Toc382058167)

[QT CREATOR 66](#_Toc382058168)

[ОБЗОР 66](#_Toc382058169)

[МОДА ПРИВЕТСТВИЯ 66](#_Toc382058170)

[НАСТРОЙКА СОЗДАТЕЛЯ 68](#_Toc382058171)

[ПОСТРОЕНИЕ И ЗАПУСК ПРИМЕРА 69](#_Toc382058172)

[СОЗДАНИЕ ПРИЛОЖЕНИЯ QT QUICK 69](#_Toc382058173)

[ДОБАВЛЕНИЕ НАБОРОВ ИНСТРУМЕНТОВ 69](#_Toc382058174)

[ДОБАВЛЕНЕИМ ВЕРСИЙ QT 70](#_Toc382058175)

[ДОБАВЛЕНИЕ КОМПИЛЯТОРА 70](#_Toc382058176)

[УПРАВЛЕНИЕ ПРОЕКТАМИ 70](#_Toc382058177)

[СОЗДАНИЕ ПРОЕКТОВ 70](#_Toc382058178)

[ОТКРЫТИЕ ПРОЕКТОВ 72](#_Toc382058179)

[ДОБАВЛНИЕ БИБЛИОТЕК В ПРОЕКТ 72](#_Toc382058180)

[ДОБАВЛЕНИЕ НОВЫХ ЧАСТНЫХ ВИЗАРДОВ 73](#_Toc382058181)

[ИСПОЛЬЗОВАНИЕ СИСТЕМ КОНТРОЛЯ ВЕРСИЙ 74](#_Toc382058182)

[КОНФИГУРИРОВАНИЕ ПРОЕКТА 75](#_Toc382058183)

[ОПРЕДЕЛЕНИЕ НАСТРОЕК ПОСТРОЕНИЯ 75](#_Toc382058184)

[ОПРЕДЕЛЕНИЕ НАСТРОЕК ВЫПОЛНЕНИЯ 76](#_Toc382058185)

[ОПРЕДЕЛЕНИЕ НАСТРОЕК РЕДАКТОРА 77](#_Toc382058186)

[ОПРЕДЕЛЕНИЕ НАСТРОЕК СТИЛЯ КОДА 77](#_Toc382058187)

[ОПРЕДЕЛЕНИЕ ЗАВИСИМОСТЕЙ 77](#_Toc382058188)

[РАЗДЕЛЕНИЕ НАСТРОЕК ПРОЕКТА 77](#_Toc382058189)

[УПРАВЛЕНИЕ СЕССИЯМИ 78](#_Toc382058190)

[РАЗРАБОТКА ИНТЕРФЕЙСОВ ПОЛЬЗОВАТЕЛЕЙ 78](#_Toc382058191)

[РАЗРАБОТКА QT QUICK ПРИЛОЖЕНИЙ 78](#_Toc382058192)

[СОЗДАНИЕ ПРОЕКТОВ В QT QUICK 79](#_Toc382058193)

[ИСПОЛЬЗОВАНИЕ QT QUICK ДИЗАЙНЕРА 79](#_Toc382058194)

[СОЗДАНИЕ КОМПОНЕНТОВ 81](#_Toc382058195)

[РАЗРАБОТКА ОСНОВАННЫХ НА ВИДЖЕТАХ ПРИЛОЖЕНИЙ 82](#_Toc382058196)

[ДОБАВЛЕНИЕ ПЛАГИНОВ В QT DESIGNER 82](#_Toc382058197)

[КОДИРОВАНИЕ 83](#_Toc382058198)

[РАБОТА В МОДЕ РЕДАКТИРОВАНИЯ 83](#_Toc382058199)

[СЕМАНТИЧЕСКОЕ ВЫДЕЛЕНИЕ 85](#_Toc382058200)

[ПРОВЕРКА СИНТАКСИСА КОДА 85](#_Toc382058201)

[ЗАВЕРШЕНИЕ КОДА 85](#_Toc382058202)

[ОТСТУПЫ ТЕКСТА ИЛИ КОДА 86](#_Toc382058203)

[ИСПОЛЬЗОВАНИЕ МАКРОСОВ РЕДАКТИРОВАНИЯ ТЕКСТА 87](#_Toc382058204)

[СРАВНЕНИЕ ФАЙЛОВ 87](#_Toc382058205)

[ПОИСК 87](#_Toc382058206)

[ПОИСК ПРИ ПОМОЩИ ЛОКАТОРА 87](#_Toc382058207)

[РЕФАКТОРИНГ 88](#_Toc382058208)

[КОНФИГУРИРОВАНИЕ РЕДАКТОРА 88](#_Toc382058209)

[ВСТАВКА ФРАГМЕНТОВ КОДА 88](#_Toc382058210)

[ИЗМЕНЕНИЕ ЦВЕТОВОЙ СХЕМЫ 88](#_Toc382058211)

[FakeVim 89](#_Toc382058212)

[РЕДАКТИРОВАНИЕ MIME-ТИПОВ 89](#_Toc382058213)

[ПОСТРОЕНИЕ И ЗАПУСК 89](#_Toc382058214)

[ПОСТРОЕНИЕ ДЛЯ МНОГИХ ПЛАТФОРМ 90](#_Toc382058215)

[ЗАПУСК НА МНОГИХ ПЛАТФОРМАХ 90](#_Toc382058216)

[ОТНОШЕНИЕ МЕЖДУ ПОСТРОЕНИЕМ, ЗАПУСКОМ И РАЗВЁРТЫВАНИЕМ 91](#_Toc382058217)

[ОТЛАДКА И АНАЛИЗ 91](#_Toc382058218)

[ОТЛАДКА 91](#_Toc382058219)

[УСТАНОВКА ОТЛАДЧИКА 92](#_Toc382058220)

[ВЗАИМОДЕЙСТВИЕ С ОТЛАДЧИКОМ 93](#_Toc382058221)

[ИСПОЛЬЗОВАНИЕ ВСПОМОГГАТЕЛЬНЫХ ФУНКЦИЙ ОТЛАДКИ 95](#_Toc382058222)

[ПРОБЛЕМЫ ПРИ ОТЛАДКЕ 96](#_Toc382058223)

[АНАЛИЗ КОДА 96](#_Toc382058224)

[VALGRID 97](#_Toc382058225)

[ДЕТЕКТИРОВАНИЕ ПОТЕРЬ ПАМЯТИ 97](#_Toc382058226)

[ПРОФАЙЛИНГ ВЫПОЛНЕНИЯ ФУНКЦИЙ 97](#_Toc382058227)

[ПРОДВИНУТОЕ ИСПОЛЬЗОВАНИЕ QT CREATOR 98](#_Toc382058228)

[CMAKE 98](#_Toc382058229)

[ИСПОЛЬЗОВАНИЕ НАСТРОЕК КОМАНДНОЙ СТРОКИ 98](#_Toc382058230)

[ГОРЯЧИЕ КЛАВИШИ 99](#_Toc382058231)

[ВНЕШНИЕ ИНСТРУМЕТНЫ 99](#_Toc382058232)

[ПОКАЗ ФАЙЛОВ СПИСКА ЗАДАЧ НА ПАНЕЛИ ISSUE 99](#_Toc382058233)

[СПРАВКА 99](#_Toc382058234)

[МОДА СПРАВКИ 99](#_Toc382058235)

[FAQ 99](#_Toc382058236)

[СОВЕТЫ И ХИТРОСТИ 99](#_Toc382058237)

[ИЗВЕСТНЫЕ БАГИ 99](#_Toc382058238)

[СЛОВАРЬ ТЕРМИНОВ QT CREATOR 100](#_Toc382058239)

[ВСЕ МОДУЛИ QT 101](#_Toc382058240)

[МОДУЛЬ ЯДРА 102](#_Toc382058241)

[ОБЪЕКТНАЯ МОДЕЛЬ 102](#_Toc382058242)

[ОБЪЕКТНЫЕ ДЕРЕВЬЯ И ВЛАДЕНИЕ 103](#_Toc382058243)

[МЕТА-ОБЪЕКТНАЯ СИСТЕМА QT 104](#_Toc382058244)

[СИСТЕМА СВОЙСТВ 105](#_Toc382058245)

[МЕХАНИЗМ СИГНАЛОВ И СЛОТОВ 107](#_Toc382058246)

[СИСТЕМА РЕСУРСОВ 109](#_Toc382058247)

[КЛАССЫ ОБЪЕКТНОЙ МОДЕЛИ 111](#_Toc382058248)

[Qt пространство имён 111](#_Toc382058249)

[Q\_DECLARE\_METATYPE() 111](#_Toc382058250)

[QVariant 112](#_Toc382058251)

[QMetaObject 113](#_Toc382058252)

[QMetaMethod 113](#_Toc382058253)

[QMetaEnum 113](#_Toc382058254)

[QMetaProperty 113](#_Toc382058255)

[QMetaTypeInfo 113](#_Toc382058256)

[QObject 114](#_Toc382058257)

[QObjectCleanUpHandler 115](#_Toc382058258)

[QPointer 115](#_Toc382058259)

[QSharedPointer 115](#_Toc382058260)

[QWeakPointer (устарел) 116](#_Toc382058261)

[QSharedDataPointer 117](#_Toc382058262)

[QExplicitlySharedDataPointer 118](#_Toc382058263)

[СИСТЕМА ВВОДА/ВЫВОДА 118](#_Toc382058264)

[QImageReader 120](#_Toc382058265)

[QImageWriter 120](#_Toc382058266)

[QTextDocumentWriter 120](#_Toc382058267)

[QFileDevice 120](#_Toc382058268)

[QFileInfo 120](#_Toc382058269)

[QFileSystemWatcher 121](#_Toc382058270)

[QIODevice 121](#_Toc382058271)

[QResource 121](#_Toc382058272)

[QSaveFile 122](#_Toc382058273)

[QTemporaryDir 122](#_Toc382058274)

[QTemporaryFile 122](#_Toc382058275)

[QUrl 122](#_Toc382058276)

[QUrlQuery 123](#_Toc382058277)

[QSocketNotifier 123](#_Toc382058278)

[QFile 124](#_Toc382058279)

[QTextStream 124](#_Toc382058280)

[QInputMethodEvent::Attribute 125](#_Toc382058281)

[QDataStream 125](#_Toc382058282)

[СЕРИАЛИЗАЦИЯ ТИПОВ ДАННЫХ 126](#_Toc382058283)

[QBuffer 126](#_Toc382058284)

[QByteArray 126](#_Toc382058285)

[QBitArray 128](#_Toc382058286)

[КОНТЕЙНЕРНЫЕ КЛАССЫ 129](#_Toc382058287)

[QVector 132](#_Toc382058288)

[QList 132](#_Toc382058289)

[QLinkedList 133](#_Toc382058290)

[QVarLengthArray 133](#_Toc382058291)

[QStack 133](#_Toc382058292)

[QQueue 134](#_Toc382058293)

[QStringList 134](#_Toc382058294)

[QMap 134](#_Toc382058295)

[QHash 134](#_Toc382058296)

[QSet 135](#_Toc382058297)

[QMultiMap 135](#_Toc382058298)

[QMultiHash 135](#_Toc382058299)

[QCache 135](#_Toc382058300)

[QContiguousCache 136](#_Toc382058301)

[QPair 136](#_Toc382058302)

[ПОДДЕРЖКА ПОТОКОВ В КТ 136](#_Toc382058303)

[ИНИЦИИРОВНАИЕ ПОТОКОВ ПРИ ПОМОЩИ QTHREAD 136](#_Toc382058304)

[СИНХРОНИЗАЦИЯ ПОТОКОВ 137](#_Toc382058305)

[РЕЕНТЕРАБЕЛЬНОСТЬ И ПОТОКОБЕЗОПАСНОСТЬ 137](#_Toc382058306)

[ПОТОКИ И QOBJECTS 139](#_Toc382058307)

[ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ 141](#_Toc382058308)

[ПОДДЕРЖКА ПОТОКОВ В МОДУЛЯХ QT 141](#_Toc382058309)

[QAtomicInt 142](#_Toc382058310)

[QAtomicPointer 142](#_Toc382058311)

[QMutex 143](#_Toc382058312)

[QMutexLocker 143](#_Toc382058313)

[QReadWriteLock 143](#_Toc382058314)

[QReadLocker 143](#_Toc382058315)

[QWriteLocker 143](#_Toc382058316)

[QRunnable 143](#_Toc382058317)

[QThreadPool 144](#_Toc382058318)

[QSemaphore 144](#_Toc382058319)

[QThreadStorage 145](#_Toc382058320)

[QWaitCondition 145](#_Toc382058321)

[Классы интернационализации и локализации 146](#_Toc382058322)

[QLocale 146](#_Toc382058323)

[QString 146](#_Toc382058324)

[QStringRef 147](#_Toc382058325)

[QChar 147](#_Toc382058326)

[QLatin1Char 147](#_Toc382058327)

[QLatin1String 147](#_Toc382058328)

[АНИМАЦИИ 148](#_Toc382058329)

[QAbstractAnimation 149](#_Toc382058330)

[QAnimationGroup 149](#_Toc382058331)

[QParallelAnimationGroup 149](#_Toc382058332)

[QPauseAnimation 149](#_Toc382058333)

[QPropertyAnimation 150](#_Toc382058334)

[QSequentialAnimationGroup 150](#_Toc382058335)

[QVariantAnimation 150](#_Toc382058336)

[QEasingCurve 150](#_Toc382058337)

[QTimeLine 151](#_Toc382058338)

[КАРКАС МАШИНЫ СОСТОЯНИЙ 151](#_Toc382058339)

[QKeyEventTransition 153](#_Toc382058340)

[QMouseEventTransition 153](#_Toc382058341)

[QAbstractState 153](#_Toc382058342)

[QAbstractTransition 154](#_Toc382058343)

[QEventTransition 154](#_Toc382058344)

[QFinalState 154](#_Toc382058345)

[QHistoryState 154](#_Toc382058346)

[QSignalTransition 155](#_Toc382058347)

[QState 155](#_Toc382058348)

[QStateMachine 155](#_Toc382058349)

[SignalEvent 156](#_Toc382058350)

[WrappedEvent 156](#_Toc382058351)

[ПЛАГИНЫ В QT 156](#_Toc382058352)

[РАЗВЁРТЫВАНИЕ ПЛАГИНОВ 158](#_Toc382058353)

[ИСПОЛЬЗОВАНИЕ QT.CONF 159](#_Toc382058354)

[КЛАССЫ ПЛАГИНОВ В КТ 160](#_Toc382058355)

[QLibraryInfo 160](#_Toc382058356)

[QIconEnginePlugin 160](#_Toc382058357)

[QImageIOPlugin 160](#_Toc382058358)

[QGenericPlugin 160](#_Toc382058359)

[QGenericPluginFactory 161](#_Toc382058360)

[QStylePlugin 161](#_Toc382058361)

[QLibrary 161](#_Toc382058362)

[<QtPlugin> 161](#_Toc382058363)

[СИСТЕМА СОБЫТИЙ 162](#_Toc382058364)

[QAbstractEventDispatcher 163](#_Toc382058365)

[QBasicTimer 164](#_Toc382058366)

[QChildEvent 164](#_Toc382058367)

[QDynamicPropertyChangeEvent 164](#_Toc382058368)

[QEvent 164](#_Toc382058369)

[QTimerEvent 165](#_Toc382058370)

[QTimer 165](#_Toc382058371)

[ТАЙМЕРЫ 165](#_Toc382058372)

[QGestureEvent 166](#_Toc382058373)

[ЖЕСТЫ В ВИДЖЕТАХ И ГРАФИЧЕСКИХ ПРЕДСТАВЛЕНИЯХ 166](#_Toc382058374)

[QEnterEvent 167](#_Toc382058375)

[QInputEvent 167](#_Toc382058376)

[QMouseEvent 167](#_Toc382058377)

[QHoverEvent 167](#_Toc382058378)

[QFocusEvent 168](#_Toc382058379)

[QPaintEvent 168](#_Toc382058380)

[QMoveEvent 169](#_Toc382058381)

[QExposeEvent 169](#_Toc382058382)

[QResizeEvent 169](#_Toc382058383)

[QCloseEvent 169](#_Toc382058384)

[QIconDragEvent 169](#_Toc382058385)

[QContextMenuEvent 170](#_Toc382058386)

[QInpurMethodEvent 170](#_Toc382058387)

[QTabletEvent 170](#_Toc382058388)

[QDropEvent 170](#_Toc382058389)

[QDragEnterEvent 171](#_Toc382058390)

[QDragMoveEvent 171](#_Toc382058391)

[QDragLeaveEvent 171](#_Toc382058392)

[QHelpEvent 171](#_Toc382058393)

[QStatusTipEvent 171](#_Toc382058394)

[QWhatsThisClickedEvent 172](#_Toc382058395)

[QActionEvent 172](#_Toc382058396)

[QHideEvent 172](#_Toc382058397)

[QShowEvent 172](#_Toc382058398)

[QFileOpenEvent 172](#_Toc382058399)

[QShortcutEvent 173](#_Toc382058400)

[QWindowStateChangeEvent 173](#_Toc382058401)

[QTouchEvent 173](#_Toc382058402)

[QScrollPrepareEvent 173](#_Toc382058403)

[QScrollEvent 173](#_Toc382058404)

[НЕЯВНОЕ РАЗДЕЛЕНИЕ 174](#_Toc382058405)

[ГЛУБОКОЕ И НЕПОЛНОЕ КОПИРОВАНИЕ 174](#_Toc382058406)

[JSON ПОДДЕРЖКА В QT 174](#_Toc382058407)

[QJsonArray 175](#_Toc382058408)

[QJsonDocument 175](#_Toc382058409)

[QJsonObject 175](#_Toc382058410)

[QJsonObject::iterator 175](#_Toc382058411)

[QJsonParseError 175](#_Toc382058412)

[QJsonValue 175](#_Toc382058413)

[АЛГОРИТМЫ QT 176](#_Toc382058414)

[ГЛОБАЛЬНЫЕ ОБЪЯВЛЕНИЯ 176](#_Toc382058415)

[ДРУГИЕ КЛАССЫ ЯДРА 177](#_Toc382058416)

[QProcessEnvironment 177](#_Toc382058417)

[QProcess 177](#_Toc382058418)

[QSettings 178](#_Toc382058419)

[QRegExp 179](#_Toc382058420)

[МОДУЛЬ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ 181](#_Toc382058421)

[СИСТЕМА РИСОВАНИЯ 182](#_Toc382058422)

[ЧТЕНИЕ И ЗАПИСЬ ФАЙЛОВ ИЗОБРАЖЕНИЙ 182](#_Toc382058423)

[СИСТЕМА КООРДИНАТ 183](#_Toc382058424)

[УСТРОЙСТВА РИСОВАНИЯ И БЭКЕНДЫ 184](#_Toc382058425)

[РИСОВАНИЕ И ЗАПОЛНЕНИЕ 185](#_Toc382058426)

[Классы рисования 186](#_Toc382058427)

[QLine 188](#_Toc382058428)

[QLineF 188](#_Toc382058429)

[QMargins 189](#_Toc382058430)

[QPoint 189](#_Toc382058431)

[QPointF 189](#_Toc382058432)

[QRect 189](#_Toc382058433)

[QRectF 189](#_Toc382058434)

[QSize 189](#_Toc382058435)

[QSizeF 190](#_Toc382058436)

[QSvgGenerator 190](#_Toc382058437)

[QSvgRenderer 190](#_Toc382058438)

[QSvgWidget 190](#_Toc382058439)

[QStylePainter 191](#_Toc382058440)

[QColormap 191](#_Toc382058441)

[QBitmap 191](#_Toc382058442)

[QIcon 191](#_Toc382058443)

[QIconEngine 191](#_Toc382058444)

[QImage 192](#_Toc382058445)

[QImageReader 192](#_Toc382058446)

[QImageWriter 193](#_Toc382058447)

[QPixmap 193](#_Toc382058448)

[QGenericMatrix 193](#_Toc382058449)

[QVector2D 193](#_Toc382058450)

[QBrush 193](#_Toc382058451)

[QGradient 193](#_Toc382058452)

[QLinearGradient 194](#_Toc382058453)

[QRadialGradient 194](#_Toc382058454)

[QConicalGradient 194](#_Toc382058455)

[QColor 194](#_Toc382058456)

[QPagedPaintDevice 195](#_Toc382058457)

[QPaintDevice 195](#_Toc382058458)

[QPaintEngine 195](#_Toc382058459)

[QPainter 195](#_Toc382058460)

[QPainterPath 197](#_Toc382058461)

[QPainterPathStroker 198](#_Toc382058462)

[QPdfWriter 198](#_Toc382058463)

[QPen 198](#_Toc382058464)

[QPolygon 198](#_Toc382058465)

[QPolygonF 198](#_Toc382058466)

[QRegion 198](#_Toc382058467)

[QTransform 199](#_Toc382058468)

[QFont 199](#_Toc382058469)

[QFontMetrics 200](#_Toc382058470)

[QFontMetricsF 200](#_Toc382058471)

[QSupportedWritingSystems 200](#_Toc382058472)

[QPainterPath::Element 201](#_Toc382058473)

[QPixmapCache::Key 201](#_Toc382058474)

[QPainter::PixmapFragment 201](#_Toc382058475)

[QTextLayout::FormatRange 201](#_Toc382058476)

[QBackingStore 201](#_Toc382058477)

[QClipboard 202](#_Toc382058478)

[QCursor 202](#_Toc382058479)

[QFontInfo 202](#_Toc382058480)

[QGlyphRun 202](#_Toc382058481)

[QPaintEngineState 203](#_Toc382058482)

[QPalette 203](#_Toc382058483)

[QPixmapCache 203](#_Toc382058484)

[QRasterPaintEngine 204](#_Toc382058485)

[QRawFont 204](#_Toc382058486)

[QStaticText 204](#_Toc382058487)

[QSurface 205](#_Toc382058488)

[QSurfaceFormat 205](#_Toc382058489)

[НЕКОТОРЫЕ ДРУГИЕ КЛАССЫ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ 205](#_Toc382058490)

[класс QDesktopServices обеспечивает методы для доступа к общим службам десктопа. Данная система делает лёгким использование системы справки. 205](#_Toc382058491)

[класс QDoubleValidator обеспечивает выбор диапазона вещественных чисел. 205](#_Toc382058492)

[QDesktopServices 206](#_Toc382058493)

[QDoubleValidator 206](#_Toc382058494)

[QGuiApplication 206](#_Toc382058495)

[QIntValidator 207](#_Toc382058496)

[QKeySequence 207](#_Toc382058497)

[QMovie 208](#_Toc382058498)

[QPlatformSystemTrayIcon 208](#_Toc382058499)

[QStyleHints 208](#_Toc382058500)

[QRegExpValidator 209](#_Toc382058501)

[QScreen 209](#_Toc382058502)

[QDrag 209](#_Toc382058503)

[QTouchDevice 210](#_Toc382058504)

[QValidator 210](#_Toc382058505)

[QWindow 210](#_Toc382058506)

[УПРАВЛЕНИЕ СЕССИЯМИ 210](#_Toc382058507)

[КЛАССЫ МОДЕЛИ/ПРЕДСТАВЛЕНИЯ 211](#_Toc382058508)

[QStandardItem 211](#_Toc382058509)

[QStandardItemModel 212](#_Toc382058510)

[КЛАССЫ ТЕКСТОВОГО ДОКУМЕНТА 212](#_Toc382058511)

[QAbstractTextDocumentLayout::PaintContext 215](#_Toc382058512)

[QAbstractTextDocumentLayout 215](#_Toc382058513)

[QSyntaxHighlighter 215](#_Toc382058514)

[QTextBlock 215](#_Toc382058515)

[QTextBlockFormat 216](#_Toc382058516)

[QTextBlockGroup 216](#_Toc382058517)

[QTextBlockUserDate 216](#_Toc382058518)

[QTextCharFormat 216](#_Toc382058519)

[QTextCursor 217](#_Toc382058520)

[QTextDocument 217](#_Toc382058521)

[QTextDocumentFragment 217](#_Toc382058522)

[QTextFormat 218](#_Toc382058523)

[QTextFragment 218](#_Toc382058524)

[QTextFrame 218](#_Toc382058525)

[QTextFrameFormat 218](#_Toc382058526)

[QTextImageFormat 219](#_Toc382058527)

[QTextInlineObject 219](#_Toc382058528)

[QTextItem 219](#_Toc382058529)

[QTextLayout 219](#_Toc382058530)

[QTextLength 219](#_Toc382058531)

[QTextLine 219](#_Toc382058532)

[QTextList 220](#_Toc382058533)

[QTextListFormat 220](#_Toc382058534)

[QTextObject 220](#_Toc382058535)

[QTextObjectInterface 220](#_Toc382058536)

[QTextOption 221](#_Toc382058537)

[QTextTable 221](#_Toc382058538)

[QTextTableCell 221](#_Toc382058539)

[QTextTableCellFormat 221](#_Toc382058540)

[QTextTableFormat 221](#_Toc382058541)

[QAbstractTextDocumentLayout::Selection 221](#_Toc382058542)

[QTextBlock::iterator 222](#_Toc382058543)

[QPlainTextDocumentLayout 222](#_Toc382058544)

[РИСОВАНИЕ В 3D 222](#_Toc382058545)

[QGenericMatrix 222](#_Toc382058546)

[QMatrix4x4 222](#_Toc382058547)

[QQuaternion 222](#_Toc382058548)

[QVector2D 223](#_Toc382058549)

[QVector3D 223](#_Toc382058550)

[QVector4D 223](#_Toc382058551)

[QOpenGLBuffer 223](#_Toc382058552)

[QOpenGLDebugLogger 223](#_Toc382058553)

[QOpenGLDebugMessage 224](#_Toc382058554)

[QOpenGLFramebufferObjectFormat 225](#_Toc382058555)

[QOpenGLFramebufferObject 225](#_Toc382058556)

[QOpenGLFunctions 225](#_Toc382058557)

[QOpenGLPaintDevice 226](#_Toc382058558)

[QOpenGLShaderProgram 226](#_Toc382058559)

[QOffscreenSurface 227](#_Toc382058560)

[QML 228](#_Toc382058561)

[ОБЗОР 228](#_Toc382058562)

[ОСНОВЫ СИНТАКСИСА 228](#_Toc382058563)

[АТРИБУТЫ ОБЪЕКТОВ QML 229](#_Toc382058564)

[СВЯЗЫВАНИЕ СВОЙСТВ 230](#_Toc382058565)

[СИСТЕМА СИГНАЛОВ И ОБРАБОТЧИКОВ СИГНАЛОВ 231](#_Toc382058566)

[ИНТЕГРИРОВАНИЕ QML И JAVA SCRIPT 231](#_Toc382058567)

[JAVA SCRIPT ВЫРАЖЕНИЕ В QML ДОКУМЕНТАХ 231](#_Toc382058568)

[ОПРЕДЕЛЕНИЕ JAVA SCRIPT РЕСУРСОВ В QML 232](#_Toc382058569)

[JAVA SCRIPT ХОСТ СРЕДА 233](#_Toc382058570)

[ДИНАМИЧЕСКОЕ СОЗДАНИЕ QML ОБЪЕКТОВ ИЗ JAVA SCRIPT 234](#_Toc382058571)

[ИНТЕГРИРОВАНИЕ JAVA SCRIPT В QML 235](#_Toc382058572)

[СИСТЕМА ТИПОВ QML 235](#_Toc382058573)

[БАЗОВЫЕ ТИПЫ 235](#_Toc382058574)

[ОБЪЕКТНЫЕ ТИПЫ QML 236](#_Toc382058575)

[ОПРЕДЕЛЕНИЕ ОБЪЕКТНЫХ ТИПОВ ПРИ ПОМОЩИ QML ДОКУМЕНТОВ 236](#_Toc382058576)

[ОПРЕДЕЛЕНИЕ ТИПОВ QML ИЗ С++ 237](#_Toc382058577)

[QML МОДУЛИ 238](#_Toc382058578)

[ФАЙЛЫ ОПРЕДЕЛЕНИЯ МОДУЛЕЙ QML 239](#_Toc382058579)

[ИДЕНТИФИЦИРУЕМЫЕ МОДУЛИ 240](#_Toc382058580)

[СОЗДАНИЕ С++ ПЛАГИНОВ ДЛЯ QML 240](#_Toc382058581)

[QML ДОКУМЕНТЫ 240](#_Toc382058582)

[СТРУКТУРА QML ДОКУМЕНТА 241](#_Toc382058583)

[РАЗВЁРТЫВАНИЕ QML ДОКУМЕНТОВ 241](#_Toc382058584)

[ЗАГРУЗКА РЕСУРСОВ И СЕТЕВАЯ ПРОЗРАЧНОСТЬ 242](#_Toc382058585)

[РАЗРЕШЕНИЕ ОБЛАСТИ ВИДИМОСТИ И ИМЁН 242](#_Toc382058586)

[СОГЛАШЕНИЯ КОДИРОВАНИЯ QML 243](#_Toc382058587)

[СЛОВАРЬ ТЕРМИНОВ QML 244](#_Toc382058588)

[ВИЗУАЛЬНЫЕ ЭЛЕМЕНТЫ В QML 245](#_Toc382058589)

[ПОДДЕРЖИВАЕМЫЕ ТИПЫ ВВОДА ПОЛЬЗОВАТЕЛЯ 245](#_Toc382058590)

[АНИМАЦИИ В QML 245](#_Toc382058591)

[ОТОБРАЖЕНИЕ ТЕКСТА В QML 246](#_Toc382058592)

[ПОЗИЦИОНИРОВАНИЕ И МАКЕТЫ В QML 246](#_Toc382058593)

[ПОДДЕРЖКА СТИЛЕЙ И ТЕМ 246](#_Toc382058594)

[ОБЗОР ЭЛЕМЕНТОВ УПРАВЛЕНИЯ QT QUICK 247](#_Toc382058595)

[ИСПОЛЬЗОВАНИЕ СИСТЕМЫ ЧАСТИЦ QML 247](#_Toc382058596)

[ГРАФИЧЕСКИЕ ЭФФЕКТЫ 247](#_Toc382058597)

[РЕСУРСЫ РАЗРАБОТКИ QML ПРИЛОЖЕНИЙ 248](#_Toc382058598)

[ВОПРОСЫ ПРОИЗВОДИТЕЛЬНОСТИ И ПРЕДЛОЖЕНИЯ 248](#_Toc382058599)

[ИНТЕРНАЦИОНАЛИЗАЦИЯ И ЛОКАЛИЗАЦИЯ ПРИ ПОМОЩИ QT QUICK 253](#_Toc382058600)

[QT QML ТИПЫ 254](#_Toc382058601)

[Component 254](#_Toc382058602)

[QtObject 254](#_Toc382058603)

[Binding 255](#_Toc382058604)

[Connections 255](#_Toc382058605)

[Instantiator 255](#_Toc382058606)

[Timer 255](#_Toc382058607)

[Qt объект 255](#_Toc382058608)

[Qt Quick 256](#_Toc382058609)

[Типы окон и экранов 256](#_Toc382058610)

[Screen 256](#_Toc382058611)

[Window 256](#_Toc382058612)

[CloseEvent 256](#_Toc382058613)

[Типы моделей данных 257](#_Toc382058614)

[XmlRole 257](#_Toc382058615)

[XmlListModel 257](#_Toc382058616)

[FolderListModel 257](#_Toc382058617)

[SQLite 257](#_Toc382058618)

[Модуль частиц 257](#_Toc382058619)

[Модуль диалогов 258](#_Toc382058620)

[ColorDialog 258](#_Toc382058621)

[FileDialog 258](#_Toc382058622)

[Модуль элементов управления 258](#_Toc382058623)

[ApplicationWindow 258](#_Toc382058624)

[Некоторые типы элементов управления 258](#_Toc382058625)

[SplitView 259](#_Toc382058626)

[StackView 259](#_Toc382058627)

[Некоторые типы элементов управления 259](#_Toc382058628)

[StackView 260](#_Toc382058629)

[Типы для обработки стилей 261](#_Toc382058630)

[Модуль макетов 261](#_Toc382058631)

[Визуальные типы 262](#_Toc382058632)

[Item 262](#_Toc382058633)

[Rectangle, Image 262](#_Toc382058634)

[AnimatedImage 262](#_Toc382058635)

[AnimatedSprite 262](#_Toc382058636)

[Другие визуальные типы 263](#_Toc382058637)

[Repeater 263](#_Toc382058638)

[Loader 263](#_Toc382058639)

[Визуальные типы для трансформаций 263](#_Toc382058640)

[Модуль ввода пользователя 264](#_Toc382058641)

[Flickable 264](#_Toc382058642)

[Типы ввода-вывода 264](#_Toc382058643)

[Типы расположения элементов 264](#_Toc382058644)

[Состояния, переходы и анимации 265](#_Toc382058645)

[Типы моделей и представлений, а также хранилищ данных и доступа 266](#_Toc382058646)

[DelegateModel 266](#_Toc382058647)

[DelegateModelGroup 267](#_Toc382058648)

[ListModel 267](#_Toc382058649)

[List elements 267](#_Toc382058650)

[ObjectModel 267](#_Toc382058651)

[Графические эффекты. 267](#_Toc382058652)

[Некоторые удобные типы. 268](#_Toc382058653)

[Типы для канвы (наподобие HTML5) 268](#_Toc382058654)

[Модуль сенсоров 268](#_Toc382058655)

[QT QUICK 268](#_Toc382058656)

[ВИЗУАЛЬНАЯ КАНВА QT QUICK 269](#_Toc382058657)

[ВИЗУАЛЬНЫЕ КООРДИНАТЫ В QT QUICK 269](#_Toc382058658)

[ВИЗУАЛЬНЫЕ РОДИТЕЛИ В QT QUICK 269](#_Toc382058659)

[ТИПЫ ПРЕОБРАЗОВАНИЙ QT QUICK 270](#_Toc382058660)

[ВВОД ПОЛЬЗОВАТЕЛЯ В QT QUICK 270](#_Toc382058661)

[СОБЫТИЯ МЫШИ 271](#_Toc382058662)

[ФОКУС КЛАВИАТУРЫ В QT QUICK 271](#_Toc382058663)

[ОБРАБОТКА ВВОДА ТЕКСТА В QT QUICK И ВАЛИДАТОРЫ 272](#_Toc382058664)

[ПОЗИЦИОНИРОВАНИЕ 272](#_Toc382058665)

[ПОЗИЦИОНИРОВАНИЕ ПРИ ПОМОЩИ ЯКОРЕЙ 273](#_Toc382058666)

[ПОЗИЦИОНЕРЫ ЭЛЕМЕНТОВ 274](#_Toc382058667)

[СОСТОЯНИЯ, ПЕРЕХОДЫ И АНИМАЦИИ 274](#_Toc382058668)

[СОСТОЯНИЯ QT QUICK 275](#_Toc382058669)

[ТИПЫ АНИМАЦИЙ И ПЕРЕХОДОВ 276](#_Toc382058670)

[ИСПОЛЬЗОВАНИЕ QT QUICK ПОВЕДЕНИЙ ВМЕСТЕ С СОСТОЯНИЯМИ 278](#_Toc382058671)

[СПРАЙТОВЫЕ АНИМАЦИИ 278](#_Toc382058672)

[МОДЕЛИ, ПРЕДСТАВЛЕНИЯ И СОХРЕНЕНИЕ ДАННЫХ В QT QUICK 279](#_Toc382058673)

[МОДЕЛИ И ПРЕДСТАВЛЕНИЯ В QT QUICK 279](#_Toc382058674)

[ИСПОЛЬЗОВАНИЕ МОДЕЛЕЙ С++ С QT QUICK ПРЕДСТАВЛЕНИЯМИ 280](#_Toc382058675)

[ГРАФИЧЕСКИЕ ЭФФЕКТЫ 280](#_Toc382058676)

[УДОБНЫЕ ТИПЫ В QT QUICK 281](#_Toc382058677)

[ГРАФ СИНОВ В QT QUICK 282](#_Toc382058678)

[КЛАССЫ ГРАФА СИНОВ 283](#_Toc382058679)

[QSGNode 284](#_Toc382058680)

[QSGGeometryNode 284](#_Toc382058681)

[QSGClipNode 284](#_Toc382058682)

[QSGTransformNode 284](#_Toc382058683)

[QSGOpacityNode 285](#_Toc382058684)

[QSGMaterialShader 285](#_Toc382058685)

[QSGMaterialType 285](#_Toc382058686)

[QSGMaterial 286](#_Toc382058687)

[QSGFlatColorMaterial 286](#_Toc382058688)

[QSGSimpleMaterialShader 286](#_Toc382058689)

[QSGSimpleMaterial 286](#_Toc382058690)

[QSGOpaqueTextureMaterial 286](#_Toc382058691)

[QSGTextureMaterial 287](#_Toc382058692)

[QSGVertexColorMaterial 287](#_Toc382058693)

[QSGSimpleRectNode 287](#_Toc382058694)

[QSGSimpleTextureNode 287](#_Toc382058695)

[QSGGeometry::Attribute 287](#_Toc382058696)

[QSGGeometry::AttributeSet 287](#_Toc382058697)

[QSGGeometry::ColoredPoint2D 288](#_Toc382058698)

[QSGGeometry::Point2D 288](#_Toc382058699)

[QSGBasicGeometryNode 288](#_Toc382058700)

[QSGDynamicTexture 288](#_Toc382058701)

[QSGGeometry 288](#_Toc382058702)

[QSGGeometry::TexturedPoint2D 288](#_Toc382058703)

[*Программный интерфейс КМЛ для С++* 288](#_Toc382058704)

[ВАЖНЫЕ С++ КЛАССЫ, КОТОРЫЕ ОБЕСПЕЧИВАЕТ QML МОДУЛЬ 288](#_Toc382058705)

[ИНТЕГРИРОВАНИЕ QML И С++ 289](#_Toc382058706)

[ПОКАЗ АТРИБУТОВ С++ ТИПАМ QML 290](#_Toc382058707)

[НАПИСАНИЕ РАСШИРЕНИЙ QML ПРИ ПОМОЩИ QT 291](#_Toc382058708)

[СОЗДАНИЕ НОВОГО ТИПА QML 291](#_Toc382058709)

[СОЕДИНЕНИЕ С С++ МЕТОДАМИ И СИГНАЛАМИ ИЗ QML 291](#_Toc382058710)

[ДОБАВЛЕНИЕ СВЯЗЫВАНИЯ СВОЙСТВ 291](#_Toc382058711)

[ИСПОЛЬЗОВАНИЕ ТИПОВ ЧАСТНЫХ СВОЙСТВ 291](#_Toc382058712)

[ИСПОЛЬЗОВАНИЕ СВОЙСТВ ТИПА СПИСКОВ 291](#_Toc382058713)

[НАПИСАНИЕ ПЛАГИНА РАСШИРЕНИЯ QML 291](#_Toc382058714)

[ОПРЕДЕЛЕНИЕ ТИПОВ QML ИЗ С++ 292](#_Toc382058715)

[ВСТРОЕННЫЕ С++ ОБЪЕКТЫ В QML ПРИ ПОМОЩИ СВОЙСТВ КОНТЕКСТА 294](#_Toc382058716)

[ИНТЕГРИРОВАНИЕ С QML ОБЪЕКТАМИ ИЗ С++ 294](#_Toc382058717)

[ПРЕОБАРЗОВАНИЕ ДАННЫХ МЕЖДУ QML И С++ 295](#_Toc382058718)

[С++ КЛАССЫ ДЛЯ QML 296](#_Toc382058719)

[QJSEngine 296](#_Toc382058720)

[QJSValue 296](#_Toc382058721)

[QJSValueIterator 296](#_Toc382058722)

[QQmlAbstractUrlInterceptor 296](#_Toc382058723)

[QQmlApplicationEngine 297](#_Toc382058724)

[QQmlComponent 297](#_Toc382058725)

[QQmlContext 297](#_Toc382058726)

[QQmlEngine 297](#_Toc382058727)

[QQmlError 297](#_Toc382058728)

[QQmlExpression 298](#_Toc382058729)

[QQmlExtensionPlugin 298](#_Toc382058730)

[QQmlImageProviderBase 298](#_Toc382058731)

[QQmlIncubationController 298](#_Toc382058732)

[QQmlListProperty 298](#_Toc382058733)

[QQmlIncubator 298](#_Toc382058734)

[QQmlListReference 299](#_Toc382058735)

[QQmlNetworkAccessManagerFactory 299](#_Toc382058736)

[QQmlParserStatus 299](#_Toc382058737)

[QQmlProperty 299](#_Toc382058738)

[QQmlPropertyMap 299](#_Toc382058739)

[QQmlPropertyValueSource 300](#_Toc382058740)

[QQmlScriptString 300](#_Toc382058741)

[ТОЧКИ РАСШИРЕНИЯ С++, ОБЕСПЕЧИВАЕМЫЕ QT QUICK 300](#_Toc382058742)

[QQuickImageProvider 300](#_Toc382058743)

[QQuickItem 301](#_Toc382058744)

[QQuickWindow 301](#_Toc382058745)

[QQuickTextDocument 301](#_Toc382058746)

[QQuickTextureFactory 301](#_Toc382058747)

[QQuickView 302](#_Toc382058748)

[QQuickItem::ItemChangeData 302](#_Toc382058749)

[ВИДЖЕТЫ 303](#_Toc382058750)

[ИНТЕРФЕЙСЫ ПОЛЬЗОВАТЕЛЯ 303](#_Toc382058751)

[ВИДЖЕТЫ ОКОН И ДИАЛОГОВ 306](#_Toc382058752)

[РУКОВОДСТВО ПО ВИДЖЕТАМ 306](#_Toc382058753)

[ГЛАВНОЕ ОКНО ПРИЛОЖЕНИЯ 307](#_Toc382058754)

[QAction 307](#_Toc382058755)

[QActionGroup 308](#_Toc382058756)

[QWidgetAction 308](#_Toc382058757)

[QDockWidget 308](#_Toc382058758)

[QMainWindow 308](#_Toc382058759)

[QMdiArea 308](#_Toc382058760)

[QMdiSubWindow 309](#_Toc382058761)

[QMenu 309](#_Toc382058762)

[QSizeGrip 309](#_Toc382058763)

[QStatusBar 309](#_Toc382058764)

[QToolBar 309](#_Toc382058765)

[ДИАЛОГОВЫЕ ОКНА 310](#_Toc382058766)

[QColorDialog 310](#_Toc382058767)

[QFileDialog 310](#_Toc382058768)

[QFontDialog 310](#_Toc382058769)

[QInputDialog 310](#_Toc382058770)

[QMessageBox 310](#_Toc382058771)

[QProgressDialog 311](#_Toc382058772)

[QDialog 311](#_Toc382058773)

[QDialogButtonBox 312](#_Toc382058774)

[ДЕСКТОП ИНТЕГРАЦИЯ 312](#_Toc382058775)

[QDesktopServices 313](#_Toc382058776)

[QDesktopWidget 313](#_Toc382058777)

[QSystemTrayIcon 313](#_Toc382058778)

[ФОКУС ВВОДА КЛАВИАТУРЫ В ВИДЖЕТАХ 314](#_Toc382058779)

[СТИЛИ И СТИЛИЗОВАННЫЕ ВИДЖЕТЫ 314](#_Toc382058780)

[QCursor 319](#_Toc382058781)

[QPalette 319](#_Toc382058782)

[QCommonStyle 319](#_Toc382058783)

[QStyleFactory 319](#_Toc382058784)

[QStyleOption 320](#_Toc382058785)

[QStyleHintReturn 320](#_Toc382058786)

[QStyleHintReturnMask 320](#_Toc382058787)

[QStyleHintReturnVariant 320](#_Toc382058788)

[QStylePainter 320](#_Toc382058789)

[QProxyStyle 320](#_Toc382058790)

[ДРУГИЕ КЛАССЫ СТИЛЕЙ 321](#_Toc382058791)

[Некоторые другие классы виджетов 321](#_Toc382058792)

[QFontDatabase 321](#_Toc382058793)

[QGraphicsAnchorLayout 321](#_Toc382058794)

[QGraphicsAnchor 321](#_Toc382058795)

[ШИТЫ СТИЛЕЙ 321](#_Toc382058796)

[СИНТАКСИС ШИТОВ СТИЛЕЙ 322](#_Toc382058797)

[ИНТЕГРАЦИЯ ШИТОВ СТИЛЕЙ С QT ДИЗАЙНЕРОМ 323](#_Toc382058798)

[НАСТРОЙКА QT ВИДЖЕТОВ С ИСПОЛЬЗОВАНИЕМ ШИТОВ СТИЛЕЙ 323](#_Toc382058799)

[СПРАВОЧНИК ПО ШИТАМ СТИЛЕЙ 324](#_Toc382058800)

[ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ QT ШИТОВ СТИЛЕЙ 325](#_Toc382058801)

[УПРАВЛЕНИЕ МАКЕТАМИ 325](#_Toc382058802)

[QGroupBox 327](#_Toc382058803)

[QButtonGroup 327](#_Toc382058804)

[ПРОГРАММИРОВАНИЕ МОДЕЛИ/ПРЕДСТАВЛЕНИЯ 327](#_Toc382058805)

[РУКОВОДСТВО ПО МОДЕЛИ/ПРЕДСТАВЛЕНИЮ 331](#_Toc382058806)

[QAbstractItemModel 333](#_Toc382058807)

[QAbstractListModel 333](#_Toc382058808)

[QAbstractTableModel 333](#_Toc382058809)

[QModelIndex 333](#_Toc382058810)

[QPersistentModelIndex 333](#_Toc382058811)

[QAbstractProxyModel 334](#_Toc382058812)

[QIdentityProxyModel 334](#_Toc382058813)

[QItemSelection 334](#_Toc382058814)

[QItemSelectionModel 334](#_Toc382058815)

[QItemSelectionRange 334](#_Toc382058816)

[QSortFilterProxyModel 334](#_Toc382058817)

[QStringListModel 335](#_Toc382058818)

[QStandardItem 335](#_Toc382058819)

[QStandardItemModel 335](#_Toc382058820)

[QFileSystemModel 335](#_Toc382058821)

[QAbstractItemDelegate 336](#_Toc382058822)

[QAbstractItemView 336](#_Toc382058823)

[QColumnView 336](#_Toc382058824)

[QDataWidgetMapper 336](#_Toc382058825)

[QHeaderView 337](#_Toc382058826)

[QItemDelegate 337](#_Toc382058827)

[QItemEditorFactory 337](#_Toc382058828)

[QItemEditorCreatorBase 337](#_Toc382058829)

[QItemEditorCreator 338](#_Toc382058830)

[QStandardItemEditorCreator 338](#_Toc382058831)

[QListView 338](#_Toc382058832)

[QListWidgetItem 338](#_Toc382058833)

[QListWidget 338](#_Toc382058834)

[QStyledItemDelegate 338](#_Toc382058835)

[QTableView 339](#_Toc382058836)

[QTableWidgetSelectionRange 339](#_Toc382058837)

[QTableWidgetItem 339](#_Toc382058838)

[QTableWidget 339](#_Toc382058839)

[QTreeView 339](#_Toc382058840)

[QTreeWidgetItem 339](#_Toc382058841)

[QTreeWidget 339](#_Toc382058842)

[ГРАФИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ 340](#_Toc382058843)

[QGraphicsEffect 343](#_Toc382058844)

[QGraphicsGridLayout 343](#_Toc382058845)

[QGraphicsObject 344](#_Toc382058846)

[QAbstractGraphicsShapeItem 344](#_Toc382058847)

[QGraphicsRotation 345](#_Toc382058848)

[QGraphicsScale 345](#_Toc382058849)

[КЛАССЫ РАЗНЫХ ТИПОВ ЭЛЕМЕНТОВ СИНА 345](#_Toc382058850)

[QGraphicsItemGroup 345](#_Toc382058851)

[QGraphicsLayout 345](#_Toc382058852)

[QGraphicsLayoutItem 345](#_Toc382058853)

[QGraphicsProxyWidget 345](#_Toc382058854)

[QGraphicsScene 345](#_Toc382058855)

[QGraphicsSceneEvent 346](#_Toc382058856)

[КЛАССЫ СОБЫТИЙ СИН ГРАФА 346](#_Toc382058857)

[QGraphicsTransform 346](#_Toc382058858)

[QGraphicsView 347](#_Toc382058859)

[QGraphicsWidget 347](#_Toc382058860)

[QStyleOptionGraphicsItem 347](#_Toc382058861)

[РУКОВОДСТВО ПО ВИДЖЕТАМ 348](#_Toc382058862)

[БАЗОВЫЕ ВИДЖЕТЫ 348](#_Toc382058863)

[КЛАССЫ ВИДЖЕТОВ 350](#_Toc382058864)

[QCommandLinkButton 350](#_Toc382058865)

[Виджеты редактирования дат и времён 350](#_Toc382058866)

[QDial 350](#_Toc382058867)

[QFocusFrame 350](#_Toc382058868)

[QFontComboBox 350](#_Toc382058869)

[QLabel 350](#_Toc382058870)

[QLCDNumber 350](#_Toc382058871)

[QLineEdit 351](#_Toc382058872)

[QMenu 351](#_Toc382058873)

[QWidgetAction 351](#_Toc382058874)

[Некоторые другие классы виджетов 351](#_Toc382058875)

[QScrollArea 351](#_Toc382058876)

[QScrollBar 351](#_Toc382058877)

[QSizeGrip 352](#_Toc382058878)

[QSlider 352](#_Toc382058879)

[QDoubleSpinBox 353](#_Toc382058880)

[QSpinBox 353](#_Toc382058881)

[QTabBar 353](#_Toc382058882)

[QTabWidget 353](#_Toc382058883)

[QToolBox 354](#_Toc382058884)

[QToolButton 354](#_Toc382058885)

[КЛАССЫ СЛОЖНЫХ ВИДЖЕТОВ 354](#_Toc382058886)

[QColumnView 354](#_Toc382058887)

[QDataWidgetMapper 354](#_Toc382058888)

[QListView 355](#_Toc382058889)

[QTableView 355](#_Toc382058890)

[QTreeView 355](#_Toc382058891)

[QDesktopWidget 355](#_Toc382058892)

[QUndoView 355](#_Toc382058893)

[QCalendarWidget 355](#_Toc382058894)

[АБСТРАКТНЫЕ КЛАССЫ ВИДЖЕТОВ 356](#_Toc382058895)

[QAbstractButton 356](#_Toc382058896)

[QAbstractScrollArea 356](#_Toc382058897)

[QAbstractSlider 357](#_Toc382058898)

[QAbstractSpinBox 357](#_Toc382058899)

[КЛАССЫ МАКЕТОВ ВИДЖЕТОВ 357](#_Toc382058900)

[QFrame 357](#_Toc382058901)

[QButtonGroup 357](#_Toc382058902)

[QGroupBox 357](#_Toc382058903)

[QSplitterHandler 358](#_Toc382058904)

[QSplitter 358](#_Toc382058905)

[QStackedWidget 358](#_Toc382058906)

[QSizePolicy 358](#_Toc382058907)

[QTabWidget 359](#_Toc382058908)

[ДРУГИЕ КЛАССЫ ВИДЖЕТОВ 359](#_Toc382058909)

[QTextEdit::ExtraSelection 359](#_Toc382058910)

[QCompleter 359](#_Toc382058911)

[QErrorMessage 359](#_Toc382058912)

[QFileIconProvider 359](#_Toc382058913)

[QPlainTextEdit 360](#_Toc382058914)

[QRubberBand 360](#_Toc382058915)

[QScroller 360](#_Toc382058916)

[QSpacerItem 360](#_Toc382058917)

[QSplashScreen 360](#_Toc382058918)

[QTileRules 361](#_Toc382058919)

[QToolTip 361](#_Toc382058920)

[QUndoCommand 361](#_Toc382058921)

[QUndoGroup 361](#_Toc382058922)

[QWhatsThis 362](#_Toc382058923)

[QWizard 362](#_Toc382058924)

[QWizardPage 363](#_Toc382058925)

[QT WEBKIT 363](#_Toc382058926)

[WebView 363](#_Toc382058927)

[QT WEBKIT ВИДЖЕТЫ 363](#_Toc382058928)

[QT WEBKIT МОСТ 365](#_Toc382058929)

[QWebView 367](#_Toc382058930)

[QWebPage 368](#_Toc382058931)

[QWebFrame 368](#_Toc382058932)

[QWebElement 369](#_Toc382058933)

[QWebSettings 369](#_Toc382058934)

[QT SQL 369](#_Toc382058935)

[SQL ПРОРАММИРОВАНИЕ 369](#_Toc382058936)

[СОЕДИНЕНИЕ С БАЗАМИ ДАННЫХ 370](#_Toc382058937)

[SQL ДРАЙВЕРЫ БАЗ ДАННЫХ 370](#_Toc382058938)

[ВЫПОЛНЕНИЕ SQL УТВЕРЖДЕНИЙ 371](#_Toc382058939)

[ТИПЫ ДАННЫХ ДЛЯ ПОДДЕРЖИВАЕМЫХ QT СИСТЕМ БАЗ ДАННЫХ 372](#_Toc382058940)

[ИСПОЛЬЗОВАНИЕ КЛАССОВ SQL МОДЕЛИ 372](#_Toc382058941)

[ПРЕДСТАВЛЕНИЕ ДАННЫХ В ТАБЛИЧНОМ ПРЕДСТАВЛЕНИИ 372](#_Toc382058942)

[СОЗДАНИЕ ФОРМ, ОСВЕДОМЛЁННЫХ О ДАННЫХ 373](#_Toc382058943)

[КЛАССЫ SQL 373](#_Toc382058944)

[QSql 373](#_Toc382058945)

[QSqlDatabase 373](#_Toc382058946)

[QSqlDriver 374](#_Toc382058947)

[QSqlDriverCreator 374](#_Toc382058948)

[QSqlDriverCreatorBase 374](#_Toc382058949)

[QSqlDriverPlugin 374](#_Toc382058950)

[QSqlError 374](#_Toc382058951)

[QSqlField 374](#_Toc382058952)

[QSqlIndex 375](#_Toc382058953)

[QSqlQuery 375](#_Toc382058954)

[QSqlQueryModel 375](#_Toc382058955)

[QSqlRecord 375](#_Toc382058956)

[QSqlRelation 375](#_Toc382058957)

[QSqlRelationalDelegate 376](#_Toc382058958)

[QSqlRelationalTableModel 376](#_Toc382058959)

[QSqlResult 376](#_Toc382058960)

[QSqlTableModel 376](#_Toc382058961)

[QT ТЕСТ 376](#_Toc382058962)

[QSerialPort обеспечивает функции для доступа к последовательным портам. 385](#_Toc382058963)

*Комментарии по поводу дальнейшей работы с конспектом. В целом сайт qt документации очень информативен и замечательно структурирован. Поэтому не нужно дополнительно структурировать данный конспект. Можно в будущем создать лишь высокоуровневый граф. Сейчас моя задача – это понять данный конспект и определить, что понятно не до конца. Далее следует изучить дополнительные источники, которые нужны для лучшего понимания тех или иных тем данной документации.*

*Обозначения*

*Красный цвет – то, что непонятно.*

*Зелёный цвет – то, что важно и следует запомнить.*

*Зелёный фон – мои комментарии, которые возникли на этапе изучения конспекта.*

*Жёлтый шрифт – рекомендации авторов документации.*

# QT – ВВЕДЕНИЕ

Сайт <http://qt-project.org/doc/qt-5.1/qtdoc/index.html>

Qt – это полная библиотека с инструментами, разработанными, чтобы рационализировать создание приложений и графических интерфейсов пользователей для десктопов, встроенных и мобильных платформ.

Qt Framework – это интуитивные API для С++ и для программирования (подобного на CSS/JavaScript) средствами Qt Quick для быстрого создания GUI.

Qt Creator IDE – это мощная кросс-платформенная интегрированная среда разработки, включающая инструменты GUI дизайна и средства отладки на имитаторе устройства, инструменты и наборы компиляторов, поддержку интернационализации и многое другое.

<http://qt-project.org/doc/qt-5.1/qtdoc/gettingstarted.html> - страница, где находятся ссылки на все ресурсы касательно основ qt.

## СРЕДСТВА РАЗРАБОТКИ

### QT CREATOR

<http://qt-project.org/doc/qt-5.1/qtdoc/topics-app-development.html>

Qt Creator обеспечивает вас инструментами, которые позволяют вам пройти через полный цикл разработки приложения, начиная от создания проекта и заканчивая развёртыванием приложения на целевой платформе. Решения некоторых задач в данной среде автоматизированы. Также она предоставляет инструменты для синтаксического анализа кода.

Следующие инструменты qt интегрированы в Qt Creator:

* Qt Designer – для разработки и построения графического интерфейса пользователя из виджетов Qt. Получить доступ к дизайнеру из создателя можно, перейдя в Design mode.
* qmake – для построения приложений для разных целевых платформ. Вы также можете использовать и другие средства построения, такие как CMake или Autotools. Вы можете запустить их из создателя, чтобы построить приложение. Вы указываете систему построения в Project mode.
* Qt Linguist – для локализации приложений. Он содержит инструменты для сотрудников, типично вовлечённых в разработку локализующихся приложений: разработчиков, переводчиков и менеджеров выпуска. Его можно запустить из меню создателя Tools.
* Qt Assistant – для просмотра документации Qt. Документацию также можно просматривать в создателе. Документация автоматически отображается в Help mode.
* makeqpf – для создания pre-rendered шрифтов для встроенных устройств,
* Meta-Object Compiler (moc) – для генерации метаобъектной информации о подклассах QObject.
* User Interface Compiler (uic) – для генерации кода С++ из файлов интерфейса пользователя,
* Resource Compiler (rcc) – встраивает ресурсы в приложения Qt во время процесса построения.

<http://qt-project.org/doc/qt-5.1/qtdesigner/qtdesigner-manual.html>

#### QT DESIGNER

Qt Designer – это инструмент Qt для построения графических интерфейсов пользователей с использованием Qt Widgets. Вы можете придумать и настроить ваши окна или диалоги в (WYSIWYG) манере (what-you-see-is-what-you-get) и протестировать их с использованием разных стилей и разрешений.

Виджеты и формы, созданные при помощи дизайнера, плавно интегрируются в код, с использованием механизма сигналов и слотов, так что вы можете легко присвоить поведение графическим элементам. Все свойства, установленные в дизайнере, могут быть изменены динамически внутри кода. Более того, особенности наподобие widget promotion или custom plugins позволяют вам использовать ваши собственные компоненты вместе с дизайнером.

<http://qt-project.org/doc/qt-5.1/qtdesigner/designer-editing-mode.html>

##### DESIGNER MODES

Qt Designer предоставляет четыре моды для редактирования:

* мода редактирования виджетов,
* мода редактирования сигналов и слотов,
* мода редактирования партнёров
* мода редактирования порядка следования виджетов при нажатии табуляции.

<http://qt-project.org/doc/qt-5.1/qtdesigner/designer-widget-mode.html>

###### МОДА РЕДАКТИРОВАНИЯ

В данной моде объекты можно перетащить из главного окна виджетов на форму, которые затем можно редактировать:

* изменять размер,
* перетаскивать внутри формы
* перетаскивать между формами
* изменять интерактивно свойства объектов

UI файлы, сохранённые дизайнером, содержат информацию об объектах, использованных пользователем, и также о любых деталях соединения сигналов и слотов между ними.

*Редактор свойств* всегда содержит набор свойств текущего выбранного объекта на форме. Модифицированные свойства отображаются при помощи жирного шрифта. *Также есть возможность создания динамических свойств.*

*Инспектор объектов* отображает иерархический список всех объектов на форме, которая сейчас редактируется. Выбрав соответствующий пункт на инспекторе объектов, можно выбрать соответствующий виджет.

|  |  |
| --- | --- |
| **Клавиша** | **Предназначение** |
| Перетаскивание виджетов с ctrl | виджет будет передвигаться пиксел за пикселем, а иначе он будет передвигать с шагом сетки, которая отображается на форме. |
| Выбирать виджеты, удерживая Shift | Чтобы выбрать несколько виджетов |
| Ctrl + Shift + передвигать виджеты | Чтобы передвигать виджеты внутри макетов |
| Сtrl + перетаскивание виджета | Чтобы клонировать виджет |
| Сtrl + F | Чтобы получить местоположение объекта в инспекторе объектов |
| F3 | Включение моды редактирования виджетов |
| Ctrl + R | Предварительный просмотр формы |
| F4 | Включение моды редактирования сигналов и слотов. |
| нажатие на один объект и перетаскивание курсора с него на другой объект. Затем следует выбрать сигнал от одного объекта и слот от другого объекта | Соединение сигнала со слотом. |
| первый виджет кликается мышкой. Затем кликаются все остальные виджеты в том порядке, в котором вы желаете, чтобы они шли. Если вы сделали что-то не так, то следует произвести двойной щелчок мыши по форме. Если вы желаете изменить порядок следования виджетов в некоторой части цепочки следования виджетов, то вам необходимо, удерживая клавишу ctrl, нажать на соответствующий виджет. | Установка порядка следования виджетов на форме |
| Ctrl + 1 | Помещает выделенные объекты в горизонтальный макет |
| Ctrl + 2 | Помещает выделенные объекты в вертикальный макет |
| Ctrl + 5 | Помещает выделенные объекты в сеточный макет |
| Ctrl + 6 | Помещает выделенные объекты в макет формы |
| Ctrl + 3 | Создаёт горизонтальный разделитель и помещает выделенные объекты внутри него |
| Ctrl + 4 | Создаёт вертикальный разделитель и помещает выделенные объекты внутри него |
| Ctrl + j | Приводит в порядок размер макета |

###### МОДА РЕДАКТИРОВАНИЯ СИГНАЛОВ И СЛОТОВ

<http://qt-project.org/doc/qt-5.1/qtdesigner/designer-connection-mode.html>

При использовании моды редактирования сигналов и слотов, можно соединить объекты на форме. Все данные соединения сохраняются при сохранении формы. Чтобы соединить объект с формой, просто отпустите курсор мыши над формой. Соединения можно легко удалять, а также редактировать.

<http://qt-project.org/doc/qt-5.1/qtdesigner/designer-buddy-mode.html>

###### МОДА ОПРЕДЕЛЕНИЯ ПАРТНЁРОВ

Виджет партнёр допускает фокус ввода от имени метки, когда пользователь нажимает комбинацию горячих клавиш данной метки.

Одна из наиболее полезных базовых особенностей в Qt – это поддержка виджетов партнёров. В данном случае все соединения производятся также как и в моде редактирования сигналов и слотов, но только соединения в данной моде должны начинаться с метки.

Протестировать партнёров можно с использованием предварительного просмотра формы.

Для каждой метки может быть выбран только один виджет партнёр. Чтобы изменить партнёра, следует для начала удалить текущее соединение с партнёром.

Предшествующий символ & делает следующую букву мнемоническим символом меню.

###### МОДА ПОРЯДКА ТАБУЛЯЦИИ

<http://qt-project.org/doc/qt-5.1/qtdesigner/designer-tab-order.html>

Многие пользователи ожидают, что смогут проводить навигацию между виджетами только с использованием клавиатуры. Qt даёт возможность пользователям проводить навигацию между виджетами при помощи клавиш tab и shift+tab. Порядок по умолчанию совпадает с тем порядком, в котором виджеты конструировались.

**ВИДЖЕТЫ КОНТЕЙНЕРЫ**

<http://qt-project.org/doc/qt-5.1/qtdesigner/designer-using-containers.html>

виджеты контейнеры обеспечивают более высокоуровневый контроль над группами объектов на форме.

Qt Designer обеспечивает визуальную обратную связь, чтобы помочь вам разместить объекты внутри ваших контейнеров. Когда вы перетаскиваете виджет из вкладыша на форму, каждый контейнер будет выделяться, когда курсор будет располагаться над ним.

QFrame используется для заключения и группировки виджетов, также как и для обеспечения декорирования. Они используются как фундамент для более сложных контейнеров, но они также могут быть использованы как заполнители на форме.

QGroupBox обычно используется для группирования вместе набора флажков и переключателей.

Stacked Widgets. Это наборы виджетов, в которых только самый верхний слой виден. Контроль за слоем, который виден, обычно осуществляется при помощи другого виджета.

Tab Widgets позволяют разработчику дробить содержание виджета в разных, помеченных метками секциях, только одна из которых отображена в данное время.

ToolBox Widgets обеспечивают серию страниц или отсеков в инструментах. Они обрабатываются похожим образом на предыдущий виджет. Есть иллюстрация.

Dock Widgets – это плавающие панели, которые либо присоединяются к доковым облястям Главного окна, либо всплывают как независимые окна инструментов.

##### ШАБЛОН ГЛАВНОГО ОКНА

<http://qt-project.org/doc/qt-5.1/qtdesigner/designer-creating-mainwindows.html>

Для каждого интерфейса пользователя Qt Designer обеспечивает набор различных шаблонов для форм. Шаблон главного окна используется, чтобы создать окна приложения с меню, панелью инструментов и dock widgets.

Окно может иметь только одно меню, но много разных панелей инструментов. Пункты меню можно перетаскивать внутри меню.

*Также показано, как создавать Dock Widgets*.

<http://qt-project.org/doc/qt-5.1/qtdesigner/designer-resources.html>

##### РЕДАКТИРОВАНИЕ РЕСУРСОВ

Qt Designer полностью поддерживает систему ресурсов Qt, давая возможность ресурсам быть определёнными вместе с формами, когда они разрабатываются. Для того чтобы помочь разработчикам управлять ресурсами для их приложений, редактор ресурсов дизайнера позволяет определить ресурсы для каждой формы. Другими словами, каждая форма может иметь отдельный файл ресурсов.

Отмечается, что включаемые ресурсы должны находиться внутри директории файла ресурсов или в одной из её поддиректорий.

После создания файлов ресурсов, его можно активно использовать при разработке формы. Во время изменения свойств со значениями, которые могут быть определены с помощью ресурсов, редактор свойств дизайнера позволяет вам указать ресурс в дополнение к опции выбора файла ресурсов обычным образом.

<http://qt-project.org/doc/qt-5.1/qtdesigner/designer-stylesheet.html>

##### РЕДАКТОР СТИЛЕЙ

В дизайнере есть специальный редактор стилей, при помощи которого можно изменять стили. Для этого следует нажать на виджет правой кнопкой мыши и выбрать пункт «Изменить стили».

<http://qt-project.org/doc/qt-5.1/qtdesigner/designer-using-a-ui-file.html>

##### UI ФАЙЛЫ. СПОСОБЫ ИСПОЛЬЗОВАНИЯ ФОРМ ДИЗАЙНЕРА. АВТОМАТИЧЕСКОЕ И РУЧНОЕ СОЕДИНЕНИЕ СИГНАЛОВ И СЛОТОВ

В Qt UI файлы дизайнера представляют из себя деревья виджетов в формате XML.

Формы можно производить:

* во время компилирования, что означает, что формы могут преобразовываться в код, который компилируется во время построения (при помощи uic);
* во время работы, что означает, что формы производятся при помощи класса QuiLoader, который динамически конструирует виджет при разборе XML файла.

При построении формы во время компиляции при помощи qmake и uic генерируется код, который содержит объект формы. Это структура С++, которая содержит указатели на виджеты формы, макеты, группы кнопок, а также действия; функцию член setupUi(), которая строит дерево виджетов на родительском виджете; функцию член retranslateUi(), которая как-то связана с переводом строковых констант виджетов.

Сгенерированный код может быть использован затем для расширения подклассов стандартных виджетов.

Созданная на этапе компиляции форма может быть использована вами при помощи одного из следующих подходов:

* прямой подход (в исполняемый файл включается файл формы, создаётся объект описания интерфейса Ui::SomeForm, который вызывает функцию setupUi(), которой в качестве аргумента передаётся виджет, который и будет отображать данную форму. Прямой подход следует использовать для управления простыми автономными компонентами вашего приложения).
* подход одиночного наследования с использованием закрытого члена (К сожалению, часто компоненты, созданные при помощи дизайнера, требуют тесной интеграции с остальной частью кода. Это можно исправить при помощи подхода одиночного наследования. В данном случае объект интерфейса формы делается закрытым членом класса, являющегося подклассом формы. Функция setupUi() вызывается в конструкторе виджета, а в качестве аргумента в ней указывается соответственно this. Тогда компоненты формы просто становятся компонентами подкласса класса QWidget. Также необходимо в данном случае включить файл, который генерирует uic при создании формы. Если каждый интерфейс к тому же поместить в отдельное пространство имён, то можно использовать несколько слоёв для одного и того же виджета, например, чтобы использовать их как табуляции).
* Подход одиночного наследования с использованием указателя (При использовании члена указателя необходимо включить пространство имён Ui, которое содержит класс формы. А в подклассе виджета включить закрытый член-указатель на данный класс. В конструкторе класса тогда следует генерировать данный объект и вызывать всё ту же функцию setupUi(this) (для примера смотри данный сайт). В деструкторе же необходимо очистить данный указатель. В данном случае (использование единичного наследования с использованием указателя на виджет) преимущество состоит в том, что заголовочный файл подкласса виджета не включает заголовочный файл формы, в результате чего форма затем может быть изменена без перекомпиляции файлов исходного кода. Это особенно важно, если на класс накладываются ограничения на двоичную совместимость). Авторы рекомендуют данное приближение для библиотек и больших приложений.
* подход множественного наследования (При подходе множественного наследования, подкласс наследует не только класс виджета, но также и класс интерфейса формы. Всё преимущество данного подхода (множественное наследование) сводится к тому, что ко всем виджетам формы можно обращаться напрямую без префикса ui. )

В файле проекта объявление FORMS указывает qmake, какие файлы генерировать при помощи uic.

Генерировать формы во время выполнения можно при помощи модуля QtUiTools, который содержит класс QUiLoader. Для начала необходимо подключить данный модуль, для чего в файл проекта qmake включается объявление QT += uitools. Для создания виджета необходимо создать объект файла, который бы указывал на файл формы, после чего открыть данный файл и передать его в качестве аргумента функции load() класса объекта класса QUiLoader, которая вернёт созданный виджет. После этого файл формы следует закрыть.

Чтобы локализовать объекты на сгенерированной форме, необходимо использовать функцию findChild<class\*>(“name”). При помощи данных виджетов локализуются виджеты на форме, которые присваиваются указателям членам того класса, для которого динамически создавался виджет. Эта возможность позволяет пользователю изменять динамически графический интерфейс.

*Вопрос по ходу. Не понимаю, следует ли в qt на каждый вызов new вызывать delete? Пока ответ таков: при соотношении композиции между объектами указатели следует удалять в деструкторе родительского класса.*

Автоматические соединения: соединения между сигналами и слотами форм, созданных во время компиляции или выполнения, могут быть установлены как вручную, так и автоматически.

* Если мы действуем вручную, то необходимо создать частный слот, который затем при помощи функции connect() соединить с некоторым сигналом.
* Если мы хотим автоматически соединять слот с сигналом, то необходимо его имя написать в следующем формате: on\_object name\_signal name(signal parameters).

*Вообще удобнейшая вещь. Но здесь есть две проблемы. Во-первых, не отражается внутренний смысл слота, так как при разных условиях при нажатии одной и той же кнопки могут использоваться различные слоты. Во-вторых, как временно отсоединять такие слоты.*

##### ЧАСТНЫЕ ВИДЖЕТЫ В ДИЗАЙНЕРЕ. ПРОДВИЖЕНИЕ ВИДЖЕТОВ.

<http://qt-project.org/doc/qt-5.1/qtdesigner/designer-using-custom-widgets.html>

Qt Designer может отображать частные виджеты при помощи гибкого механизма плагинов, позволяющего пользователю и третьей стороне расширять целый ряд проектируемых виджетов. С другой стороны, возможно, использовать существующие виджеты как заполнители для классов виджетов, что обеспечивает похожий API.

По ряду причин (виджеты недоступны во время проектирования, виджеты могут зависеть от платформы и для каждой платформы приходится разрабатывать свой собственный виджет; исходный код частного виджета по каким-то причинам недоступный) некоторые виджеты не могут быть использованы как стандартные виджеты в дизайнере. Тем не менее, в данном случае всё равно можно достичь цели использования частного виджета при помощи такой особенности дизайнера, как продвижение (promotion). В других ситуациях, когда код частного виджета доступен, можно адаптировать частный виджет для использования в дизайнере.

Для продвижения виджета следует выбрать стандартный виджет того класса, который является наиболее близким к частному виджету. Поместить его на форму и в меню выбрать пункт «Promote to», после чего заменить имя стандартного класса на имя нового класса. Если нет близких к данному стандартных виджетов, то следует заменить класс QWidget.

Чтобы использовать частный виджет напрямую в дизайнере, необходимо гарантировать, что плагин будет находиться на том же пути, по которому Qt Designer ищет плагины.

Плагины сохраняются в $QTDIR/plugins/designer, которая загружается, когда Qt Designer начинает работу.

###### СОЗДАНИЕ ЧАСТНЫХ ВИДЖЕТОВ ДЛЯ QT ДИЗАЙНЕРА ПРИ ПОМОЩИ ПЛАГИНОВ

<http://qt-project.org/doc/qt-5.1/qtdesigner/designer-creating-custom-widgets.html>

При использовании частных виджетов в дизайнере все их свойства, сигналы и слоты могут обрабатываться через Qt Designer точно таким же образом, как и для стандартных виджетов. Также частные виджеты будут отображаться, также как отображаются стандартные виджеты.

Для интегрирования виджета в Qt Designer необходимо иметь подходящее описание виджета и соответствующий .pro файл.

Чтобы проинформировать Qt Designer о типе виджета, следует создать подкласс класса [QDesignerCustomWidgetInterface](http://qt-project.org/doc/qt-5.1/qtdesigner/qdesignercustomwidgetinterface.html), который описывает все свойства, которые ваш виджет предоставляет. Большинство из них реализованы как некоторые чисто виртуальные функции, которые возвращают соответствующие свойства в виде возвращаемых значений. Все они перечислены в данной части, а также в описании к данному классу.

Одной из функций является domXml(), которая возвращает отрывок UI файла, который используется фабрикой виджетов дизайнера, чтобы создать частный виджет и его применимые свойства. UI файл может быть загружен при помощи тэга <ui>. Этот же тэг позволяет добавить информацию касательно частного виджета. Тэга <widget> достаточно, если не надо никакой дополнительной информации. Тэг <property> позволяет добавлять некоторые параметры данного виджета, например, размеры по умолчанию.

Ещё одной особенностью данной функции является та, что если она возвращает пустую строку, то виджет не будет установлен в ящик виджетов дизайнера. Но он всё равно может использоваться другими виджетами на форме. Эта особенность используется, чтобы спрятать виджеты, которые пользователям не следует явно создавать, но которые требуются для работы других виджетов. Также нужно описать некоторую информацию для строки каждого свойства.

Особенно хочется отметить, что тэг <addpagemethod> используется для контейнерных классов, которые не являются наследниками стандартных контейнерных классов.

Для обеспечения корректной работы плагинов следует, во-первых, экспортировать плагин, чтобы он мог быть загружен дизайнером (макрос Q\_PLUGIN\_METADATA()); а во-вторых, следует определить класс каждого частного виджета внутри плагина, который Qt Designer будет инстанцировать (макрос QDESIGNER\_WIDGET\_EXPORT).

Следует изучить механизм загрузки.

Также отмечается, что некоторые частные виджеты могут влиять на поведение дизайнера, например, если они вызывают функцию grabKeyboard(). Чтобы предоставить частным виджетам специальное поведение, обеспечьте применение функции initialize(), чтобы настроить процесс конструирования виджета для специального поведения дизайнера. Эта функция будет вызвана первый раз до любого вызова функции createWidget() и может поэтому установить внутренние флаги, которые затем могут быть протестированы, когда Qt Designer вызовет функцию createWidget().

Следует отметить, что если qt настроен на построение как в отладочной моде, так и в моде выпуска, то плагин будет построен только в моде выпуска, поэтому необходимо обеспечить, чтобы плагины также строились в моде выпуска при помощи строки в файле проекта: CONFIG += release. Если плагины построены в моде, которая несовместима с дизайнером, то они не будут загружены и установлены. Также следует обеспечить, чтобы плагины устанавливались вместе с другими плагинами виджетов дизайнера: Переменная $$[QT\_INSTALL\_PLUGINS] является указателем на место локализации установленных плагинов qt.

Также можно настроить Qt Designer на просмотр других мест на наличие плагинов при помощи установки переменной среды QT\_PLUGIN\_PATH перед запуском приложения.

Для более подробного руководства по настройке путей библиотек и плагинов следует смотреть [QCoreApplication::libraryPaths](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html#libraryPaths)().

Также отмечается, что в реальном мире вы не хотите иметь зависимостей приложения от используемого частного виджета. Поэтому заголовочные файлы частного виджета можно поместить в специальный .pri файл, а затем добавить данный файл в .pro файл.

Также можно с этой целью использовать разделяемые библиотеки, которые также можно создавать при помощи дизайнера.

###### РАСШИРЕНИЯ ДЛЯ ЧАСТНЫХ ВИДЖЕТОВ

<http://qt-project.org/doc/qt-5.1/qtdesigner/designer-creating-custom-widgets-extensions.html>

когда вы создали плагин частного виджета для дизайнера, вы можете обеспечить его ожидаемым поведением и функциональностью внутри рабочего пространства дизайнера, используя расширения частных виджетов.

Рассмотрим типы расширений. [QDesignerContainerExtension](http://qt-project.org/doc/qt-5.1/qtdesigner/qdesignercontainerextension.html) необходим, когда используется частный многостраничный контейнер. Он позволяет добавлять и удалять страницы из многостраничного контейнера. [QDesignerTaskMenuExtension](http://qt-project.org/doc/qt-5.1/qtdesigner/qdesignertaskmenuextension.html) полезен для частных виджетов, так как позволяет добавлять частные записи в меню задач дизайнера. [QDesignerMemberSheetExtension](http://qt-project.org/doc/qt-5.1/qtdesigner/qdesignermembersheetextension.html) позволяет манипулировать функциями членами виджета, когда производится соединение сигналов и слотов. [**QDesignerPropertySheetExtension**](http://qt-project.org/doc/qt-5.1/qtdesigner/qdesignerpropertysheetextension.html)**,** [**QDesignerDynamicPropertySheetExtension**](http://qt-project.org/doc/qt-5.1/qtdesigner/qdesignerdynamicpropertysheetextension.html)позволяют настраивать, как свойства данных виджетов должны отображаться в редакторе свойств дизайнера.

Qt Designer всегда запрашивает расширения частного виджета, когда к нему происходит обращение.

Также следует иметь в виду, что все виджеты имеют страницы свойств по умолчанию, поэтому после создания расширенных страниц свойств, старые страницы будут переписаны.

Для того чтобы подключить расширение, следует некоторому классу унаследовать классы QObject и класс расширения. Так как мы применяем интерфейс, мы должны гарантировать, что он станет известен метаобъектной системе, для чего следует использовать макрос Q\_INTERFACE() в определении класса расширения. *Пример приведён на данной странице.*

Это позволяет дизайнеру использовать функцию [qobject\_cast](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html#qobject_cast)(), чтобы запрашивать поддерживаемые интерфейсы, используя только указатель на объект QObject.

В дизайнере расширения не создаются, пока они не понадобятся. Поэтому когда применяются расширения, следует породить подкласс класса [QExtensionFactory](http://qt-project.org/doc/qt-5.1/qtdesigner/qextensionfactory.html), который способен инстанцировать расширения. Также данную фабрику следует зарегистрировать при помощи менеджера расширений дизайнера, который управляет конструированием расширений.

Когда расширение запрашивается, менеджер расширений дизайнера запустит через его зарегистрированные фабрики вызов функции [QExtensionFactory::createExtension](http://qt-project.org/doc/qt-5.1/qtdesigner/qextensionfactory.html#createExtension)() для каждой из них, пока он не найдёт некоторую, которая будет способна создавать запрашиваемое расширение для выбранного виджета. Далее эта фабрика создаст экземпляр данного расширения.

Если данную функцию переопределить в подклассе, то можно создать своё собственное расширение, такое как, например, расширение для многостраничного виджета.

Регистрация фабрики расширений обычно происходит в функции [QDesignerCustomWidgetInterface::initialize](http://qt-project.org/doc/qt-5.1/qtdesigner/qdesignercustomwidgetinterface.html#initialize)(), которая переопределяется в интерфейсе для плагина.

##### XML-СХЕМА ДЛЯ ФОРМАТА UI ФАЙЛА

<http://qt-project.org/doc/qt-5.1/qtdesigner/designer-ui-file-format.html>

в данной части описывается формат UI файла, используемый в дизайнере. Он обычно описывается на языке XML.

Авторы подчёркивают, что данный формат может быть изменён в будущих версиях qt.

##### МАКЕТЫ В ДИЗАЙНЕРЕ

<http://qt-project.org/doc/qt-5.1/qtdesigner/designer-layouts.html>

Перед тем как использовать форму, объекты на форме необходимо поместить в макеты. Это гарантирует то, что объекты будут отображаться правильно, когда форма будет просматриваться или использоваться в приложении. Размещение объектов в макете также гарантирует, что они будут правильно изменять размер, когда будет изменяться размер формы.

Наипростейший способ создания макета – это создание макета предопределённого вида для группы объектов, которые уже находятся на форме. После того как виджеты были вставлены в макет, становится невозможно перемещать или изменять их размер по отдельности. Макет сам управляет геометрией каждого виджета внутри себя, принимая во внимание подсказки размеров, обеспечиваемые разделителями.

Для изменения геометрии объектов следует либо разрушить макет (Ctrl+O) и изменить размер отдельного объекта, либо изменить размер самого макета в целом. Также можно добавлять или удалять из макета разделители, чтобы влиять на расположение и размеры объектов.

Макет самого верхнего уровня может быть установлен при отмене любого выделения и применении макета ко всей форме в целом. Чтобы проверить, был ли создан макет самого верхнего уровня, следует произвести предварительный просмотр вашего виджета и попробовать изменить его размер.

В целом для формы можно выбрать два варианта распределения виджетов:

* использование вложенных горизонтальных и вертикальных макетов;
* а также использование сеточного макета, который является менее гибким, но лучше подходит для некоторых случаев.
* Макет формы размещает виджеты в два столбца, причём первый столбец заполняется метками, а второй столбец содержит виджеты с полями ввода: строка редактирования, счётчик и т.д. данный класс также придерживается того стиля, который характерен для данной платформы.
* Также есть макет разделитель (splitter), который, хотя является контейнером, но обрабатывается как макет. Он также может выравнивать элементы по вертикали или горизонтали, но при этом он позволяет пользователю приводить в порядок количество пространства, выделенного каждому объекту.

##### СОХРАНЕНИЕ, ПРОСМОТР И ПЕЧАТЬ ФОРМ В QT ДИЗАЙНЕРЕ

<http://qt-project.org/doc/qt-5.1/qtdesigner/designer-preview.html>

полезно предварительно просматривать форму перед использованием. Для этого следует зайти в меню Form и выбрать Preview, или нажать комбинацию горячих клавиш Ctrl+R.

Также можно просматривать окно с использованием различных стилей. Для этого следует зайти в меню Edit и выбрать пункт Preferences. После этого появится диалог, в котором следует активировать флажок Print/Preview Configuration. Там же можно выбрать стиль. А также можно выбрать частный стиль, созданный при помощи средств qt.

Также возможно просматривать код, сгенерированный uic для формы дизайнера. Для этого в меню Form следует выбрать пункт View Code.

Формы можно распечатывать или сохранять как изображения.

### QMAKE

<http://qt-project.org/doc/qt-5.1/qmake/qmake-manual.html>

qmake позволяет упростить процесс построения разрабатываемого проекта при помощи разных платформ. Он автоматизирует генерирование Make-файлов, так что только несколько линий информации необходимо, чтобы создать каждый Make-файл.

Что такое Make-file???

Он генерирует Make-файл, основанный на информации в файле проекта (.pro). Файлы проекта создаются разработчиком и обычно они простые, но могут быть созданы и более сложные файлы проектов для сложных проектов.

Также qmake содержит дополнительные средства для поддержки разработки с помощью Qt, автоматически включая правила построения для moc и uic. Также qmake может быть использован для генерирования проекта для Microsoft Visual Studio.

#### ОБЗОР QMAKE

<http://qt-project.org/doc/qt-5.1/qmake/qmake-overview.html>

qmake расширяет информацию каждого проектного файла (.pro) в Make-файл, который выполняет необходимые команды для компиляции и связывания.

Проекты описываются при помощи содержания файлов проекта (.pro). qmake использует данную информацию, чтобы сгенерировать Make-файлы, которые содержат все команды, которые необходимы для построения каждого проекта. Файл проекта обычно содержит список заголовочных файлов и файлов исходного кода, общую информацию конфигурирования, а также любые специфические для приложения детали, такие как список дополнительных библиотек, с которыми следует связаться, а также список дополнительных путей, которые будут использоваться. Файл проекта также может включать и некоторые другие элементы, такие как объявления переменных, комментарии, встроенные функции и некоторые простые структуры управления.

Для простых проектов достаточно создавать список заголовочных файлов и файлов исходного кода, а также некоторые базовые настроечные параметры. Можно создавать более сложные файлы проектов для более сложных проектов:

* Можно использовать некоторые шаблоны для построения приложения или библиотеки. С этой целью следует использовать визард Qt Creator.
* Также можно использовать qmake для генерирования файла проекта. Для этого следует использовать опции командной строки qmake.
* Также может оказаться полезным использовать некоторые специфичные для платформы переменные.
* Также qmake предоставляет возможность управлять процессом построения.

#### НЕКОТОРЫЕ ОСОБЕННОСТИ QMAKE

<http://qt-project.org/doc/qt-5.1/qmake/qmake-tutorial.html>

Знак «\» после имён файлов позволяет объявить список заголовочных файлов.

HEADERS используется для объявления заголовочных файлов.

TARGET используется для определения имени для того, что разрабатывается в проекте. По умолчанию имя задаётся таким, какого оно для файла проекта. А расширение получающихся файлов затем определяется на этапе сборки в зависимости от платформы.

Для того чтобы добавить отладочную версию приложения следует использовать переменную DEBUG += debug

Что такое отладочная версия приложения.

Здесь создаётся дополнительное пространство имён, которое указывает, что при сборке приложения на платформе Windows следует включить вот такие вот файлы в проект. Если построение производится для какой-либо другой платформы, то qmake просто игнорирует это.

Символ «!» используется для отрицания.

Также можно определять более чем одно условие для выполнения некоторой информации построения. Например, вы желаете использовать платформу Windows, но при этом вы также хотите увидеть некоторую отладочную информацию при помощи консоли. С этой целью используются вложенные области видимости. Первая создаёт одну область видимости, а вторая создаёт вторую область видимости, которая находится внутри первой.

#### СОЗДАНИЕ ФАЙЛОВ ПРОЕКТА

<http://qt-project.org/doc/qt-5.1/qmake/qmake-project-files.html>

Эти простые программные файлы используют простой декларативный стиль, определяющий стандартные переменные, чтобы указать исходные и заголовочные файлы, что используются в проекте. Сложные проекты могут использовать структуры контроля потоков, чтобы настроить процесс построения. Поддержка использования простых программных конструктов позволяет вам описывать различные процессы построения для разных платформ и сред.

Сейчас опишем различные типы элементов, которые используются в файле проекта.

* Переменные используются в файле проекта для удержания списков строк. В наипростейшем проекте эти переменные информируют qmake о настройках конфигурации, которые следует использовать, или обеспечивают имена файлов и путей в процессе построения. qmake просматривает определённые переменные в каждом файле проекта и использует их содержание, чтобы определить, что следует написать в Make-файле. Например, HEADERS и SOURCES переменные используются, чтобы сказать qmake о заголовочных и исходных файлах, которые находятся в той же директории, что и файл проекта. Если предварять имя переменной символом «$$», то тогда считывается значение переменной. Данный символ также часто используется встроенными функциями, которые управляют строками и списками значений. Обычно пробелы разделяют значения в обозначении переменных. Если же названия переменных сами включают пробел, то название заключается в двойные кавычки. То же самое касается и путей вместе с библиотеками. Перечислим наиболее часто используемые переменные:
* CONFIG – общие настройки конфигурации проекта; Переменная CONFIG определяет настройки и особенности, при помощи которых должен быть настроен проект. Проект должен быть построен либо в моде отладки, либо в моде выпуска. Для обеспечения построения двух мод следует присвоить данной переменной значение debug\_and\_released. Если присвоить данной переменной два значения по отдельности, то будет эффект на моду выпуска. Это связано с особенностями построения Make-файла. Если переменная CONFIG содержит значение qt, поддержка qmake приложений qt доступна. Это даёт возможность более тщательно настроить, какой из модулей qt используется в вашем приложении. Это достигается при помощи переменной QT, которая может быть использована для объявления требуемых модулей расширения. По умолчанию данная переменная содержит gui и core модули. Поэтому всегда следует использовать оператор «+=» для добавления новых модулей. Если нужно удалить некоторый модуль из проекта, то используется оператор «-=». Если в переменной CONFIG определить по отдельности моды отладки и выпуска, то мода отладки просто перепишет моду выпуска. Для одновременного создания обоих мод используется значение данной переменной debug\_and\_release. Если данной переменной присвоить значение: build\_all (гарантирует, что по умолчанию в проекте будут использоваться обе моды для построения).
  + DESTDIR – директория, в которой исполняемый или двоичный файл будут помещены;
  + FORMS – список UI файлов, которые обрабатываются uic;
  + HEADERS – список имён заголовочных файлов, используемых во время построения проекта;
  + QT – список модулей qt, используемых в проекте; Для построения плагина для дизайнера следует присвоить особые значения переменной QT: widgets designer.
  + RESOURCES – список файлов ресурсов (.qrc), которые следует включить в конечный проект.
  + SOURCES – список файлов исходного кода, которые следует использовать во время построения проекта;
  + TEMPLATE – шаблон, который используется для проекта. Это определяет, будет ли конечный продукт приложением, библиотекой или плагином.
  + Если вы используете другие библиотеки в вашем проекте, то вам необходимо объявить их в файле проекта. Для этого в переменную LIBS следует добавить путь библиотеки и её название.
  + Пути, которые содержат заголовочные файлы, также должны быть определены подобным образом в переменной INCLUDEPATH.
  + DEPENDPATH (поиск путей зависимостей для приложения).
  + DEFINES (список других директив препроцессора, необходимых приложению).
* Комментарии начинаются с символа «#» и продолжаются до конца линии. Чтобы включить данный символ в названия, следует использовать встроенную переменную LITERAL\_HASH.
* qmake обеспечивает набор встроенных функций для обеспечения обработки содержания переменных. Наиболее часто используемая в файлах проекта функция – это функция include(), которая в качестве аргумента получает имя файла. Содержание соответствующего файла тогда включается в содержание файла проекта в том месте, где находится данная функция. Обычно её используют для включения других файлов проекта. Более сложные операции, которые обычно требуют циклы, обеспечиваются встроенными функциями, такими как find(), unique(), count(). Эти функции, как и многие другие, используются для манипулирования строками, путями, поддержки ввода пользователя, а также вызова внешних служб. Функцию CONFIG() следует использовать, чтобы протестировать присутствие определённых настроек конфигурации.
* Фигурные скобки позволяют создавать условия выполнения и функции внутри данных скобок выполняются только в том случае, если условие выполняется.

Переменная TEMPLATE используется, чтобы определить тип проекта, что будет построен. Если это не объявляется в файле проекта, qmake полагает, что следует построить приложение. Перечислим типы проектов:

* app – приложение;
* lib – библиотека;
* subdirs – Make-файл содержит правила для поддиректорий, определённых при помощи переменной SUBDIRS. Каждая поддиректория должна содержать свой собственный файл проекта;
* vcapp – проект приложения для Visual Studio;
* vclib, vcsubdirs – то же самое.

Когда используется шаблон subdirs, qmake генерирует Make-файл, чтобы проверить каждую определённую поддиректорию, обработать любой файл проекта, который будет там найден и запустить инструмент построения для вновь созданного Make-файла.

*Чем отличается мода отладки от моды выпуска внутренне.*

Есть особенные настройки, которые оказывают эффект, когда используются на соответствующей платформе:

* qt – проект является приложением qt и его следует связывать с библиотекой qt. Это значение добавляется по умолчанию, но его можно удалить с использованием qmake для проекта не qt;
* x11 – проект является X11 приложением или библиотекой. Это значение не нужно, если цель использует Qt.

Каждый из данных шаблонов обеспечивает вас более специализированными настройками конфигурации, чтобы улучшить процесс построения.

qmake может быть настроен при помощи экстра особенностей конфигурации, которая определена в специальном файле особенностей (.prf). Эти особенности часто обеспечивают поддержку частных инструментов, которые используются во время процесса построения. Для подсоединения некоторых особенностей построения следует добавить их имена в переменную CONFIG.

#### ПОСТРОЕНИЕ ПРОЕКТОВ ОБЩИХ ТИПОВ

<http://qt-project.org/doc/qt-5.1/qmake/qmake-common-projects.html>

* app шаблон говорит qmake строить программное приложение. При помощи дополнительных параметров можно также уточнить тип приложения. Эти параметры являются значениями переменной CONFIG:
* windows – графическое приложение Windows
* console – консольное приложение windows
* testcase – приложение является автоматизированным тестом. testcase проект – это проект для выполнения автоматизированных тестов. Любое приложение можно отметить как автоматизированный тест. Тест считается пройденным, если он заканчивается с нулевым кодом.

При использовании данного шаблона qmake распознаёт следующие переменные:

HEADERS, SOURCES, FORMS, LEXSOURCES (список файлов исходного кода Lex), YACCSOURCES (список файлов исходного кода Yacc), TARGET (имя исполняемого файла, по умолчанию совпадает с названием проекта), DESTDIR (директория, где располагается исполняемый файл), DEFINES (список других директив препроцессора, необходимых приложению), INCLUDEPATH (список дополнительных путей, необходимых приложению), DEPENDPATH (поиск путей зависимостей для приложения) (что это такое???), VPATH (путь поиска поддерживаемых файлов), DEF\_FILE (только для windows, .def файл, который ссылается по отношению к приложению???), RC\_FILE (только для windows, файл ресурсов для приложения), RES\_FILE (только для windows, файл ресурсов, который ссылается по отношению к приложению).

Рассмотрим некоторые особенности построения модульных тестов в кт:

Выполнение check target может быть настроено при помощи определённых переменных Makefile. Это TESTRUNNER (например, завершает тест, если он не был выполнен в течение определённого времени); TESTAGS (дополнительные аргументы, которые присоединяются к каждой команде теста. – это задание формата и т. д. ). Их следует устанавливать во время вызова make инструмента, но не в файле проекта. Большинство инструментов поддерживают установку данных переменных прямо в командной строке.

* CONFIG: insignificant\_test – код выхода теста будет игнорироваться во время проверки make.
* Часто тесты пишутся при помощи [QTest](http://qt-project.org/doc/qt-5.1/qttestlib/qtest.html) или [TestCase](http://qt-project.org/doc/qt-5.1/qtquick/testcase.html), но нет требования использовать именно CONFIG+=testcase and make check.

Рассмотрим некоторые особенности построения библиотек:

При использовании шаблона lib также присутствует дополнительная переменная VERSION, которая указывает версию библиотеки.

К переменной CONFIG могут быть добавлены следующие значения, чтобы указать тип библиотеки, которая будет строиться:

* dll (динамическая библиотека),
* staticlib (статическая библиотека),
* plugin (плагин).

Важно особо отметить, что имя библиотеки является платформенно зависимым, так как для некоторых платформ существуют особые правила написания имён файлов библиотек.

Цели в различных модах построения можно присвоить различные имена.

#### ВЫПОЛНЕНИЕ QMAKE В КОМАНДНОЙ СТРОКЕ. МОДЫ И НАСТРОЙКИ.

<http://qt-project.org/doc/qt-5.1/qmake/qmake-running.html>

поведение qmake можно настраивать при помощи специальных инструкций в командной строке. При этом можно получать разного рода отладочную информацию.

Синтаксис запуска qmake имеет следующую форму:

* qmake [mode] [options] files.

qmake может работать в двух модах: в моде по умолчанию он использует для построения Make-файла информацию из файла проекта. Моду следует устанавливать до установки каких-либо опций:

* -makefile (будет создаваться Make-файл);
* -project (будет создаваться файл проекта).

Файлы в аргументе представляют собой несколько файлов проектов, из которых и будет строиться Make-файл. Теперь перечислим некоторые опции.

* Основные опции:
  + -help (qmake будет предоставлять некоторую помощь),
  + -o file (будет записывать выходную информацию в некоторый файл),
  + -d (печатает также и отладочную информацию),
  + -t tmpl (задаётся шаблон цели),
  + -tp prefix (добавляет префикс к переменной шаблона),
  + -Wall (qmake будет сообщать все известные предупреждения),
  + -Wnone (предупреждений не будет),
  + -Wparser (только ошибки синтаксиса),
  + -Wlogic (отображается информация о некоторых логических ловушках).
* Настройки для моды мэйкфайла
  + -after (qmake обработает присваивания, написанные после имён файлов в командной строке. Это присваивания значений некоторым переменным. Их можно присваивать и до имён файлов, но тогда без команды after),
  + -nocache (qmake будет игнорировать qmake.cache файлы),
  + -nodepend (qmake не будет генерировать информацию о зависимостях),
  + -cache file (qmake будет использовать данный файл как кэш файл и игнорировать другие найденные файлы),
  + -spec spec (qmake будет использовать данный путь для получения информации о платформе и компиляторе и игнорировать значение переменной QMAKESPEC).
* Настройки для моды проекта:
  + -r (qmake будет просматривать обеспеченные директории рекурсивно),
  + -nopwd (qmake не будет смотреть в вашу текущую директорию для исходного кода. Он будет только использовать определённые файлы). В названиях файлов можно указывать также и директории, где находятся некоторые файлы, но тогда эти директории будут включены в переменную DEPENDPATH.

#### СПЕЦИФИЧЕСКИЕ ПАРАМЕТРЫ ДЛЯ ПЛАТФОРМ

<http://qt-project.org/doc/qt-5.1/qmake/qmake-platform-notes.html>

в qt существуют некоторые специфичные для некоторых платформ переменные. В частности для платформы Windows можно создавать проекты вижуал студио.

#### ЯЗЫК QMAKE

<http://qt-project.org/doc/qt-5.1/qmake/qmake-language.html>

* операторы
  + = присваивание; присваивает данной переменной некоторое значение
  + += присоединение; присоединяет к данной переменной некоторое значение
  + -= удаление некоторых предустановленных значений переменных
  + \*= оператор присоединяет значение к переменной, но только если такого значения ещё не было. Это предотвращает использования в одной переменной нескольких одинаковых значений.
  + ~= - заменяет некоторое значение переменной другим значением переменной.
  + $$ - извлекает значение из переменной
  + $$(environment variable) – функция считывает значение переменной среды во время работы qmake
  + $( environment variable) – получение некоторой информации уже во время обработки Make-файла.
  + $$[] – оператор для доступа к свойствам qmake (в качестве аргумента используется значение переменной с префиксом QT\_). Обычно свойства, доступные с данным оператором, используются для интеграции в qt сторонних плагинов. Переменная предоставляет некоторые свойства qt.
  + ! оператор отрицания, который активно используется в условиях.
  + «:» действует как логическое И.
  + «|» действует как логическое ИЛИ и позволяет соединить вместе несколько условий, требуя, чтобы только одно из них было правильным.
* Области видимости: если условие области видимости выполняется, то все инструкции в ней выполняются. У области видимости такой синтаксис: <condition>{ <command или definition> … } Важно отметить, что первая скобка должна быть на той же строке, что и условие. Области видимости можно вкладывать одна в одну. Если в условии к области видимости использовать такой синтаксис: <условие1>:<условие2>:<условие3>… , то это равносильно тому, что мы имеем вложенные области видимости, причём самая внешняя соответствует самому первому условию. Условие может также быть применимо к единичной строке. В данном случае фигурные скобки просто опускаются, а применяемая строка отделяется от условия при помощи пробелов. Также существует оператор else для областей видимости, который позволяет, создавать сложные тесты для приложения. Каким образом он позволяет создавать сложные тесты для приложения? <условие1> { } else { }
  + Значения переменной CONFIG обрабатываются qmake по-особенному. Значения данной переменной могут обрабатываться как отдельные условия.
  + В директории Qt mkspecs расположена спецификация платформы. Например, можно включить такой код в файл проекта с целью проверки, является ли данная среда средой Linux.

В файле проекта можно создавать свои собственные переменные:

MY\_VARIABLE = value

Данная запись означает, что можно создавать переменные, которые содержат часть имени, совпадающую с содержимым некоторой другой переменной.

qmake содержит некоторые встроенные функции для обработки значений переменных. Эти функции обрабатывают аргументы, передаваемые им, и выдают некоторое значение.

* В qmake можно выделить группу функций замещения, которые располагаются справа от оператора присваивания и преобразуют значения некоторых переменных в те значения, которые необходимо присвоить.
* В qmake есть группа функций, называемых тестовыми, которые могут служить в качестве тестовых выражений. Эти функции не возвращают значение, но генерируют вместо этого правду или ложь.

Можно писать собственные функции.

#### ДОБАВЛЕНИЕ НОВЫХ ОСОБЕННОСТЕЙ КОНФИГУРАЦИИ

<http://qt-project.org/doc/qt-5.1/qmake/qmake-advanced-usage.html>

qmake позволяет вам добавлять собственные значения в переменную CONFIG. Эти средства – это наборы частных функций и определений в файле .prf, которые могут находиться в одной из многих стандартных директорий. Эти местоположения могут быть определены в некотором количестве мест, и qmake проверяет каждую из них в следующем порядке, когда он просматривает файл .prf.

1. QMAKEFEATURES (директории, указанные внутри данной переменной среды и данной переменной свойств),
2. директория средств, которая находится внутри директорий mkspecs, которые могут располагаться в каждой директории, перечисленной в переменной среды QMAKEPATH (например, $QMAKEPATH/mkspecs/<features>),
3. в специальной директории, которая находится в директории, обеспечиваемой переменной QMAKESPEC.
4. В любой директории с особенностями, которая расположена в следующей директории: data\_install/mkspecs.

При этом важно, чтобы директория называлась features/, features/unix, features/win32, features/macx.

Также следует отметить, что файл .prf может иметь название, состоящее только из прописных букв.

Для системы Unix также является обычной практикой использовать инструменты построения для установки приложений и библиотек. Например, директория для файла документации может быть указана следующим образом:

documentation.path = /usr/local/program/doc

documentation.files = docs/\*

первая строка указывает директорию, в которую должны быть установлены файлы из переменной documentation. В данном случае эта переменная содержит директорию docs/.

После того как набор для установки будет полностью описан, вы можете добавить его в список установки при помощи следующей линии:

INSTALLS += documentation

Также при помощи qmake можно создавать специальные команды, которые будут восприниматься только на некоторой платформе.

Если не трогать переменную INSTALLS, то qmake автоматически выполнит установку. По умолчанию у данной переменной есть некоторые предустановленные значения (они описаны в данной части).

В чём состоит механизм установки приложения?

qmake пытается делать всё так, как ожидается от кросс-платформенного средства построения. Но часто выходит неидеально, когда вам действительно нужно использовать платформенно-зависимые команды. Это может осуществлять при помощи специальных инструкций.

Настройка выхода Make-файла производится через объектное API. Объекты определяются автоматически при задании их членов. Например:

mytarget.target = .buildfile

mytarget.commands = touch $$mytarget.target

mytarget.depends = mytarget2

mytarget2.commands = @echo Building $$mytarget.target

Определения выше определяют новый таргет qmake, который называется mytarget, содержащий таргет Make-файла под названием .buildfile, который по очереди генерируется при помощи функции touch(). В конце член .depends определяет, что mytarget зависит от mytarget2, другого таргета, который определяется впоследствии. Второй таргет является просто манекеном, который передаёт некоторую информацию на консоль.

В конце следует использовать некоторую переменную QMAKE\_EXTRA\_TARGETS, чтобы проинструктировать qmake, что этот объект является таргетом, который необходимо построить.

QMAKE\_EXTRA\_TARGETS += mytarget mytarget2

Понятно, что вы также хотите связать данный таргет с таргетом построения, который есть у qmake. Для этого вам просто необходимо включить таргет вашего Make-файла в список PRE\_TARGETDEPS.

Также возможно настроить qmake таким образом, чтобы можно было добавить новые компиляторы и препроцессоры.

new\_moc.output = moc\_${QMAKE\_FILE\_BASE}.cpp

new\_moc.commands = moc ${QMAKE\_FILE\_NAME} -o ${QMAKE\_FILE\_OUT}

new\_moc.depend\_command = g++ -E -M ${QMAKE\_FILE\_NAME} | sed "s,^.\*: ,,"

new\_moc.input = NEW\_HEADERS

QMAKE\_EXTRA\_COMPILERS += new\_moc

Например, сверху вместо moc добавляется некоторый новый компилятор. В данной части руководства описаны все члены, которые можно определить для нового компилятора.

*Пока я не совсем понял данную тему, но в будущем, возможно, прочитав остальную часть руководства, я разберусь с данной проблемой и смогу эффективно её использовать.*

При создании библиотек часто так бывает, что qmake полагается на платформу, для которой пишется библиотека, и считает, что она знает, с какими другими библиотеками данная библиотека связана, и позволяет платформе захватить их самостоятельно. Однако бывает, что этого недостаточно. Например, когда статически связывается библиотека, с которой никакие другие библиотеки не связываются, поэтому никаких зависимостей к ней нет. Но приложение, которое затем будет использовать данную библиотеку должно будет знать, где находятся лексемы, которые данная библиотека использует. Qt пытается следить за зависимостями библиотеки, когда это необходимо, если вы явно дадите возможность слежения.

Первый шаг – это разрешить отслеживать зависимости самой библиотеке. Для этого используется следующая строка: CONFIG += create\_prl

Данный способ работает только с шаблоном lib. Когда данная опция доступна, qmake создаст файл с расширением .prl, который сохранит некоторую мета-информацию о библиотеке. Этот метафайл просто как простой файл проекта, но только содержит объявления внутренних переменных. Когда будете устанавливать эту библиотеку, задавая её как таргет в переменной INSTALLS, qmake автоматически скопирует .prl файл в путь для установки.

Вторым шагом является предоставление возможности чтения этой метаинформации в приложениях, которые используют эту статическую библиотеку:

CONFIG += link\_prl

Когда это сделано, qmake обработает все библиотеки, подключенные к приложению и найдёт их метаинформацию.

*Вот не совсем я понял, о чём здесь идёт речь.*

Файлы qmake должны быть созданы только при помощи qmake и не должны переноситься с платформы на платформу, так как они имеют некоторую платформенно-зависимую информацию.

#### ИСПОЛЬЗОВАНИЕ ПРЕДКОМПИЛИРОВАННЫХ ЗАГОЛОВКОВ

<http://qt-project.org/doc/qt-5.1/qmake/qmake-precompiledheaders.html>

Предкомпилируемые заголовочные файлы (PCH) – это особенность работы некоторых компиляторов, которая позволяет компилировать стабильное тело кода и сохранять скомпилированное состояние кода в бинарном файле. Во время последовательных компиляций, компилятор будет загружать сохранённое состояние и продолжать компилирование определённого файла. Каждая последующая компиляция будет быстрее, так как стабильный код не нужно будет перекомпилировать.

Qt поддерживает такие файлы для следующих компиляторов и платформ: Windows: nmake, VS 2008 and later; Mac Os X: Makefile, Xcode; Unix: GCC 3.4 and above.

Чтобы добавить предкомпилируемые файлы в ваш проект следует просто добавить его в переменную PRECOMPILED\_HEADER. Данный файл не следует включать в переменную HEADERS, так как это будет сделано автоматически, если обеспечивается поддержка предкомпилируемых заголовочных файлов.

Существует некоторая опция precompiled\_header, которая проверяет, есть ли возможность предкомпиляции на данной платформе. Затем в блок этого условия можно включить опцию для использования заголовочных файлов предкомпилирования.

Существует некоторая проблема, связанная с тем, что на некоторых платформах имена файлов объектов имеют тот же суффикс, что и заголовочные файлы предкомпиляции. Это может вызвать то неприятное обстоятельство, что будет существовать два разных файла с одинаковым именем. Поэтому для таких файлов следует придумывать особенные имена.

*Какого предназначение DEFINE, IFN\_DEF, pragma и т. д.?*

*В большом проекте это может вызвать дополнительные трудности, связанные с тем, что если код явится всё же не таким стабильным, то придётся искать этот заголовочный файл предкомпиляции, затем ещё один, и в конце мы придём к тому, что забудем, где и когда что включалось, а где и когда всё выключалось.*

#### КОНФИГУРИРОВАНИЕ QMAKE

[*http://qt-project.org/doc/qt-5.1/qmake/qmake-environment-reference.html*](http://qt-project.org/doc/qt-5.1/qmake/qmake-environment-reference.html)

при помощи следующей строки вы можете создать некоторые частные свойства для qmake и присвоить им значения: qmake -set PROPERTY VALUE

чтобы получить свойства qmake, следует написать следующую команду: qmake -query PROPERTY

если вы желаете получить все свойства qmake, то нужно вызвать такую строку: qmake –query

qmake содержит также некоторые встроенные свойства, которые перечислены в таблице в данной части; все они начинаются с QMAKE\_ или QT\_HOST, или QT\_INSTALL, или QT\_SYSROOT, или QT\_VERSION. При вызове данной команды информация будет сохранена в объект QSettings, который может сохраняться в разных платформах в разных местах.

Что касается переменной QT\_VERSION, то рекомендуется, чтобы вы запрашивали не номер версии qt, а номер версии некоторого модуля qt: $$QT.<module>.version.

Qt требует некоторое описание платформы и компилятора, которое содержит много значений по умолчанию, которые используются, чтобы сгенерировать Make-файл. Стандартный пакет qt содержит много таких файлов, которые находятся в директории mcspec.

QMAKESPEC будет автоматически добавлена к переменной INCLUDEPATH.

Кэш-файл – это специальный файл qmake, чтобы находить настройки, которые не определяются в файле конфигурирования qmake.conf, в файле проекта, а также в командной строке. Когда qmake запускается, он сначала ищет файл .qmake.cache в родительской директории текущей директории, если только не определена команда –nocache. Если qmake не находит данный файл, то он просто молча игнорирует данный шаг, а затем просто переходит к обработке файла проекта.

Также в qt существуют некоторые переменные, которые хранят расширения файлов qt. Все они перечислены в данной части и могут быть изменены, что требуется на некоторых платформах. Все они начинаются с QMAKE\_. Это довольно важная и полезная особенность.

#### ССЫЛКИ

<http://qt-project.org/doc/qt-5.1/qmake/qmake-reference.html>

все функции qmake делятся на замещающие и тестирующие. Замещающие возвращают некоторое значение, а тестирующие возвращают булево значение. Функции применяются в двух местах:

* фундаментальная функциональность обеспечивается встроенными функциями;
* более сложные функции применяются в библиотеке файлов функций (.prf).

#### ПЕРЕМЕННЫЕ

<http://qt-project.org/doc/qt-5.1/qmake/qmake-variable-reference.html>

Фундаментальное поведение qmake подвергается влиянию объявлений переменных, которые определяют процесс построения каждого проекта. Некоторые являются общими для всех платформ, тогда как другие используются, чтобы настроить поведение на конкретных платформах.

* Переменная CONFIG определяет конфигурацию проекта и настройки компилятора. Значение распознаётся внутри qmake и имеет особое предназначение. К данной переменной можно добавлять также собственные значения.
* DEPENDPATH – определяет все директории, в которые следует смотреть, чтобы определить зависимости, когда просматриваются заголовочные файлы.
* DESTDIR – определяет, где разместить целевой файл.
* INCLUDEPATH – определяет директории #include, в которых следует искать во время компиляции проекта.
* INSTALLS – определяет список ресурсов, которые будут установлены при процедуре установки.

*Что такое Lex файлы?*

* LIBS – определяет список библиотек, которые следует связать с проектом. Символ библиотеки –l является специфическим для Unix, однако библиотека будет корректно обрабатываться и на платформе Windows. По умолчанию список библиотек, которые будут использованы qmake уменьшается до списка уникальных имён. В переменной CONFIG есть специальная переменная, которая позволяет это отменить.
* LITERAL\_HASH – макрос для символа # для использования в именах.
* MAKEFILE определяет имя генерируемого Make-файла.
* MOC\_DIR – определяет директорию, в которой должны быть сохранены все промежуточные файлы moc.
* OBJECTS – переменная образуется из SOURCES так, что она содержит имена объектных файлов с расширением, специфичным для данной платформы. Что такое объектный файл???
* OBJECTS\_DIR – директория, в которой содержатся данные промежуточные объектные файлы.

*Получается, что при сборке среда разработки создаёт из каждого файла исходного кода некий объектный файл, из которых затем создаётся исполняемый файл приложения?*

* POST\_TARGETDEPS, PRE\_ TARGETDEPS мне не понятны.
* PRECOMPILED\_HEADER – указывает заголовочные файлы предкомпиляции.
* PWD – содержит полностью путь, по которому располагается анализируемый файл.
* OUT\_PWD – то же, но для сгенерированного Make-файла.
* QMAKESPEC – системная переменная, которая содержит полный путь конфигурирования qmake, когда генерируется Make-файл.
* QMAKE\_CC – переменная, в которой задаётся компилятор С, который должен использоваться для компиляции вставок языка С. Необходимо задавать только имя его исполняемого файла, а также путь к нему в переменной PATH.
* QMAKE\_CХХ – определяет компилятор, который будет использован для компиляции кода на С++ подобно переменной QMAKE\_CC.
* QMAKE\_EXTRA\_COMPILERS – определяет список дополнительных компиляторов или препроцессоров.
* QMAKE\_EXTRA\_TARGETS – определяет список дополнительных таргетов.
* QMAKE\_LFLAGS – определяет набор флагов, которые устанавливаются для компоновщика. Если вы хотите изменить флаги для некоторой платформы или типа проекта, используйте одну из специализированных переменных вместо данной.
* QT – определяет модули qt, которые будут использованы в проекте. Значения данной переменной перечислены в данной части.
* QTPLUGIN – определяет список имён статических плагинов Qt, которые необходимо присоединить к приложению, так что они будут доступны как встроенные ресурсы.
* REQUIRES – определяет список значений, которые оцениваются как условия. Если одно из условий является ложью, qmake пропускает данный проект и его поддиректории во время построения.
* SUBDIRS – *в данной директории определены некоторые операции, которые можно использовать для установления порядка сборки поддиректорий.*
* TRANSLATIONS – определяет список файлов переводов (.ts), которые содержат перевод текста интерфейса пользователя на другой язык.
* VERSION – определяет номер версии приложения.
* VPATH – пути, к которым qmake обращается, когда не может открыть некоторый файл.
* \_PRO\_FILE\_ - местоположение файла проекта, который используется.
* \_PRO\_FILE\_PWD – то же, но только к директории. Не следует пробовать переписать значение данной переменной.

*Все остальные переменные относятся к одной из следующих категорий:*

* *специальные переменные для некоторой среды;*
* *переменные для qmake,*
* *переменные, которые редко будут использоваться.*

#### ФУНКЦИИ ЗАМЕЩЕНИЯ

<http://qt-project.org/doc/qt-5.1/qmake/qmake-function-reference.html>

это функции для обработки содержания переменных во время процесса конфигурирования. Обычно они возвращают значения, которые можно присвоить другим переменным. Эти значения можно вернуть при помощи префикса к функциям $$. Функции замещения можно поделить на встроенные функции и библиотеки функций.

*Далее буду подробно описывать лишь наиболее непонятные или наиболее интересные для меня функции.*

*В большинстве случаев функции мне понятны, но при этом более подробно я буду их использовать только тогда, когда в них возникнет необходимость.*

#### ТЕСТОВЫЕ ФУНКЦИИ

<http://qt-project.org/doc/qt-5.1/qmake/qmake-test-function-reference.html>

рекомендуется использовать тестовую функцию requires() вместо переменной REQUIRES.

Тестовые библиотеки находятся в файлах .prf.

### QT LINGUIST

<http://qt-project.org/doc/qt-5.1/qtlinguist/qtlinguist-index.html>

qt предоставляет прекрасный инструмент для создания локализованных приложений. Это Qt Linguist. Менеджеры проекта, переводчики и разработчики могут использовать этот инструмент для достижения своих целей. Они могут использовать lupdate инструмент для синхронизации исходного кода и переводов, а также lrelease инструмент для создания файлов динамических переводов, которые использует выпускаемое приложение.

Переводчики используют Qt Linguist для перевода текста в приложениях.

Разработчики способны создавать приложения Qt, которые содержат переводимый текст.

#### ОБЗОР ПРОЦЕССА ПЕРЕВОДА

<http://qt-project.org/doc/qt-5.1/qtlinguist/linguist-overview.html>

большинство текста в приложении, который следует переводить, представляет из себя короткие слова или фразы, в основном отображаемые на элементах управления. Обычно разработчик пишет эти фразы в тексте.

Менеджер выпуска генерирует несколько файлов перевода, которые производятся из исходного кода и передаёт их переводчику. Переводчик открывает эти файлы при помощи Qt Linguist, вводит переводы и сохраняет результаты назад в файлы трансляции, которые они затем передают назад менеджерам выпуска. Затем менеджер выпуска генерирует сжатые версии данных файлов, готовых к использованию приложением.

#### РУКОВОДСТВО ПО ЛИНГВИСТУ ДЛЯ МЕНЕДЖЕРОВ

<http://qt-project.org/doc/qt-5.1/qtlinguist/linguist-manager.html>

два инструмента доступны менеджерам:

* lupdate
* lrelease,

которые могут обрабатывать файлы проекта qmake или работать напрямую с файловой системой.

В данной части перечислен порядок создания файлов перевода для локализованных приложений:

* вначале создаются файлы исходных переводов (TS) (при помощи lupdate), которые содержат только переводимую информацию.
* Затем они передаются переводчику, который при помощи Qt Linguist создаёт переводы. lupdate просматривает все файлы исходного кода и форм и выбирает из них все текстовые выражения, на основании чего создаются TS файлы.
* Затем lupdate используется для перевода новых фрагментов текста, которые были добавлены во время разработки приложения.
* Затем lrelease используется для построения из TS-файлов QM-файлов, которые используются приложением.

Файлы должны быть помещены в файл проекта приложения, чтобы lupdate знал, какие файлы переводов ему обрабатывать.

TS – это простой, читаемый людьми XML формат, который может использоваться системами контроля версий.

lrelease преобразует TS-файл в QM-файл – это файл в двоичном формате, который используется локализованными приложениями. TS-файлы должны быть заданы либо в командной строке, либо в файле проекта Qt.

Важно отметить, что если файлы QM не создались, то приложение будет хорошо работать с текстом, который находится в файлах исходного кода. Там есть замечание, согласно которому lrelease переводит только те строки, которые отмечены, как завершённые.

Также могут быть пропущены переводы некоторых слов, вместо которых будет просто вставляться исходный текст из файлов исходного кода.

#### РУКОВОДСТВО ПО ЛИНГВИСТУ ДЛЯ ПЕРЕВОДЧИКОВ

<http://qt-project.org/doc/qt-5.1/qtlinguist/linguist-translators.html>

Qt Linguist – это инструмент для добавления переводов в приложения qt. Данный инструмент записывается вместе с qt. Данный инструмент имеет несколько меню:

* контекст (список контекстов, в которых встречаются строки, которые следует перевести),
* строки (для просмотра переводимых строк, находящихся в контексте),
* ресурсы и формы (для просмотра файлов, где используется данная строка, если этот код доступен),
* фразы и предположения (для просмотра возможных переводов данной строки),
* предупреждения (список строк, которые не прошли проверку тестов),
* область перевода для перевода строк.

*Также можно преобразовывать данный TS-файл в QM-файл. При выполнении команды qmake release все TS-файлы преобразуются в QM-файлы.*

*Также можно оставлять некоторые строки непереведёнными для перевода в будущем.*

*Если некоторая фраза встречается несколько раз, то Qt Linguist при её последующем появлении предлагает предыдущий перевод в разделе Phrases and Guesses, но можно для данной фразы в зависимости от контекста также формировать и новый перевод.*

*Если в одном контексте есть две одинаковые фразы, которые должны быть по-разному переведены, то чтобы их затем выделить из контекста, следует снабдить их разными комментариями разработчика в соответствующем разделе.*

Ускоритель – это клавиша или комбинация клавиш, которая, когда нажата, вызывает выполнение некоторого действия приложением. Есть два вида ускорителей клавиатуры: Alt и Ctrl. Подчёркнутый снизу символ в меню или на кнопке обозначает, что при нажатии Alt+данный символ на клавиатуре инициируется то же действие, что при выборе соответствующего меню или соответствующей кнопки. Чтобы идентифицировать данный ускоритель, следует букву, которая будет инициировать выполнение данного действия, сопроводить спереди символом «&».

Переводчик может изменять букву, которая входит в ускоритель.

* Alt ускорители должны быть разными в рамках одного визуального элемента, но могут быть одинаковыми у разных визуальных элементов.
* Ctrl ускорители могут существовать независимо от любого визуального элемента управления. Они часто инициируют действия из меню, которые в противном случае пришлось бы инициировать при помощи множественного нажатия на клавиши.

Каждый такой ускоритель показан в меню Строк как отдельная строка, например, **Ctrl+Enter**. Также переводчик должен полагаться на разработчика графического интерфейса в том, что он напишет переводимый комментарий к тому, какое действие выполняется данным ускорителем.

*Также предоставляется возможность использования пронумерованных аргументов, которые пишутся одинаково в различных языках. Данные аргументы обозначаются при помощи %i.*

в названиях TS-файлов лучше явно задавать, какая в них размещена локализация: язык и страна. Но также и в самом Qt Linguist есть такая возможность. Для этого необходимо зайти в меню Edit.

Каждый контекст – это некоторый подкласс класса QObject. Каждому контексту могут соответствовать некоторые значки, которые указывают состояние перевода.

В меню строк отображаются все строки из данного контекста. Напротив них также могут ставиться некоторые тематические значки, которые перечислены в данной части.

Если перевод совпадает с исходным текстом, то есть специальная команда, которая копирует исходный текст в область перевода (Translation > Copy from source text). Также строки можно копировать из блока фраз и предположений, щёлкнув по строке двойным щелчком мыши. Также можно использовать batch translation, чтобы взять перевод фраз из книги фраз. Данные книги можно создавать и редактировать. Они создаются в меню Phrases.

Qt Linguist предлагает следующие тесты для проверки текста:

* тест ускорителя проверяет переводимые фразы, которые не имеют подчёркивания или предшествующего символа и наоборот.
* Тесты пунктуации проверяют отличия в пунктуации между исходными и переведёнными фразами, когда это может быть важно.
* Тест фраз проверяет исходные фразы, которые также есть в книге фраз, но чьи переводы отличаются от тех, которые даны в книге фраз.
* Тест местоположения маркеров проверяет, используются ли одинаковые переменные в одних и тех же местах в исходном тексте и в переводе (%1, %2, %3).

Можно одновременно загрузить множество файлов переводов.

#### РУКОВОДСТВО ПО ЛИНГВИСТУ ДЛЯ РАЗРАБОТЧИКОВ

<http://qt-project.org/doc/qt-5.1/qtlinguist/linguist-programmers.html>

Qt минимизирует затраты производительности при использовании переводов, переводя фразы для каждого окна, когда оно создаётся. Во многих приложениях Главное окно встречается только раз. Только окна, которые регулярно создаются и уничтожаются будут вызывать накладные затраты производительности.

Создание приложений с возможностью динамического изменения языка возможно при помощи Qt, но требует определённого количества вмешательства разработчика и будет влиять на производительность во время работы программы.

Для определения файлов ресурсов для переводов в файле проекта qmake имеется переменная TRANSLATIONS. Затем lupdate просматривает переменные SOURCES, FORMS и HEADERS, в которой находит файлы исходного кода, анализирует их и выделяет из них строки.

Если ваш компилятор использует другую кодировку для времени работы, нежели используется в файлах исходного кода, то необходимо установить переменную CODECFORSRC (если вы хотите использовать не символы ASCII).

Если разместить в файлах исходного кода не файлы С++, а, например, файлы QML или JavaScript, то компилятор попытается построить их как будто они являются C++ файлами. *Показано также, куда помещать файлы java script и qml.*

Разрабатывайте ваше приложение так, чтобы оно могло быть использовано в разных регионах и местах без инженерных изменений. Для этого обращайте внимание на следующие моменты при написании ваших приложений:

1. пусть ваше приложение будет искать и загружать подходящие файлы переводов;
2. обозначьте видимый пользователем текст и ускорители Ctrl как целевые для перевода (*в виде комментариев???*);
3. обеспечьте контекст для текста, который следует перевести;
4. устраняйте неоднозначности идентичного текста;
5. используйте пронумерованные аргументы как указатели места, которые замещаются текстом или числовыми значениями (*данная техника пока мне не понятна: где определять замещающий текст???*);
6. интернационализируйте числа, даты, время и валюту;
7. помечайте строки вне функций как переводимые (*не понял, о чём здесь идёт речь*).

Каждый текст, который следует перевести, требует контекст, чтобы помочь переводчику идентифицировать в программе появление текста. В случае многих идентичных текстов, которые требуют разного перевода, переводчик также требует некоторую информацию, чтобы устранить неоднозначность исходного текста. Пометка текста как текста для перевода автоматически вызовет то, что имя класса будет использоваться как базовый контекст информации. В некоторых случаях у разработчика можно потребовать добавить дополнительную информацию, чтобы помочь переводчику.

#### TS ФОРМАТ ФАЙЛОВ

<http://qt-project.org/doc/qt-5.1/qtlinguist/linguist-ts-file-format.html>

далее приведён формат TS-файла. В отличие от формата для формы дизайнера, данный формат описан при помощи DTD. Данный формат может измениться в будущих выпусках qt.

#### ПЕРЕВОДЫ, ОСНОВАННЫЕ НА ТЕКСТОВЫХ ИДЕНТИФИКАТОРАХ

<http://qt-project.org/doc/qt-5.1/qtlinguist/linguist-id-based-i18n.html>

Механизм перевода по ID – система промышленной силы для интернационализации и локализации. Каждому тексту в приложении приписывается уникальный идентификатор, и эти идентификаторы используются напрямую в исходном коде в месте, где находится простой текст. Это требует немного большей работы разработчиков графического интерфейса пользователя, но делает намного более лёгким управление большого числа переводимых текстов.

В одном приложении, однако, можно использовать либо функции, основанные на механизме ID, либо функции для простого текста. При использовании системы ID вместо простого текста, общий метод интернационализации приложения тот же, но есть некоторые специфические детали:

1. Функции и макросы для данной системы отличны от таковых для системы с простым текстом: qsTrId() вместо qsTr(), QT\_TRID\_NOOP() вместо QT\_TR\_NOOP().
2. Use text IDs as user interface strings rather than plain text strings. For example, qsTrId("id-back-not-front") () (*видимо, названия идентификаторов должны удовлетворять действиям пользователя в GUI*)
3. нельзя определять параметр контекста вместе с идентификатором. Для двух одинаковых текстов с разным значением следует использовать разные идентификаторы.
4. //% комментарий в коде как раз и присваивает имя ID (то есть непосредственно текст). Без этого комментария сам идентификатор отобразится на элементе интерфейса пользователя.
5. //: комментарий обеспечивает экстра информацию переводчику. Для данной системы данная экстра информация становится очень существенной, так как без неё вы только имеете идентификатор текста, который для переводчика может ничего не значить.

*Как я понял вся экстра информация помещается прямо в коде в виде комментариев.*

Что касается более продвинутой организации локализации, то здесь необходимо удалить переменную TRANSLATIONS из файла проекта и написать отдельный скрипт, в котором происходит управление переводами: осуществляются все вызовы lupdate и lreaease.

Не понимаю значение комментария вида //~ ?

#### ИНТЕРНАЦИОНАЛИЗАЦИЯ И ЛОКАЛИЗАЦИЯ

<http://qt-project.org/doc/qt-5.1/qtdoc/internationalization.html>

Интернационализация и локализация приложения – это процессы адаптации приложения к разным языкам, региональным отличиям и техническим требованиям целевого рынка. Интернационализация значит написание такого приложения, которое может быть адаптировано к разным языкам и регионам без внесения инженерных изменений в проект. Локализация означает адаптацию интернационализированного программного обеспечения для специального региона или языка при помощи добавления определённых компонентов (таких как дата, время и численный форматы), а также переводящегося текста.

Классы, которые обеспечивают интернационализацию Qt приложений: QTextCodec, QTextDecoder, QTextEncoder, QTranslator, QLocale.

Иногда интернационализация производится легко. Например, из американского английского в австралийский. Но иногда это сделать очень сложно, например, с корейского на немецкий, так как в данном случае программное обеспечение будет функционировать в разных языках, а также использовать разные техники ввода, разную кодировку символов и конвенции представления.

Qt пытается сделать интернационализацию как можно более безболезненной для разработчиков. Все средства ввода и рисования текста в qt предлагают встроенную поддержку для всех поддерживаемых языков. *В данной части перечислены все вложенные в qt языки.* Список данных языков поддерживается и будет работать на всех платформах, пока система содержит шрифты для данный систем написания.

*В целом в некоторых языках могут быть некоторые особенности написания различных символов. Qt освобождает разработчика от необходимости учитывать это, так как данная среда разработки сама управляет написанием символов из разных языков в своих нативных элементах управления. Есть, правда, два исключения, касающиеся функции QPainter:drawText() и написания собственных средств ввода, так как в некоторых языках ширина и форма символа зависят от того, какие символы его окружают. Поэтому проще бывает просто унаследовать некоторый встроенный класс ввода qt.*

Qt обеспечивает прекрасную поддержку для перевода С++ и Qt Quick приложений на локальные языки. В репозитории qttranslations находятся файлы переводов строк, которые содержатся внутри qt.

*Там есть ссылка на статью, посвящённую переводу qt на другие языки.*

Обычно Главная функция приложения будет включать следующее:

1. создание объекта QTranslator;
2. затем перевод загружается в соответствии с текущим местом расположения,
3. а объект переводчик устанавливается в приложение.

Отмечается, что при загрузке необходимого перевода разработчики должны запрашивать путь к переводам во время выполнения, передавая некоторой функции не переменную среды, а некоторую другую переменную (при помощи [QLibraryInfo::TranslationsPath](http://qt-project.org/doc/qt-5.1/qtcore/qlibraryinfo.html#LibraryLocation-enum)).

Класс QTextCodec и его средства в QTextStream делают простой поддержку многих кодировок для ваших данных пользователя. Когда приложение начинает работу, место действия машины определит 8-битную кодировку, используемую во время работы с внешними 8-битными данными. В данном классе есть функция, которая возвращает кодек, который может быть использован для преобразования между локальной кодировкой и кодировкой юникода. Внутри qt определены большинство данных кодеков. Если всё же в qt нет некоторой кодировки, то можно создать подкласс данного класса или обратиться к разработчикам qt.

Для локализации дат, времён, чисел и строк с валютами следует использовать класс QLocale.

Большинство картинок локализовать не рекомендуется.

Некоторые приложения способны во время работы изменять язык пользователя. Для обеспечения возникновения названий на новом языке на всех элементах управления следует переопределить функцию changeEvent(), чтобы она проверяла, не произошло ли событие LanguageChange. Если да, то затем следует обновить текст, который отображается на каждом виджете с использованием функции tr(). Всё это делается в коде данной функции. Все события, связанные с некоторыми изменениями должны заканчиваться вызовом этой функции по умолчанию. События об изменении языка будут отправлены, когда новый перевод будет установлен при помощи функции QCoreApplication::InstallTranslator().

На некоторых платформах существуют определённые ограничения для различного рода кодировок. Поэтому, в общем, qt приложения должны рассматриваться также с точки зрения ограничений, специфичных для данной платформы. *Затем перечислены некоторые особенности взаимодействия qt с различными платформами с точки зрения кодировок. Пока я не совсем понял данные особенности, но в будущем их нужно будет изучить.*

#### НАПИСАНИЕ ИСХОДНОГО КОДА ДЛЯ ПЕРЕВОДА

<http://qt-project.org/doc/qt-5.1/qtdoc/i18n-source-translation.html>

класс QString поддерживает кодировку Unicode. Все внутренние функции qt используют именно данный класс, но названия элементов управления могут быть не только QString, но и обычными С-строками.

Qt предполагает, что все С-строки по умолчанию написаны в кодировке UTF-8. Поэтому если у вас другая кодировка, то необходимо для нормального преобразования текста использовать специализированные классы.

Для того чтобы текст в коде был переводимым (заключённый в кавычки), необходимо, чтобы по отношению к нему использовалась функция QCoreApplication::translate(). Это легко реализовать, если для такого текста всегда вызывать функцию tr(). Также эта функция указывает в программе на те строки, которые должны быть переводимыми.

Если строка находится не в объекте QObject, то можно использовать QCoreApplication::translate() напрямую.

Каждому тексту, выделенному для перевода, qt выделяет определённый индекс, связанный с контекстом перевода, который обычно совпадает с именем класса QObject, в котором находится переводимый текст.

Функция tr() просматривает переводимый текст сквозь объект QTranslator. Но при этом на объект приложения должен быть установлен какой-либо из объектов-переводчиков (*видимо, в Главной функции приложения*). Что касается файлов QML, то там тоже самое, но использовать нужно функцию qsTr().

Для создания контекста следует вызвать макрос Q\_OBJECT, который переписывает контекст. **ЭТОТ МАКРОС НУЖНО ПЕРЕОПРЕДЕЛЯТЬ В КАЖДОМ ПОДКЛАССЕ, ЧТОБЫ ОН ПЕРЕПИСАЛ КОНТЕКСТ, ИНАЧЕ ОН БУДЕТ ОБРАЩАТЬСЯ К КОНТЕКСТУ РОДИТЕЛЬСКОГО КЛАССА**. Функция tr() будет обращаться именно к данному контексту. Когда lupdate работает, то он сохраняет текст в функции tr() в файле переводов в том контексте, к которому она относится.

*Раздел по локализации чисел понял плохо.*

Если необходимо обеспечить интернационализацию класса, который не является наследником Q\_OBJECT, то следует использовать макрос Q\_DECLARE\_TRACE\_FUNCTIONS(), или функцию QCoreApplication::translate() напрямую.

Комментарии переводчика

Разработчики могут сообщить информацию по каждой строке для перевода, чтобы облегчить работу переводчиков. Она извлекается, когда lupdate используется, чтобы обрабатывать файлы исходного кода. Рекомендуемый способ добавления комментариев – это комментировать перед функцией tr() в вашем коде. Комментарии должны быть в следующем виде:

//: … или /\*: … \*/

Тогда комментарии будут связаны со строкой, которая находится в функции tr().

Добавление мета-данных в строку. Дополнительные данные могут быть добавлены в каждое переводимое сообщение. Они добавляются при помощи специального комментария:

//= <id>

Это можно использовать для передачи сообщению уникального идентификатора, чтобы поддержать инструменты, которым он нужен. Альтернативный способ добавить мета-данные к сообщению – это использовать следующий синтаксис:

//~ <field name> <field contents>

Также можно использовать ключевое слово TRANSLATOR.

Вместе с префиксом «extra-» имя поля будет образовывать XML-элемент.

Многозначность. Если в рамках одного и того же контекста есть две абсолютно одинаковые строки, то в функции tr() можно помещать дополнительный строковый аргумент, который используется для того, чтобы отличить одинаковые строки.

Обработка множественных чисел. Некоторые строки содержат указатели места на различные целочисленные значения и их необходимо переводить по-разному в зависимости от того, какие значения имеют эти целочисленные элементы. С этой целью в функции tr() передаётся дополнительный аргумент.

также в данном примере видно, что если символ поместить в круглые скобки, то при переводе он становится полужирным.

*В данной части есть специальная ссылка на обработку слов, которые могут иметь как множественное, так и единственное число в зависимости от действий пользователя.*

Если вместо %n использовать %Ln, то при переводе будет отображаться локализованная версия данного числа.

Предположим, например, что вы желаете иметь переводимый текст в вашей программе, но вне функций. Для осуществления данной задачи используются два макроса QT\_TR\_NOOP() и QT\_TRANSLATE\_NOOP(). Они просто помечают текст, как тот, который должен выделять инструмент lupdate. Но макрос расширяет только текст, без контекста. *Полезные макросы для множественно повторяющихся строк.*

При этом следует помнить, что если вы делаете невозможным автоматическую конвертацию с массива строк в класс QString при помощи макроса QT\_NO\_CAST\_FROM\_ASCII, то вы можете неожиданно потерять некоторые строки.

Значения клавишей-ускорителей также должны быть переведены. Для этого используется некоторый класс QKeySequence. *Как это делается, показано в данной части.*

Использование перечисляемых аргументов. Это просто использование знакомой функции arg() класса QString. Если в разных языках изменяется порядок символов, то в строках переводов также следует изменять данный порядок.

### МЕТА-ОБЪЕКТНЫЙ КОМПИЛЯТОР

<http://qt-project.org/doc/qt-5.1/qtdoc/moc.html#moc>

это программа, которая обрабатывает расширения языка С++, предоставляемые qt.

Moc считывает заголовочные файлы С++, и если он находит один или более классов, объявления которых включают макрос Q\_OBJECT, то он производит С++ файл исходного кода, содержащий мета-объектный код для данных классов. Также данный код требуется для механизма сигналов и слотов, предоставления информации о типах во время выполнения и для системы динамических свойств.

Файл исходного кода, генерируемый moc, должен быть скомпилирован и связан с файлом реализации класса. qmake использует moc самостоятельно, так что нет необходимости работать с moc напрямую.

Также moc используется, если объявлены макросы: Q\_PROPERTY() и Q\_ENUMS(). Первый макрос объявляет свойства объекта, а второй – набор перечисляемых типов внутри класса, который используется внутри системы свойств. Q\_FLAGS() макрос объявляет перечисления, которые используются как флаги. Q\_CLASSINFO() позволяет вам добавлять дополнительные пары имя/значение в мета-объект класса.

Каждый раз как qmake фиксирует файл с некоторыми макросами, он вызывает moc для построения файла исходного мета-объектного кода, который называется moc\_myclass.cpp, который затем компилируется как обычно, преобразуясь в объектный файл. Этот файл затем следует включить в список объектных файлов, которые связываются вместе на конечной фазе построения программы.

Если в make-файл добавить некоторые инструкции, то во время построения он может публиковать некоторую выходную информацию о работе moc.

*Затем идёт несколько разделов, посвящённых ручному управлению moc. Но указывается, что лучше доверить данное управление qmake.*

Диагностика. Moc будет выдавать предупреждения вам о некоторых опасностях или неправильных конструкциях в объявлении класса Q\_OBJECT.

1. Нельзя пользоваться moc для шаблонов. Так что шаблон класса не может иметь сигналов и слотов.
2. При множественном наследовании требуется также, чтобы класс Q\_Object был на первом месте.
3. Указатели на функции не могут быть также сигналами или слотами.
4. При использовании функции connect() она сравнивает типы посимвольно, буквально, так что в аргументах сигналов и слотов следует указывать полные типы: MyClass:EnumerationType.
5. Вложенные классы не могут иметь сигналов и слотов.
6. Сигналы и слоты не должны возвращать ссылки, иначе они будут обработаны как void.
7. В секции сигналов и слотов класса могут помещаться только сигналы и слоты и никакие другие конструкты.

ЭТО ОЧЕНЬ ВАЖНАЯ ИНФОРМАЦИЯ, КОТОРУЮ СЛЕДУЕТ ЗАПОМНИТЬ.

### КОМПИЛЯТОР ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ (UIC)

<http://qt-project.org/doc/qt-5.1/qtdoc/uic.html>

uic считывает xml файл .ui, который создаётся при помощи дизайнера и преобразует его в заголовочный файл С++. Также указаны некоторые опции данного компилятора, если его использовать вручную.

### КОМПИЛЯТОР РЕСУРСОВ (RCC)

<http://qt-project.org/doc/qt-5.1/qtdoc/rcc.html>

rcc (компилятор ресурсов) используется для включения ресурсов в приложение Qt во время процесса построения. Он генерирует С++ файл исходного кода, содержащий данные, определённые в файле ресурсов .qrc. *Далее по тексту определено, как настраивать данный компилятор и как им управлять.*

### QT CREATOR

<http://qt-project.org/doc/qtcreator-2.8/index.html>

Это интегрированная кросс-платформенная среда разработки.

#### ОБЗОР

<http://qt-project.org/doc/qtcreator-2.8/creator-overview.html>

для построения приложения создателю необходима та же информация, что и компилятору. Эта информация устанавливается в настройках проекта.

Qt Creator обеспечивает два средства разработки графического интерфейса пользователя:

1. Qt Designer
2. QtQuick.

Первый используется для построения классического структурированного интерфейса для поддерживаемых платформ; второй используется для быстрого создания современного и интуитивного интерфейса пользователя. Qt Creator понимает код С++ и QML.

Qt Creator интегрирован с кросс-платформенными системами построения: qmake и CMake. Вы также можете полностью контролировать команды процесса построения.

Также Qt Creator интегрирован с различными внешними отладчиками:

1. GNU Symbolic Debugger (GDB),
2. Microsoft Console Debugger (CDB) и
3. internal Java-Script debugger.

Также Qt Creator интегрирован с Valgrind, чтобы тестировать производительность процессора и расход памяти.

Для мобильных платформ Qt Creator позволяет создавать специальные установщики для загрузки приложения в различные мобильные магазины.

#### МОДА ПРИВЕТСТВИЯ

<http://qt-project.org/doc/qtcreator-2.8/creator-quick-tour.html>

При загрузке создателя он переходит в моду приветствия, где вы можете:

* открыть предыдущие секции и проекты;
* создать новые проекты;
* открыть учебники и проекты-примеры;
* прочитать новости от сообщества qt или из онлайн лабораторий;
* осуществить обратную связь с командой разработчиков.

Там есть меню мод, выбор инструментария построения, отладки, непосредственно построение и поиск по проекту.

Qt Creator предоставляет следующие моды:

* мода приветствия для открытия проектов;
* мода редактирования кода;
* мода построения интерфейса;
* мода отладки для проверки состояния приложения во время отладки;
* мода проекта для настройки построения и выполнения проекта;
* мода анализа для проверки затрат памяти и профайла С++ или QML;
* мода помощи, где находится документация qt.

Также в создателе можно просматривать содержание проекта. Это осуществляется на боковой панели, которая доступна в модах редактирования и отладки. Там можно выбрать то, что отображает данная панель:

* файлы проектов текущей сессии;
* открытые в настоящее время файлы;
* просмотреть все закладки текущей сессии;
* файловая система показывает все файлы в выбранной директории;
* отображения классов показывает иерархию классов в выбранном проекте;
* аутлайн показывает иерархию символов в С++ и иерархию типов в QML;
* иерархия типов показывает базовые классы данного класса.

На боковой панели есть специальные средства для переключения панели, для разделения панели с целью отобразить некоторую другую информацию, а также для закрытия панели.

При просмотре проектов файлы можно сортировать и фильтровать, а также просматривать их абсолютные пути. *Как это делается, описано в данной части*.

Если при просмотре проектов не видно файла, то следует переключится к просмотру файловой системы.

Чтобы просмотреть базовые классы данного, нажмите правой кнопкой мыши на класс и выберите пункт «показать иерархию типов».

Область задач создателя может отображать одну из следующих задач:

* вопросы,
* результаты поиска,
* вывод приложения,
* вывод компилятора,
* консоль для QML/JS,
* to-do entries,
* контроль версий,
* общие сообщения.

Область вопросов обеспечивает следующие вопросы:

* анализатор – ошибки при работе системы анализа кода;
* ошибки и предупреждения во время построения;
* вывод компилятора;
* записи из файла задач (.tasks), который генерируется при помощи инструментов сканирования и анализа кода;
* также некоторые другие записи.

Там также есть фильтр, который позволяет выбрать подходящую информацию для вывода приложения.

Область задач поиска обеспечивает сохранение предыдущих поисков в истории поисков и возможность последующего выбора из них. Область вывода приложения отображает статус программы, когда она выполняется, а также вывод отладки. Вывод компилятора – это более детальная информация, чем та, которая отображается в области вопросов.

*Что делает to-do, я пока не понял. Он недоступен по умолчанию, но может загружаться как плагин*.

Также Qt Creator обеспечивает специальный менеджер для просмотра изображений из соответствующих файлов.

Также Qt Creator обеспечивает много клавиш для навигации по создателю.

*Мне следует затем детально изучить Qt Creator.*

#### НАСТРОЙКА СОЗДАТЕЛЯ

<http://qt-project.org/doc/qtcreator-2.8/creator-configuring.html>

если вы установили Qt Creator в составе пакета установки, то вы можете сразу его использовать. Но если вы установили его отдельно, то его необходимо настроить. При этом следует указать создателю пути к компиляторам и библиотекам qt, которые он будет использовать. Для использования создателя для разработки необходимо, чтобы была установлена библиотека qt и компилятор. Необходимо добавить в определённом меню настроек набор инструментов, библиотеку qt, а также компилятор.

Также можно добавлять к создателю некоторые устройства для тестирования приложения на них. Это делается в меню настроек «Устройства».

Также можно изменять в создателе схему цветов редактора кода. Это делается в меню настроек текстового редактора. Также при работе с кодом Qt Creator может давать некоторые подсказки о том, что следует ввести. Эти подсказки можно редактировать в специальном меню настроек текстового редактора: Code snippets.

Также можно настраивать систему контроля версий, что также делается в создателе в специальном меню настроек системы контроля версий.

Также Qt Creator предоставляет различные экспериментальные плагины, которые можно установить или сделать недоступными в зависимости от, того что вы делаете. Для этого следует обратиться по следующей ссылке: Help > About Plugins.

*В данном разделе есть много ссылок на более подробные настройки, которые затем следует изучить.*

#### ПОСТРОЕНИЕ И ЗАПУСК ПРИМЕРА

<http://qt-project.org/doc/qtcreator-2.8/creator-build-example-application.html>

Для проверки того, что Qt Creator установлен правильно, вам следует запустить на нём пример и протестировать его работу. *Как это делается, показано в данной части.*

<http://qt-project.org/doc/qtcreator-2.8/creator-tutorials.html>

вы можете создавать приложения для разных платформ при помощи разных базовых технологий. Далее они рассматриваются детальнее.

#### СОЗДАНИЕ ПРИЛОЖЕНИЯ QT QUICK

<http://qt-project.org/doc/qtcreator-2.8/creator-qml-application.html>

#### ДОБАВЛЕНИЕ НАБОРОВ ИНСТРУМЕНТОВ

<http://qt-project.org/doc/qtcreator-2.8/creator-targets.html>

Qt Creator группирует настройки построения и выполнения приложения как некоторые наборы инструментов. Каждый набор инструментов содержит некоторые значения, которые определяют некоторую среду, такую как устройство или компилятор, версию библиотеки, и команду отладчика, которая будет использована, а также некоторые метаданные, такие как иконка или имя Kitа. Однажды определив Kit, вы можете затем использовать его для построения и выполнения проекта.

*Далее идёт описание каждого пункта установки Kitа. Особенно хочется отметить последний пункт, в котором могут задаваться специальные команды для qmake. Если они не заданы, то используются команды по умолчанию. Затем отмечается, как загрузить отладчик. Для этого в целом необходимо выбрать тип отладчика и указать путь к его исполняемому файлу. Но при этом очень важно то, что это за исполняемый файл. Например, отладчики для Windows должны поддерживать сценарии питона. При выборе отладчика следует обращаться к данному разделу более детально.*

#### ДОБАВЛЕНЕИМ ВЕРСИЙ QT

<http://qt-project.org/doc/qtcreator-2.8/creator-project-qmake.html>

Qt Creator поддерживает множественные версии qt, которые можно устанавливать для построения проекта. Каждая версия устанавливается для конкретного Kitа. *В данной части показано, как установить нужную версию библиотеки для данного Kitа. Для этого стандартно необходимо название версии и путь.*

#### ДОБАВЛЕНИЕ КОМПИЛЯТОРА

<http://qt-project.org/doc/qtcreator-2.8/creator-tool-chains.html>

компилятор определяется также для конкретного Kitа. Для некоторых компиляторов нужно задавать директорию хранения некоторых файлов, а также пути к исполняемым файлам компилятора и имя компилятора.

*также там указан путь для добавления частного компилятора, который не перечислен в стандартном списке компиляторов для создателя в данной части.*

*Также тут указаны некоторые ошибки, которые могут возникнуть при работе с компилятором MinGW. Если в области задач при выводе компилятора отсутствуют слеши в путях файла, то следует написать в командной строке некоторые дополнительные команды, которые дадут некоторые дополнительные пути. Затем следует взять эти пути и добавить в переменную PATH.*

*Аналогичную процедуру для частных компиляторов я не понял.*

#### УПРАВЛЕНИЕ ПРОЕКТАМИ

<http://qt-project.org/doc/qtcreator-2.8/creator-project-managing.html>

отображены основные аспекты управления проектами, о которых далее речь пойдёт более подробнее.

* Создание проектов.
* Использование систем контроля версий.
* Конфигурирование проектов.
* Управление сессиями.

##### СОЗДАНИЕ ПРОЕКТОВ

<http://qt-project.org/doc/qtcreator-2.8/creator-project-creating.html>

создание проектов предоставляет вам возможность:

* группировать файлы вместе,
* добавлять частные шаги построения,
* включать формы и файлы исходного кода,
* определять настройки для работающих приложений.

Для создания проекта вас будет шаг за шагом сопровождать специальный визард.

Кроме использования qmake и CMake вы также можете использовать другие средства построения проектов, используя Qt Creator просто как редактор кода.

При создании проекта

1. вначале вы выбираете шаблон проекта.
2. Далее вы выбираете локацию проекта и
3. определяете его настройки.
4. После создания проекта визард создаёт все необходимые для выбранного вами проекта файлы.

Вы можете использовать визард для создания следующих типов приложений:

* приложения (
  + с графическим интерфейсом;
  + мобильные приложения;
  + приложения Qt Quick – использование встроенных типов для создания графических интерфейсов пользователя;
  + консольные приложения,
  + создание веб-приложений);
* библиотеки (
  + разделяемые или статические библиотеки С++,
  + плагин для дизайнера,
  + плагины для Quick);
* другие проекты (
  + модульные тесты,
  + частные виджеты для дизайнера, пустые проекты,
  + проекты поддиректории,
  + вставки кода – *не понял, что это такое. Видимо, некоторые дополнительные команды для построения*);
* не qt проекты (с++ проекты двух видов);
* импортировать проекты (
  + из системы контроля версии;
  + существующий проект).

Затем после выбора шаблона следуйте указаниям визарда. Обычно следует выбрать необходимый Kit.

Qt поддерживает OpenGL для отображения трёхмерной графики. Следует читать соответствующий модуль. GLSL – это специальный язык (OpenGL Shading Language), который используется для построения графики с использованием данной библиотеки. Вы можете писать файлы на данном языке прямо в создателе. Они есть в списке файлов для выбора при добавлении файла к проекту. *Затем, как я понял, в qt есть специальные классы, которые преобразуют данный класс в код С++*. *Но ранее я читал, что руководство по данной части библиотеки является устаревшим. Так что будем ждать нового.*

Чтобы отображать в qt другие файлы, добавьте в файле проекта некоторые значения переменной OTHER\_FILES:

Кроме библиотек qt вы можете добавить к вашему приложению другие библиотеки, например, системные или собственные. Чтобы скомпилировать ваш проект, вы можете добавить библиотеки в ваш проект. Процедура добавления библиотеки к приложению зависит от системы построения, которую вы используете. Вы можете добавить ваш проект как подпроект другого проекта, но при этом должен использоваться шаблон subdirs. Для определения зависимостей используйте специальный визард.

##### ОТКРЫТИЕ ПРОЕКТОВ

<http://qt-project.org/doc/qtcreator-2.8/creator-project-opening.html>

Qt Creator сохраняет информацию, которая ему необходима для построения проекта в файле .user. если Qt Creator не находит данного файла в проекте, который вы собираетесь открыть, то он просит вас ввести эту информацию самостоятельно. Если проект был создан в создателе на другой платформе, то вам будет предложено загрузить старые настройки. Лучше этого не делать, так как настройки зависят от платформы.

В меню настройки проекта вам следует выбрать соответствующий Kit. Qt Creator предложит вам названия для проекта. Вы можете их изменить. Если Qt Creator не находит построенный при помощи данного Kitа проект с заданным названием, то он предлагает создать новый проект. Если проект был построен ранее, то вы можете импортировать его с аналогичными настройками построения.

При создании нового проекта Qt Creator анализирует все файлы в существующем проекте, на основании чего строит новый проект.

##### ДОБАВЛНИЕ БИБЛИОТЕК В ПРОЕКТ

<http://qt-project.org/doc/qtcreator-2.8/creator-project-qmake-libraries.html>

в проект можно добавить другие библиотеки. Путь, по которому добавляется библиотека, зависит от типа и локации библиотеки. Вы можете добавить системную библиотеку, вашу собственную библиотеку или библиотеку третьей стороны. Как правило, к системным библиотекам не требуется указывать пути, так как они известны по умолчанию. Для ваших библиотек и библиотек третьих сторон вы должны указывать путь.

Для всех библиотек лучше указать целевую платформу, а также лучше определить, является библиотека статической или динамической. Для статически связанных внутренних библиотек Qt Creator добавляет зависимости (PRE\_TARGETDEPS в файле проекта, где указаны все внутренние библиотеки, от которых зависит приложение).

Также некоторые настройки библиотек зависят от платформы. Например, в Windows принято соглашение, согласно которому отладочная и выпускная версии библиотеки имеют одинаковые имена, но располагаются в разных директориях. Если их не указать, то у вас не будет возможности настроить то, чтобы библиотеки помещались в разные папки.

Есть альтернатива: к отладочной версии добавляется буква d.

*Я не понимаю, что имеется в виду под внутренней библиотекой.*

*также показателем пример кода в файле проекта при создании статической библиотеки. Там не просто указываются места локализации отладочной и выпускной версий для платформ Windows и Linux, но также есть и переменная* PRE\_TARGETDEPS, в которую помещается путь к библиотеке по отношению к директории OUT\_PWD.

#### ДОБАВЛЕНИЕ НОВЫХ ЧАСТНЫХ ВИЗАРДОВ

[*http://qt-project.org/doc/qtcreator-2.8/creator-project-wizards.html*](http://qt-project.org/doc/qtcreator-2.8/creator-project-wizards.html)

Если вы работаете в команде над большим приложением, то вы желаете создать некоторый стандартный способ, по которому команда будет создавать проекты и классы.

Частный визард определят интерфейс страниц визарда. На каждой странице есть поля ввода и их названия. Когда пользователь вводит туда значения, то они затем используются системой для построения. Построение может происходить двумя способами:

1. по шаблону, когда указатели в исходных файлах заменяются значениями в визарде.
2. сценарий генератора, когда пишется сценарий, который создаёт файлы. но это не рекомендуется, так как писать кросс-платформенные сценарии по существу очень сложно. Это оправдано, только если требуется сложная логика создания файлов проекта.

*Далее показаны директории, где хранятся частные визарды.*

Частные визарды имеют XML-файл, где находится его описание. Исходный файл шаблона визарда, а также некоторый скрипт.

Для создания визарда проекта следует:

1. сделать копию некоторой папки шаблона существующего визарда (лучше примера);
2. модифицировать XML-файл примера (там задаётся тип визарда, после какой страницы стандартного визарда добавляется данная страница, а также категория, к которой будет относится данный визард; далее указываются иконка и описание визарда; файлы, которые добавляются к проекту, причём данные файлы должны быть в папке частного визарда, откуда они просто копируются);
3. затем задаются опции сценария генератора;
4. затем следует код, который создаёт страницу визарда.

Затем приводится аналогичная процедура для создания визарда класса.

Для создания полей в визарде можно использовать следующие объекты: PathChooser, QCheckBox, QComboBox, QLineEdit, QTextEdit.

*Далее показывается, как при помощи XML-описания можно создавать все данные элементы.*

Шаблонные файлы исходного кода содержат указатели на место, которые затем заменяются тем, что вводит пользователь. Те места кода, которые заменяются введёнными значениями, обозначены как %FIELDNAME:%, причём с данным полем могут находится некоторые дополнительные символы, которые указывают на тип начертания. *Они показаны в данной части.*

Также в описании предусмотрены некоторые идентификаторы, обозначающие предопределённые константы. Также можно задать правила доступа для ввода пользователя. Также для создания частного визарда можно написать свой собственный скрипт.

#### ИСПОЛЬЗОВАНИЕ СИСТЕМ КОНТРОЛЯ ВЕРСИЙ

<http://qt-project.org/doc/qtcreator-2.8/creator-version-control.html>

в данной части перечислены системы контроля версий, которые поддерживаются создателем. Меркуриал присутствует в данном списке.

Qt Creator использует клиентов командной строки систем контроля для доступа к репозиториям. Поэтому данные клиенты должны быть указаны в переменной среды PATH.

Qt Creator позволяет вам создавать репозитории для проекта, если система контроля версий поддерживает локальные репозитории. Это делается на последней странице визарда при создании проекта. Также можно загрузить проект из некоторой системы контроля версий.

Меню tools имеет подменю для каждой системы контроля версий.

Также есть меню (*указано в данной части*), которое открывает окно, в котором отображаются команды, которые выполняются, временные метки а также вывод.

Если вы создаёте объект или файл в директории, которая находится под действием системы контроля версий, то вы будете запрошены на счёт того, следует ли добавить файл в систему контроля версий. Иначе можно затем добавить данные файлы и директории в систему контроля версий при помощи инструментов самой системы.

В создателе можно просматривать в режиме только чтения результат выполнения команды diff: сравнение изменений в файле по сравнению с предыдущей версией.

Также можно просматривать историю изменений в системе контроля версий при помощи создателя. *Также можно просматривать какую-то аннотацию.* Также из создателя можно закоммитить или засабмитить изменения. Также можно и отменять изменения. Но это зависит от возможностей самой системы контроля версий. *Также можно просматривать какой-то статус проекта или репозитории.* Также можно обновлять рабочее дерево насчёт последних изменений. Также можно удалять файлы из репозитория. Также можно использовать дополнительную команду, которая помечает файл для фиксации в репозитории. Также можно очистить репозиторий. *Также можно применять какие-то файлы патчей, в которых описаны инструкции для воздействия на некоторую группу файлов*. Также можно применять различные действия к фиксациям.

Что касается меркуриала, то здесь есть несколько дополнительных возможностей для данной системы контроля версий. *Все они описаны в конце данной части.*

#### КОНФИГУРИРОВАНИЕ ПРОЕКТА

<http://qt-project.org/doc/qtcreator-2.8/creator-configuring-projects.html>

для того чтобы изменить настройки для данного проекта, следует перейти в моду проекта. В настройках можно добавлять, изменять и удалять Kits.

Страница проекта состоит из следующих табов:

* построение и запуск (настройки построения и настройки запуска),
* настройки редактора,
* настройки стиля кода,
* зависимости.

Также, если открыто несколько проектов, можно использовать специальные переключатели для навигации в настройках различных проектов.

<http://qt-project.org/doc/qtcreator-2.8/creator-targets.html>

*уже рассматривалось*

[*http://qt-project.org/doc/qtcreator-2.8/creator-project-qmake.html*](http://qt-project.org/doc/qtcreator-2.8/creator-project-qmake.html)

*уже рассматривалось*

[*http://qt-project.org/doc/qtcreator-2.8/creator-tool-chains.html*](http://qt-project.org/doc/qtcreator-2.8/creator-tool-chains.html)

*уже рассматривалось*

##### ОПРЕДЕЛЕНИЕ НАСТРОЕК ПОСТРОЕНИЯ

[*http://qt-project.org/doc/qtcreator-2.8/creator-build-settings.html*](http://qt-project.org/doc/qtcreator-2.8/creator-build-settings.html)

в общем, конфигурация отладки используется для тестирования, а конфигурация выпуска для выпуска конечной версии продукта.

По умолчанию Qt Creator производит построение проекта в конфигурациях построения для выпуска и отладки. Построение отладки имеет несколько дополнительных символов, которые используются для отладки, а в построении выпуска данные символы опускаются. В общем, построение отладки используется для тестирования, а построение выпуска – для создания конечного файла установки.

По умолчанию Qt Creator строит проекты в отдельной директории от исходной директории, что называется теневой сборкой. Это позволяет разделять файлы, сгенерированные для каждого Kit построения и выполнения. Если же вы используете только один Kit, то это свойство можно изменить.

Qt Creator использует внешние процессы для выполнения задач, таких как построение или запуск приложений. Для выполнения процессов Qt Creator использует команды шела, которые являются родными для системы.

Вы можете задать различные аргументы. Также можно создавать команды шела для перенаправления и других сложных конструкций.

Также можно задавать значения для любых переменных среды. *Как это делается, показано в данной части.*

В разделе шагов построения можно изменять настройки для сборщика. Также можно задавать частные шаги построения.

Также можно использовать процесс очистки для удаления промежуточных файлов. *Как это делается, показано в данной части.*

Также можно проверять и настраивать среду построения, добавляя новые переменные среды или изменяя старые.

Следует отметить, что изменения, внесённые вами в свойства среды, сохраняются в локальном специфичном для данного проекта файле: .pro.user. поэтому они не подходят для разделения между разработчиками или их компьютерами. Для того чтобы разделять настройки, их следует помещать в файл проекта.

Также можно выполнить построение для очищенной среды, *но что это такое, я так и не понял.*

##### ОПРЕДЕЛЕНИЕ НАСТРОЕК ВЫПОЛНЕНИЯ

<http://qt-project.org/doc/qtcreator-2.8/creator-run-settings.html>

Qt Creator автоматически создаёт настройки для конфигурации запуска приложения. Настройки можно менять. Они зависят от типа проекта и типа Kitа. Конфигурация для qmake производит свои исполняемые файлы из проанализированного файла проекта. Также можно определять аргументы командной строки, которые передаются исполняемому файлу и рабочей директории для использования.

Qt Creator интегрирован с Valgrind анализатором кода для детектирования расхода памяти и профайлинга выполнения функций. Настройки анализатора можно задать глобально для всех проектов или отдельно для каждого проекта.

Также можно задавать настройки отладчика. *Можно задавать порт отладчика, но что это такое, я пока не понял. Однако там по тексту есть одно замечание, согласно которому открытие сокетов в хорошо известных портах представляет определённый риск для безопасности. Любой из интернета может подключиться к приложению, которое вы отлаживаете и выполнить любую функцию JavaScript. Поэтому вы должны быть уверены, что порт качественно защищён при помощи брандмауэра. Что всё это значит???*

Если вы попытаетесь отлаживать одновременно разные приложения при помощи одного и того же порта, то вам будет выдано сообщение об ошибке. Поэтому следует выбрать свободный порт для одного из приложений.

Также можно редактировать среду выполнения или выбрать другую среду выполнения.

Также можно создать частную конфигурацию запуска для вашего проекта. Например, когда вы создаёте библиотеку, вы можете запустить тестовое приложение, которое связывается с данной библиотекой.

*Также можно добавлять некоторые особенные настройки для проектов Qt Quick.*

#### ОПРЕДЕЛЕНИЕ НАСТРОЕК РЕДАКТОРА

<http://qt-project.org/doc/qtcreator-2.8/creator-editor-settings.html>

Qt Creator использует тип MIME файла, чтобы определить, какая мода и редактор должны быть использованы для открытия файла. Например, Qt Creator открывает .txt в моде редактирования.

#### ОПРЕДЕЛЕНИЕ НАСТРОЕК СТИЛЯ КОДА

<http://qt-project.org/doc/qtcreator-2.8/creator-code-style-settings.html>

Вы можете настраивать стиль кода для ваших нужд. Вы можете настраивать его для всех проектов либо для данного конкретного проекта. Вы также можете определить несколько наборов стилей кода и просто переключаться между ними.

*В данной части показано, как изменять стиль кода глобально, для всех проектов, а как это делать локально, для данного конкретного проекта.*

#### ОПРЕДЕЛЕНИЕ ЗАВИСИМОСТЕЙ

<http://qt-project.org/doc/qtcreator-2.8/creator-build-dependencies.html>

если вы работаете с несколькими проектами, то вы можете задавать порядок, в котором они будут строиться. Заметьте, что данное свойство касается именно сессии, а не проекта. Также таким образом нельзя строить под-проекты к проектам.

#### РАЗДЕЛЕНИЕ НАСТРОЕК ПРОЕКТА

<http://qt-project.org/doc/qtcreator-2.8/creator-sharing-project-settings.html>

Qt Creator сохраняет определённые пользователем настройки проекта в файле .pro.user. Вы можете разделять эти настройки между несколькими проектами как .pro.shared файл. Он имеет такую же структуру, как и XML-файл настроек пользователя, но дополнительно содержит также некоторые настройки разделения.

#### УПРАВЛЕНИЕ СЕССИЯМИ

<http://qt-project.org/doc/qtcreator-2.8/creator-project-managing-sessions.html>

Сессия – это произвольный набор: открытых проектов с их зависимостями, открытых редакторов, выражений и точек останова, закладок. Сессия является персональной, что значит, что она не разделяется. Она не предполагает отражение структуры проекта. Она содержит персональные данные, такие как закладки и точки останова, которые обычно не являются сферами интереса других разработчиков.

Когда вы выходите из создателя, то снимок вашего текущего рабочего пространства сохраняется как сессия. Чтобы перезаписать сессию автоматически, когда вы запускаете Qt Creator, выберите **File > Session Manager > Restore last session on startup.**

Например, вы работаете над одним проектом, и вам стало очень необходимо поработать над другим проектом. Тогда вы сохраняете его как отдельную сессию и затем возвращаетесь в него точно также, как вы его оставили, когда переходили в другой проект.

*Для этого из меню файла вызывается менеджер сессий. Как это делается, показано в данной части.* Чтобы переключиться между сессиями, используйте всё тот же менеджер. Qt Creator всегда создаёт сессию по умолчанию, сохраняя то, что было в нём, когда вы закрыли его в последний раз.

Начиная Qt Creator из командной строки, вы можете задать название сессии как аргумент и Qt Creator начнётся с данной сессии.

#### РАЗРАБОТКА ИНТЕРФЕЙСОВ ПОЛЬЗОВАТЕЛЕЙ

<http://qt-project.org/doc/qtcreator-2.8/creator-design-mode.html>

для разработки графических интерфейсов пользователя Qt Creator обеспечивает два интегрированных визуальных редактора:

* Qt Quick r и
* Qt Designe.

Интеграция включает

* управление проектом и
* завершение кода.

##### РАЗРАБОТКА QT QUICK ПРИЛОЖЕНИЙ

<http://qt-project.org/doc/qtcreator-2.8/creator-visual-editor.html>

Для разработки приложений Qt Quick можно использовать две моды:

* мода редактирования и
* мода разработки.

Обычно разработка приложения состоит из следующих стадий:

* создание или импортирование проекта;
* создание компонентов;
* создание экранов;
* добавление на экраны анимации;
* добавление методов взаимодействия пользователей;
* реализация логики приложения.

###### СОЗДАНИЕ ПРОЕКТОВ В QT QUICK

<http://qt-project.org/doc/qtcreator-2.8/quick-projects.html>

приложение QtQuick строится на основе языков C++ и QML. Проект включает класс QDeclarativeView. Вы можете выбрать шаблоны, которые затем использовать либо как встроенные типы QML, либо как компоненты Qt Quick.

При создании графического проекта QtQuick Qt Creator создаст следующие файлы:

* .qmlproject (определяет, что все файлы в папке проекта принадлежат к проекту),
* .qml (определяет элемент интерфейса пользователя, такой как экран или компонент, или целое приложение интерфейса пользователя).

Утверждение вначале файла .qml определяет модули qt, которые следует импортировать. Каждый модуль библиотеки содержит набор QML-типов по умолчанию. Для того чтобы получить те особенности, которые вам нужны, вам следует определить версию. Чтобы использовать файлы JavaScript и изображений, перекопируйте их в папку проекта.

###### ИСПОЛЬЗОВАНИЕ QT QUICK ДИЗАЙНЕРА

<http://qt-project.org/doc/qtcreator-2.8/creator-using-qt-quick-designer.html>

.qml файл можно редактировать либо в дизайнере Qt Quick, либо в редакторе кода.

В самом дизайнере есть несколько панелей:

* навигатор (показывает элементы .qml файла как древовидную структуру), библиотека (предопределённые типы qml, ваши собственные конструкты qml и импортированные элементы),
* холст,
* редактор свойств и состояний, в которых может пребывать данный элемент.

Навигатор отображает элементы в текущем .qml и их взаимоотношения. Также следует отметить, что дочерние элементы не обязательно будут находиться внутри родительских элементов. Во время работы с холстом можно разные элементы прятать и показывать для удобства, а также управлять их видимостью, делая их прозрачными и непрозрачными.

Значение z показывает положение элемента по сравнению с соседями одинакового с ним уровня. Данное значение можно изменять.

Если вы перетаскиваете дочерний элемент на другое место, то его родитель не меняется автоматически. Для этого удерживайте клавишу shift или же поменяйте его в навигаторе.

Свойства и библиотека сгруппированы тематически. Значения свойств по умолчанию имеют белый цвет, а значения, изменённые пользователем, голубой цвет. Также вместо конкретного значения свойства можно задавать выражение: код JavaScript, который связывает значение данного свойства со значением другого свойства. Также для каждого значения можно отмечать, что оно должно переводиться.

QtQuick поддерживает модели, представления и делегаты, так что если вы задаёте некоторое представление, то его модель и делегат добавляются автоматически.

*Также, насколько я понял, в QtQuick возможна утрата контекста, но что конкретно имеется в виду, мне пока непонятно.*

Также в QtQuick введён так называемый манекен модели. Дело в том, что если Qt Quick ссылается на модель С++, то он ничего не отобразит. Для того, чтобы он отобразил, следует в директории проекта создать специальную директорию, в которую поместить .qml файл с именем, таким же, как у модели, а также создать файл со специальным названием (файл-манекен).

*Фактически манекен нужен для того, чтобы определить некоторые данные, которых в принципе не предполагается, но без которых QtQuick Qt Designer не будет отображать необходимые элементы.*

Для позиционирования в QtQuick предусмотрены так называемые якоря, которые определяют положение границ элемента и его центра. Якорями можно связывать только элемент с его ближайшим родителем, а также с равноправными в иерархии элементами. Базовый якорь не задаётся в дизайнере, но задаётся в редакторе кода. Что касается отношения родитель-ребёнок, то якоря связывают только одноимённые стороны, а для отношения равноправных элементов якоря связывают противоположные стороны, чтобы держать элементы вместе. Также только для якорей можно ввести поля.

Также для каждого элемента можно обеспечивать трансформации:

* вращение,
* масштабирование,
* трансляция.

Интерфейс создаётся с целью того, чтобы модифицироваться при взаимодействии с ним пользователя. Поэтому необходимо определять для интерфейса также и некоторые его внутренние состояния. Каждый элемент в qml может иметь различные состояния, при переходе между которыми изменяются свойства элементов. Каждое состояние может обеспечивать некоторые возможности, среди которых можно выделить следующие:

* показывать одни элементы и прятать другие,
* предоставлять пользователю различные доступные действия;
* начинать, останавливать или приостанавливать анимацию;
* выполнять некоторый скрипт, требуемый в новом состоянии;
* изменять значения свойства для конкретного элемента;
* показывать различные представления или экраны.

QtQuick предоставляет всю полноту возможностей редактирования состояний.

Позиция элемента на холсте может быть как абсолютной, так и относительной, по отношению к другим элементам. Вы можете задавать абсолютные координаты элемента на холсте или использовать якоря для задания его относительного положения.

В дизайнере QtQuick есть возможность убирать границы элемента, *что показано в данной части.*

*Также при внесениях изменений следует переустанавливать каждый раз холст, так как он может отображать неправильно элементы.*

###### СОЗДАНИЕ КОМПОНЕНТОВ

<http://qt-project.org/doc/qtcreator-2.8/quick-components.html>

компонент обеспечивает способ определения новых визуальных элементов, которые затем можно повторно использовать в других qml-файлах. Компонент как чёрный ящик: он взаимодействует с окружающим миром при помощи свойств, сигналов и слотов, а сам описан в .qml файле.

Можно использовать следующие типы qml:

* изображение с границами (для фона или границы),
* изображение (битовый рисунок),
* элемент (наиболее базовый элемент из всех визуальных элементов, он не имеет изображения, но описывает наиболее общие свойства, которые имеют другие визуальные элементы),
* прямоугольник,
* текст,
* редактор текста,
* ввод текста,
* веб-представление.

Также одни компоненты могут состоять из других компонентов, и это можно просматривать во время работы.

<http://qt-project.org/doc/qtcreator-2.8/quick-buttons.html>

*показано, как создать кнопку.*

[*http://qt-project.org/doc/qtcreator-2.8/quick-scalable-image.html*](http://qt-project.org/doc/qtcreator-2.8/quick-scalable-image.html)

*показано, как при помощи объекта изображения с рамкой создать собственную кнопку.*

<http://qt-project.org/doc/qtcreator-2.8/quick-screens.html>

*по данному разделу мне пока читать нечего. Лучше читать необходимые вещи, а это отдельно выучить после.*

<http://qt-project.org/doc/qtcreator-2.8/quick-animations.html>

<http://qt-project.org/doc/qtcreator-2.8/quick-user-interaction.html>

<http://qt-project.org/doc/qtcreator-2.8/quick-export-to-qml.html>

<http://qt-project.org/doc/qtcreator-2.8/quick-application-logic.html>

<http://qt-project.org/doc/qtcreator-2.8/creator-qml-modules-with-plugins.html>

##### РАЗРАБОТКА ОСНОВАННЫХ НА ВИДЖЕТАХ ПРИЛОЖЕНИЙ

<http://qt-project.org/doc/qtcreator-2.8/creator-using-qt-designer.html>

Qt Creator автоматически использует Qt Designer для открытия всех .ui файлов при создании приложений, основанных на виджетах. *Затем идёт речь о возможностях дизайнера. Многое из этого я уже видел. Особое внимание следует обратить на самый последний раздел на данной странице, в котором описаны некие обёртки устройства – набор файлов конфигурации, которые описывают мобильное устройство.*

###### ДОБАВЛЕНИЕ ПЛАГИНОВ В QT DESIGNER

<http://qt-project.org/doc/qtcreator-2.8/adding-plugins.html>

вы можете добавлять в Qt Designer свои собственные виджеты. Наиболее гибкий способ включения плагина в приложение – это скомпилировать его в динамическую библиотеку, которая помещается отдельно и детектируется, после чего загружается во время выполнения. Приложения могут распознать плагины, которые помещаются в стандартные директории плагинов.

*Далее идёт речь о том, в каких директориях помещать плагины для дизайнера: в случае, когда он загружается отдельно, и в случае, когда он загружается как часть SDK.*

<http://qt-project.org/doc/qtcreator-2.8/creator-usability.html>

*пропускаем, так как это касается мобильных устройств.*

#### КОДИРОВАНИЕ

[*http://qt-project.org/doc/qtcreator-2.8/creator-coding.html*](http://qt-project.org/doc/qtcreator-2.8/creator-coding.html)

Написание, редактирование и навигация в коде – одна из центральных задач написания приложения. Поэтому редактор кода занимает ключевое положение в создателе. Qt Creator обеспечивает различные методы поиска. Qt Creator предоставляет некоторые возможности рефакторинга. Также вы можете настраивать редактор кода на своё усмотрение.

<http://qt-project.org/doc/qtcreator-2.8/creator-editor-functions.html>

**НАПИСАНИЕ КОДА**

для написания кода редактор кода в создателе обеспечен следующими возможностями:

* семантическое выделение,
* проверка синтаксиса,
* завершение кода,
* отступы кода;
* справка, чувствительная к содержанию;
* индикаторы ошибок.

При работе в моде редактирования вы можете использовать панель инструментов редактора для навигации между открытыми файлами и символами. Вы можете также разделить экран для работы на несколько, чтобы просматривать несколько отдельных файлов одновременно, добавлять закладки и передвигаться между символьными определениями и объявлениями.

Кроме прочих важных особенностей вы можете также выбирать и высылать другим коллегам по команде те фрагменты кода, которые например, вы планируете зафиксировать в системе контроля версий.

Также в коде вы можете задавать различного рода макросы, которые будут в коде заменяться той символьной последовательностью, которую они представляют.

##### РАБОТА В МОДЕ РЕДАКТИРОВАНИЯ

<http://qt-project.org/doc/qtcreator-2.8/creator-coding-navigating.html>

Все панели инструментов в моде редактирования расположены вверху поля редактора кода. Панель управления редактора чувствительна к содержанию и показывает элементы, уместные для открытого в настоящее время файла в редакторе. Есть кнопки для продвижения про просматриваемым файлам назад и вперёд.

Сверху редактора есть два меню:

* Меню символов
* Меню файлов.

Первое позволяет выбирать символ в файле, к которому следует перейти. Второе – выбирать файл из тех, что открыты. *Открытым файлом, как я понял, считается только тот, который был просмотрен в редакторе кода*. Элементы в меню элементов можно сортировать по алфавиту.

Для перехода на нужную строку следует нажать Ctrl-L и ввести номер строки и столбца в формате «строка-столбец».

*Также там есть некоторая опция по file-encoding, которую я не нашёл в нашем создателе. Видимо это касается более поздней версии.*

*Также в данной части показано, как разделять редактор кода для просмотра нескольких файлов и как просматривать их в новом окне.*

*Также здесь показано несколько способов установки закладки на некоторую строку. Мне наиболее понравился способ использования сочетания клавиш Ctrl-M. Также там сказано, что к закладкам можно приписывать некоторые комментарии, но в нашем создателе это не работает. Также написано сочетание клавиш для перехода от закладки к закладке, но это сочетание также не работает на нашем создателе.*

также Qt Creator предоставляет возможность перемещаться напрямую между объявлением и определением некоторого символа. Наиболее удобный способ сделать это – клавиша F2. Также можно отключить данную возможность. *Как это сделать, показано в данном документе.*

Также ссылки на определение или объявление по умолчанию показываются в той же области редактора. Это также можно выключить. *Как это делается, показано в данной части*. *К слову, в нашем создателе такой функции я не нашёл. Также там сказано об ручном обновлении модели кода, которое следует использовать в аварийных ситуациях.*

|  |  |
| --- | --- |
| **Клавиша** | **Описание действия** |
| Ctrl + L + ввод номера строки и столбца | Переход к определённому месту кода |
| Ctrl + M | Установка закладки |
| Ctrl + , | Переход к предыдущей закладке |
| Ctrl + . | Переход к следующей закладке |
| F2 | Переход между объявлением и определением файлов |
| Ctrl + space | Высвечиваются и фиксируются возможные завершения кода |
| Выделить блок кода + Tab | На блок подействует отступ |
| Выделить блок кода + Shift + Tab | Блок будет отменён |
| Alt + ( | Начать написание макроса в редакторе кода |
| Alt + ) | Закончить написание макроса в редакторе кода |
| Alt + R | Проиграть последний макрос |
| Ctrl + K | Активировать локатор |
| Ctrl + Shift + U | Найти использование данного симлова |
| Alt + Enter | Открывается контекстное меню для возможных действий рефакторинга, которые можно применить в данном месте кода |

##### СЕМАНТИЧЕСКОЕ ВЫДЕЛЕНИЕ

<http://qt-project.org/doc/qtcreator-2.8/creator-highlighting.html>

редактор кода отличает типы, локальные переменные, поля класса, виртуальные методы. *Здесь показано, как определить схему цветов для выделения кода.*

Можно загружать файлы определения для выделения кода, которые использует Qt Creator. *Как это конкретно делать, я пока не совсем понял, но в будущем я разберусь в этом.*

Также можно выделять блоки. Например, если вы помещаете курсор в код под блоком, то данный блок выделяется. *Очень полезная особенность. В данной части показано, как это установить.*

##### ПРОВЕРКА СИНТАКСИСА КОДА

<http://qt-project.org/doc/qtcreator-2.8/creator-checking-code-syntax.html>

синтаксические ошибки подчёркиваются красным, а семантические ошибки и предупреждения подчёркиваются оливковым оттенком.

*Затем там есть огромная Глава, которая посвящена JavaScript и QML. Там, видимо, собраны все возможные ошибки, которые могут возникнуть в данном случае.*

##### ЗАВЕРШЕНИЕ КОДА

<http://qt-project.org/doc/qtcreator-2.8/creator-completing-code.html>

когда вы пишете код, то Qt Creator предлагает свойства, скобки и идентификаторы, чтобы завершить код. При нажатии на клавишу Ctrl+space возможные предложения по завершению кода от создателя высветятся и зафиксируются.

*В данной части приведён список значков для каждого элемента кода.*

Фрагменты кода для вставки можно редактировать. Это делается в свойствах редактора кода. Вы можете задавать свои собственные вставки:

* вставки кода С++,
* вставки кода QML,
* вставки простых строк, например, комментариев.

*Это грандиозно!!! Я сейчас попробовал на своём коде. Всё предельно просто. Выбираешь в окне редактирования фрагментов создать новый фрагмент. В первом столбце пишешь фрагмент кода, который следует замещать (например, QLab), справа пишешь название того, чем ты будешь замещать. А внизу в специальной области ты пишешь код, который будет вставлен. Также можно очень быстро получать все тестовые метки и так далее и тому подобное.*

Редактор фрагментов обеспечивает вас возможностью выделения отступов, скобок соответствия, базового замещения кода. *Если в коде для замещения вместо некоторого имени использовать такую запись $name$, то при вставке фрагмента кода затем можно заменить данное имя, и оно автоматически изменяется во всех местах вставленного кода.*

Также есть возможность удалить все встроенные фрагменты для завершения, если они вам не очень нужны и переустановить их позднее, вернув назад все удалённые.

##### ОТСТУПЫ ТЕКСТА ИЛИ КОДА

<http://qt-project.org/doc/qtcreator-2.8/creator-indenting-code.html>

если выделить блок кода, а затем нажать tab, то код перейдёт на отступ вперёд, а если shift+tab, то один отступ будет отменён. Также можно отключить автоматические отступы, которые предоставляет Qt Creator.

Также можно задать несколько стилей написания кода и переключаться между ними, так что для каждого проекта будет свой собственный стиль написания кода.

Для установки настроек отступов для С++ редактора кода, *следуйте инструкциям, отображённым в данной части*. Для того чтобы определить отступы для редактора кода С++,

* следует зайти в инструменты-настройки-С++,
* скопировать базовый стиль отступов,
* присвоив ему уникальное имя.

Для определения разных настроек для разных проектов следует обратиться к меню Projects-Code Style Settings.

*Методом проб и ошибок я обнаружил, что новому стилю будет соответствовать только новый код, а старый код останется в старом стиле.*

*Затем в данной части написано про отступы в QML.*

для определения отступов в текстовых файлах другого назначения выберите **Tools > Options > Text Editor > Behavior**. Для определения других установок для конкретного проекта выберите **Projects** > **Editor**.

При настройке отступов вы можете определить, совершать ли их только пробелами или также и табуляциями, а также определить, чтобы они совершались смешанно. Также можно изменить величину отступа и табуляции. Можно задавать или отменять автоматическое выравнивание линий. Также можно задать уменьшение отступа при нажатии клавиши backspace.

Также в настройках можно определять, когда совершается отступ, где располагать фигурные скобки и т.д.

<http://qt-project.org/doc/qtcreator-2.8/qt-quick-toolbars.html>

тут про QtQuick. Это пока тоже мне не очень интересно.

<http://qt-project.org/doc/qtcreator-2.8/creator-editor-codepasting.html>

*чему посвящена данная часть, я плохо понял. Речь идёт о получении и загрузке кода с серверов и на сервера. В данной части перечислены серверы с кодом (но непонятно, это интернет серверы или это средство для локальных серверов).*

##### ИСПОЛЬЗОВАНИЕ МАКРОСОВ РЕДАКТИРОВАНИЯ ТЕКСТА

[*http://qt-project.org/doc/qtcreator-2.8/creator-macros.html*](http://qt-project.org/doc/qtcreator-2.8/creator-macros.html)

*Для лучшего понимания данной возможности её следует опробировать.*

##### СРАВНЕНИЕ ФАЙЛОВ

[*http://qt-project.org/doc/qtcreator-2.8/creator-diff-editor.html*](http://qt-project.org/doc/qtcreator-2.8/creator-diff-editor.html)

вы можете использовать diff редактор, чтобы проверить отличия в двух файлах. *В данной части указано, как загрузить данный редактор. В данной части описаны особенности использования данного средства.*

##### ПОИСК

[*http://qt-project.org/doc/qtcreator-2.8/creator-finding-overview.html*](http://qt-project.org/doc/qtcreator-2.8/creator-finding-overview.html)

Qt Creator предоставляет два типа поиска:

* инкрементный
* продвинутый.

Инкрементный поиск – это последовательное выделение совпадающих символов в коде в открытом в настоящее время файле. Продвинутый поиск – это поиск среди всех файлов открытого проекта, которые находятся в файловой системе.

Также в создателе есть специальная служба лёгкого поиска, называемая локатором.

<http://qt-project.org/doc/qtcreator-2.8/creator-editor-finding.html>

*в данной части описано, как осуществлять инкрементный поиск.*

*Также в данной части описано, как осуществлять продвинутый поиск.*

##### ПОИСК ПРИ ПОМОЩИ ЛОКАТОРА

<http://qt-project.org/doc/qtcreator-2.8/creator-editor-locator.html>

локатор находится слева внизу в окне создателя.

В локаторе можно задавать имя файла, после чего данный файл открывается, а также задавать номер строки и столбца, после чего курсор переходит на данную строку и столбец. Также в локаторе можно использовать различные фильтры, которые задаются некоторым специальным фильтром.

Также в данной части описывается, как создавать свои собственные фильтры локатора. Что касается информации, которую ищет локатор, то она кэшируется. Например, Qt Creator очищает кэш созданных вами фильтров раз в час. Чтобы обновить кэш вручную, следует зайти в настройки локатора и нажать команду **Refresh**.

##### РЕФАКТОРИНГ

<http://qt-project.org/doc/qtcreator-2.8/creator-editor-refactoring.html>

выполняйте рефакторинг для того, чтобы:

* улучшить внутреннее качество ваших приложений,
* улучшить расширяемость и производительность,
* улучшить читабельность и устойчивость кода,
* упростить структуру кода.

*показано, как переименовать символ, что я уже успешно умею делать. Также там перечислены возможные действия, которые позволяет делать Qt Creator с кодом С++. Там очень много полезных приёмов рефакторинга, многие из которых я не использовал. К традиционным можно отнести создание определения/объявления. А там можно совершать несколько десятков приёмов рефакторинга.*

1. *Можно переименовывать символ.*
2. *Можно изменить бинарные операции.*
3. *Упрощение условий if и while.*
4. *Редактирование строк*
5. *Создание объявления переменных*
6. *Создание определения и объявления методов*

##### КОНФИГУРИРОВАНИЕ РЕДАКТОРА

[*http://qt-project.org/doc/qtcreator-2.8/creator-editor-options.html*](http://qt-project.org/doc/qtcreator-2.8/creator-editor-options.html)

В данной части обобщены все возможные операции по редактированию кода

* Определение цветовых схем и выделений для кода
* Определение цветовых схем и выделений для других типов файлов
* Установка табуляции, отступов и поведения пробелов
* Добавление, изменение и удаление вставок кода
* Добавление, изменение и удаление макросов
* Настройка завершения кода

##### ВСТАВКА ФРАГМЕНТОВ КОДА

[*http://qt-project.org/doc/qtcreator-2.8/creator-completing-code.html#editing-code-snippets*](http://qt-project.org/doc/qtcreator-2.8/creator-completing-code.html#editing-code-snippets)

Вставки определяют некоторые конструкты С++.

##### ИЗМЕНЕНИЕ ЦВЕТОВОЙ СХЕМЫ

[*http://qt-project.org/doc/qtcreator-2.8/creator-editor-options-text.html*](http://qt-project.org/doc/qtcreator-2.8/creator-editor-options-text.html)

*показано, как изменять шрифт и цветовую схему в создателе.* Что касается кодирования файлов, то Qt Creator использует по умолчанию кодировку UTF-8.

Чтобы перерешать некоторый элемент, используйте инструмент конвертации файлов, чтобы преобразовать кодировку файла к UTF-8, когда разработка ведётся на Qt 5. Иначе преобразование строковых констант в класс QString не сможет работать так, как ожидалось. *Не совсем понял данный пункт. В будущем мне следует уточнить данную информацию.*

Если вы разрабатываете только приложения Qt4, вы можете установить другие настройки кодировки, отличные от кодировки по умолчанию. Выберите System настройку, чтобы использовать кодировку файла, используемую вашей системой.

##### FakeVim

<http://qt-project.org/doc/qtcreator-2.8/creator-editor-fakevim.html>

в моде FakeVim вы можете запустить Главный редактор способом, подобным на редактор Vim. *Пока не понимаю, что это такое, но догадываюсь, что речь идёт о некотором другом текстовом редакторе, в стиле которого можно сделать текстовый редактор создателя.*

##### РЕДАКТИРОВАНИЕ MIME-ТИПОВ

[*http://qt-project.org/doc/qtcreator-2.8/creator-mime-types.html*](http://qt-project.org/doc/qtcreator-2.8/creator-mime-types.html)

Qt Creator использует файл Internet media type (MIME), чтобы определить, в какой моде и какой использовать редактор, чтобы открыть данный файл. Для идентификации майм типа файла Qt Creator использует совпадение шаблонов и содержания. Сначала Qt Creator просматривает имя файла, чтобы проверить, совпадает ли оно с шаблоном, определённым для любого майм-типа. Если совпадения нет, то он проверяет содержание файла в магических заголовках файла, определённых внутри самого файла.

Магические заголовки могут содержать строки или байты. Qt Creator предоставляет возможность самостоятельно добавлять расширения и магические заголовки тех файлов, которые планируется использовать. *Как это сделать, показано в данной части.*

*То есть, фактически, при помощи создателя можно работать с другими типами текстовых файлов, которые важны для проекта.*

При изменении магических заголовков изменения внесутся при следующем запуске создателя.

#### ПОСТРОЕНИЕ И ЗАПУСК

<http://qt-project.org/doc/qtcreator-2.8/creator-building-running.html>

Qt Creator предоставляет поддержку построения, запуска и развёртывания приложений на различных целевых платформах или с использованием разных компиляторов, отладчиков, или версий Qt.

Kits определяют инструменты, типы устройства и другие настройки, чтобы использовать их во время построения и запуска вашего проекта. Настройки построения содержат всё, что вам нужно для компиляции исходных файлов в двоичные. Конфигурации построения используют инструменты и настройки, определённые в их соответствующих Kits.

Конфигурации запуска запускают приложение в локации, где оно было скопировано конфигурацией развёртывания. По умолчанию, когда вы выбираете функцию запуска, Qt Creator строит проект, развёртывает его на устройстве, определённом в Kit и запускает его там. Однако если вы не внесли никаких изменений в проект с момента последнего построения и развёртывания, Qt Creator просто запустит его снова.

Конфигурации развёртывания обрабатывают пакетирование и копирование необходимых файлов в локацию, в которой вы желаете запустить исполняемый файл. Файлы могут быть скопированы в локацию в файловой системе персонального компьютера или на мобильном устройстве.

Также для мобильных устройств следует настроить некоторое дополнительное программное обеспечение, чтобы соединить мобильные устройства и персональный компьютер.

По умолчанию, запуск приложения также строит приложение и развёртывает его в локации, из которой оно может быть запущено на устройстве. Вы можете изменить отношения между построением, запуском и развёртыванием конфигураций.

Вы можете использовать Qt симулятор, чтобы протестировать приложения Qt, которые намереваются для мобильных устройств в среде, подобной на ту, которая на устройстве. Также можно использовать симулятор, чтобы протестировать приложения, которые используют родные для устройства библиотеки. Для этого следует использовать эмуляторы устройств. Обычно, однако, быстрее протестировать это на реальном устройстве, чем на эмуляторе.

##### ПОСТРОЕНИЕ ДЛЯ МНОГИХ ПЛАТФОРМ

<http://qt-project.org/doc/qtcreator-2.8/creator-building-targets.html>

при построении проекта ошибки и предупреждения построения отображаются на Issues панели, а более детальная информация содержится на Comiler Output.

Для построения приложения следует выбрать Kit для проекта.

Чтобы быстро проверить вывод компиляции на изменения, которые вы сделали в одном из файлов или подпроектах, вы можете использовать команды меню Build, чтобы построить файл или подпроект.

##### ЗАПУСК НА МНОГИХ ПЛАТФОРМАХ

<http://qt-project.org/doc/qtcreator-2.8/creator-running-targets.html>

по умолчанию запуск приложения также строит и развёртывает его в локации, из которой оно может быть запущено на рабочем столе, симуляторе или на мобильном устройстве, которое присоединено к компьютеру разработчика.

*Показаны различные способы запуска приложения.*

[*http://qt-project.org/doc/qtcreator-2.8/creator-deployment.html*](http://qt-project.org/doc/qtcreator-2.8/creator-deployment.html)

[*http://qt-project.org/doc/qtcreator-2.8/creator-deploying-android.html*](http://qt-project.org/doc/qtcreator-2.8/creator-deploying-android.html)

[*http://qt-project.org/doc/qtcreator-2.8/creator-deployment-bb10.html*](http://qt-project.org/doc/qtcreator-2.8/creator-deployment-bb10.html)

[*http://qt-project.org/doc/qtcreator-2.8/creator-deployment-maemo.html*](http://qt-project.org/doc/qtcreator-2.8/creator-deployment-maemo.html)

[*http://qt-project.org/doc/qtcreator-2.8/creator-deployment-qnx.html*](http://qt-project.org/doc/qtcreator-2.8/creator-deployment-qnx.html)

[*http://qt-project.org/doc/qtcreator-2.8/creator-connecting-mobile.html*](http://qt-project.org/doc/qtcreator-2.8/creator-connecting-mobile.html)

[*http://qt-project.org/doc/qtcreator-2.8/creator-developing-android.html*](http://qt-project.org/doc/qtcreator-2.8/creator-developing-android.html)

[*http://qt-project.org/doc/qtcreator-2.8/creator-developing-bb10.html*](http://qt-project.org/doc/qtcreator-2.8/creator-developing-bb10.html)

[*http://qt-project.org/doc/qtcreator-2.8/creator-developing-generic-linux.html*](http://qt-project.org/doc/qtcreator-2.8/creator-developing-generic-linux.html)

[*http://qt-project.org/doc/qtcreator-2.8/creator-developing-maemo.html*](http://qt-project.org/doc/qtcreator-2.8/creator-developing-maemo.html)

[*http://qt-project.org/doc/qtcreator-2.8/creator-developing-meego.html*](http://qt-project.org/doc/qtcreator-2.8/creator-developing-meego.html)

[*http://qt-project.org/doc/qtcreator-2.8/creator-developing-qnx.html*](http://qt-project.org/doc/qtcreator-2.8/creator-developing-qnx.html)

*все данные части посвящены мобильным устройствам, что пока мне не нужно.*

##### ОТНОШЕНИЕ МЕЖДУ ПОСТРОЕНИЕМ, ЗАПУСКОМ И РАЗВЁРТЫВАНИЕМ

[*http://qt-project.org/doc/qtcreator-2.8/creator-build-process-customizing.html*](http://qt-project.org/doc/qtcreator-2.8/creator-build-process-customizing.html)

*указан путь, как изменить взаимоотношение между построением, развёртыванием и запуском.*

[*http://qt-project.org/doc/qtcreator-2.8/creator-maemo-emulator.html*](http://qt-project.org/doc/qtcreator-2.8/creator-maemo-emulator.html)

*тоже касается мобильных дел.*

#### ОТЛАДКА И АНАЛИЗ

[*http://qt-project.org/doc/qtcreator-2.8/creator-testing.html*](http://qt-project.org/doc/qtcreator-2.8/creator-testing.html)

если вы устанавливаете Qt Creator совместно с Qt SDK, то GNU Symgollic Debugger устанавливается автоматически и вам следует быть готовым к началу отладки после создания нового проекта. Однако вы можете изменить установки, чтобы использовать отладочные инструменты для Windows, например, вы можете производить отладку на подключенных к компьютеру мобильных устройствах.

Qt Creator интегрирует Valgrind анализатор кода для детектирования расходов памяти и профильных функций выполнения. Вы должны загрузить и установить их отдельно, чтобы использовать их из создателя. Профилирование QML доступно по умолчанию.

##### ОТЛАДКА

<http://qt-project.org/doc/qtcreator-2.8/creator-debugging.html>

Qt Creator обеспечивает плагин для отладчика, который действует как интерфейс между создателем и внешними нативными отладчиками, такими как GDB или CDB. Плагин автоматически находит отладчик в вашей системе и применяет его для вашего проекта. Отладчик можно изменять автоматически.

*Показано, как запустить отладку.*

Вы можете использовать в моде отладки специальные инструменты, чтобы проверять состояние вашего приложения во время отладки.

Также Qt Creator предоставляет некоторые более совершенные технологии, объединённые под названием debugging helpers. Эти технологии позволяют проводить быстрое тестирование сложных данных.

*Сказано про особенности отладки QtQuick приложений.*

##### УСТАНОВКА ОТЛАДЧИКА

<http://qt-project.org/doc/qtcreator-2.8/creator-debugger-engines.html>

Главные настройки отладчика связаны с некоторым Kit, при помощи которого вы строите и запускаете ваш проект. Вам необходимо устанавливать отладчик, только если автоматическая установка потерпела крах, так как нативный отладчик отсутствует. На Windows вы всегда должны устанавливать CDB самостоятельно.

Если вы хотите установить некоторый другой отладчик, то вам следует клонировать Kit и установить там новый отладчик, после чего использовать данный Kit для построения проекта.

Для установки инструментов отладки на Windows вы должны установить их и сервер символов, обеспечиваемый Microsoft, данный сервер обеспечивает путь поиска символов для отладчика. Сервер символов Microsoft обеспечивает вас отладочной информацией для управления системных библиотек для отладки приложений Windows. *Также показано, как добавить сервер символов.*

*В данной части перечислены поддерживаемые отладчики. А также для каждого отладчика указаны минимальные поддерживаемые версии. Также дана ссылка для установки инструментов отладки для Windows.*

Для того чтобы дать отладчику возможность шагнуть в код и отобразить исходный код, используя копию дерева исходного кода в локации, отличной от той, в которой библиотеки были построены, отобразите пути ресурсов на целевые пути: *как это сделать, показано в данной части, но я этого не понял.*

**ЗАПУСК ОТЛАДЧИКА**

<http://qt-project.org/doc/qtcreator-2.8/creator-debugger-operating-modes.html>

плагин отладчиков может запускать нативные отладчики в разных модах работы. Некоторые моды доступны только для некоторых операционных систем или платформ.

Вот моды работы отладчика:

* start internal (чтобы отлаживать приложения, разработанные внутри Qt, такие как основанные на Qt приложения с графическим интерфейсом пользователя);
* start external (начать отладку процессов без надлежащей установки проекта Qt либо на локальной, либо на удалённой машине);
* Attach (чтобы отладить процессы, которые уже начались и работают вне создателя, либо на локальной, либо на удалённой машине);
* Core (чтобы отладить процессы на Unix);
* Post-mortem (чтобы отладить кэшированные процессы на Windows).

Для запуска отладчика во внутренней моде, следует просто нажать отладку. Это мода по умолчанию для большинства проектов. Если вам нужна консоль, чтобы управлять вашим приложением, например, из-за того, что оно допускает ввод консоли от пользователя, то это следует установить отдельно. *Как это сделать, показано в данной части*.

Для запуска исполняемого файла, который уже присутствует на вашей локальной или удалённой машине без использования проекта, *следуйте инструкциям, которые приведены в данной части.* Это и есть мода отладчика start external.

Для того чтобы присоединить отладчик к приложению, которое уже работает на вашей машине, *следуйте инструкциям в данной части.* Это и есть мода Attach. Там нужно будет задать идентификатор процесса, который запущен на машине. Также в данном случае можно использовать Qt Creator для установки точек останова. Причём их можно устанавливать как до начала запуска проекта, так и после запуска проекта.

Core мода используется для инспекции файлов ядра, которые генерируются из кэшированных процессов в Linux и Unix системах, если система позволяет это. В данной моде также полезно иметь качественно сконфигурированный проект, содержащий исходники обрушенной программы.

Post-Mortem мода доступна только на Windows, если вы установили инструменты отладки для Windows. Программа установки создателя запрашивает вас, хотите ли вы зарегистрировать Qt Creator как post-mortem отладчик. Вы можете запустить отладчик в данной моде, если приложение обрушивается на Windows.

Qt Creator обеспечивает очень простой доступ к удалённой отладке.

##### ВЗАИМОДЕЙСТВИЕ С ОТЛАДЧИКОМ

<http://qt-project.org/doc/qtcreator-2.8/creator-debug-mode.html>

Вы можете использовать моду отладки создателя, чтобы инспектировать состояние вашего приложения во время отладки. Вы можете взаимодействовать с отладчиком несколькими способами, включая следующие:

* Идя через программу линия за линией и инструкция за инструкцией
* Прерывая работающие программы
* Устанавливая точки останова
* Проверяя содержание стека вызовов
* Проверяя и модифицируя регистры памяти отлаживаемой программы
* Проверяя список загруженных разделяемых библиотек
* Разъединяя секции кода
* Создавая снимки текущего состояния отлаживаемой программы и перепроверяя их позднее.

Также плагин отладчиков понимает некоторые свойственные для qt вещи, такие как QString, контейнеры Qt, QObject.

В моде отладки вы можете использовать несколько представлений для взаимодействия с программой, которую вы отлаживаете. Часто используемые представления показаны по умолчанию, а редко используемые представления скрыты. *В данной части показано, как это изменить.* Есть специальная функция прерывания программы во время отладки.

Если программа останавливается, то Qt Creator:

* получает данные, представляющие стек вызова в текущей позиции программы;
* получает содержания локальных переменных;
* тестирует выражения;
* обновляет регистры, модули и разъединённые представления, если вы отлаживаете основанные на С++ приложения.

*Также в данной части указаны горячие клавиши для управления отладчиком.*

Точки останова сохраняются вместе с сессией. *В данной части показано, как задавать точки останова, а также как задавать их свойства.* Можно задавать тип точки останова, также можно задавать условие, которое следует оценить прежде остановки. Остановка происходит, если условие оказывается правдой. В поле игнорирования следует указать количество раз, которое точка останова должна игнорироваться, прежде чем произойдёт остановка. В поле команд задайте команды, которые следует выполнить, когда программа остановится. В одной линии должна быть одна команда.

Точки останова можно передвигать между линиями по технике перетащи-отпусти. *Но почему-то в настоящем создателе это не очень получается.* Также точки останова можно удалять. *Способы удаления точек останова указаны в данной части.*

Точка останова данных останавливается, когда данные считываются или записываются по специальному адресу.

Когда отлаживаемая программа прерывается, Qt Creator отображает вложенные вызовы функций, приводящие к текущей позиции как след стека вызовов. Этот след стека построен из фреймов стека вызовов, каждый из которых представляет некоторую частную функцию. Для каждой функции Qt Creator пытается получить файл и номер строки. Когда бы ни останавливалась программа под контролем отладчика, он получает информацию о самом верхнем фрейме стека и отображает её в представлении локальных переменных и выражений. Они типично включают информацию о параметрах функции в этом фрейме также как и локальные переменные.

Сложные данные можно расширять, чтобы просматривать все данные.

В замечании указывается, что GDB, особенно на Linux, может оптимизировать свою работу и поэтому может не отображать некоторую информацию в данном представлении. Поэтому при работе с данным компилятором там вы можете увидеть неожиданные результаты.

Также при использовании помощников (*как их установить, написано в данной части*), будут отображаться особенности базовых объектов Qt. При этом при нажатии правой кнопкой мыши по представлению можно также изменять некоторые свойства, которые зависят от типа представляемых данных.

Также можно изменять во время отладки значения переменных простых типов. *Как это сделать, показано в данной части*.

В некоторых случаях полезно взаимодействовать с командной строкой некоторого отладчика. *В данной части показано, как это можно осуществить.* Но обычно данная особенность не нужна, так как Qt Creator сам обеспечивает всю полноту функциональности для выполнения отладки.

Отладку можно начать из командной строки для приложений C++. *Как это сделать, написано в данной части.*

Если многопоточная программа приостанавливается, то представление потоков может быть использовано, чтобы переключиться от одного потока на другой. Представление стека изменяет своё содержимое соответственно.

Представление модулей содержит информацию о модулях, включённых в приложение, что сейчас отлаживается. Модуль – это Dynamic Linked Library (.dll) on Windows, shared object (.so) in Linux and a dynamic shared library (.dylib) in Mac OS.

*Также есть представление файлов исходного кода.*

Также есть представление **Disassembler**, *назначение которого я не понял*. Кроме этого есть представление регистров центрального процессора, в котором отображены текущие состояния регистров центрального процессора.

Снимок содержит полное состояние отлаживаемой программы за раз, включая полное содержание памяти. Для работы со снимками используется специальное представление снимков.

##### ИСПОЛЬЗОВАНИЕ ВСПОМОГГАТЕЛЬНЫХ ФУНКЦИЙ ОТЛАДКИ

<http://qt-project.org/doc/qtcreator-2.8/creator-debugging-helpers.html>

В представлении локальных переменных и выражений сложные объекты и ассоциативные контейнеры отображаются сложным образом, так как внутренне они состоят из сложного набора взаимосвязанных указателей. Для более удобного представления таких данных служат вспомогательные функции отладки отладчика. Они могут быть основаны на Питоне и компилируемыми в зависимости от Kitа. Помощники позволяют обрабатывать около 80 стандартных классов Qt.

Во время использования помощников С++ отладчик загружает динамическую библиотеку в процесс отладки.

Помощник, основанный на питоне, намного проще, но он не поддерживается некоторыми системами. *Подробнее о нём написано в данной части. В данной части большая часть информации посвящена такому языку как питон, а также его использованию для отладки.*

[*http://qt-project.org/doc/qtcreator-2.8/creator-debugging-qml.html*](http://qt-project.org/doc/qtcreator-2.8/creator-debugging-qml.html)

*отладка QtQuick приложений*

[*http://qt-project.org/doc/qtcreator-2.8/creator-debugging-example.html*](http://qt-project.org/doc/qtcreator-2.8/creator-debugging-example.html)

*пример отладки кода С++*

[*http://qt-project.org/doc/qtcreator-2.8/creator-qml-debugging-example.html*](http://qt-project.org/doc/qtcreator-2.8/creator-qml-debugging-example.html)

*пример отладки приложения QtQuick*

##### ПРОБЛЕМЫ ПРИ ОТЛАДКЕ

[*http://qt-project.org/doc/qtcreator-2.8/creator-troubleshooting-debugging.html*](http://qt-project.org/doc/qtcreator-2.8/creator-troubleshooting-debugging.html)

в данной части находятся вероятные проблемы, с которыми вы можете столкнуться при отладке, и возможные пути их решения.

1. Некоторые антивирусы могут препятствовать отладчику получать информацию.
2. Если нет остановки в местах точек останова, то обратите внимание на настройки отладчика проекта (*смотри данную часть*).
3. Если отладчик не работает хорошо, попробуйте следующее: удостоверьтесь, что вы используете по крайней мере Qt Creator версии 2.1; удостоверьтесь, что отладчик установлен качественно; откройте лог отладчика и посмотрите, что пошло не так.
4. Если переменные-указатели не отображаются, то следует сделать то, *что описано в данной части.*
5. Если отладчик работает медленно, то уменьшите количество точек останова.
6. Отладчик выдаёт сообщение <вне области видимости>: скорее всего, неправильно работают помощники.
7. В некоторых дистрибутивах Linux не разрешается отладчику подключиться к работающему приложению. *В конце данной части показано, как этого избежать.*

#### АНАЛИЗ КОДА

<http://qt-project.org/doc/qtcreator-2.8/creator-analyze-mode.html>

в моде анализа можно использовать инструменты анализа кода. В данной моде можно выбрать тип анализа. Это анализ производительности и qml профайлер.

<http://qt-project.org/doc/qtcreator-2.8/creator-qml-performance-monitor.html>

про исследование QML-приложений.

##### VALGRID

<http://qt-project.org/doc/qtcreator-2.8/creator-valgrind-overview.html>

Qt Creator использует Valgrid для контроля расхода памяти и профайлинга исходного кода. Valgrid следует установить отдельно для использования в создателе.

Valgrid можно запускать удалённо, а также настраивать его.

##### ДЕТЕКТИРОВАНИЕ ПОТЕРЬ ПАМЯТИ

<http://qt-project.org/doc/qtcreator-2.8/creator-analyzer.html>

в Valgrid интегрировано средство для тестирования расходов памяти: memcheck, которое доступно локально на Linux, но только удалённо на Windows.

После установки Valgrid можно использовать memcheck прямо из создателя. *Как это сделать, показано в данной части.*

Во время работы приложения memcheck проверяет все считывания и записи памяти и собирает все вызовы, которые выделяют и освобождают память. По окончании работы приложения memcheck отображает результаты, причём можно посмотреть в стеке вызовов, что вызвало данное использование памяти.

Настройки тестирования памяти можно задавать для всех проектов глобально либо для конкретного проекта.

*В настройках можно задавать размер стека вызовов, а также файлы подавления. Но я пока не понимаю, что это означает.*

##### ПРОФАЙЛИНГ ВЫПОЛНЕНИЯ ФУНКЦИЙ

<http://qt-project.org/doc/qtcreator-2.8/creator-cache-profiler.html>

также можно использовать callgrid для тестирования выполнения некоторых функций. Это средство также можно использовать из создателя. *В данной части показано, как его включить.*Данное средство собирает информацию о вызовах функций. Также можно задавать настройки данного средства. Причём задавать их можно как для всех проектов глобально, так и для конкретного проекта локально.

*Также можно настраивать каким-то образом кэш. Относительно этого я не понял. Также можно просматривать разные ветви и перескоки программы (видимо это касается циклов и условных операторов)*

<http://qt-project.org/doc/qtcreator-2.8/creator-running-valgrind-remotely.html>

на системе Windows можно работать с Valgrid удалённо, для чего следует задать Kit и приложение. *Как это сделать, показано в данной части*.

<http://qt-project.org/doc/qtcreator-2.8/creator-publish-ovi.html>

<http://qt-project.org/doc/qtcreator-2.8/creator-publish-ovi-maemo.html>

<http://qt-project.org/doc/qtcreator-2.8/creator-publish-ovi-meego.html>

<http://qt-project.org/doc/qtcreator-2.8/creator-publishing-to-maemo-extras.html>

*публикация приложений в мобильных магазинах*

#### ПРОДВИНУТОЕ ИСПОЛЬЗОВАНИЕ QT CREATOR

[*http://qt-project.org/doc/qtcreator-2.8/creator-advanced.html*](http://qt-project.org/doc/qtcreator-2.8/creator-advanced.html)

при использовании создателя вы можете загрузить SDK с автоматическими установками и использовать сразу Qt Creator. Но вы также можете настроить его на свой лад: например, Qt Creator можно запускать и настраивать из командной строки. Также в создателе существует большое количество горячих клавиш для управления им при помощи клавиатуры. Внешние инструменты можно использовать прямо из создателя. Можно также сделать так, чтобы файлы описания внешних инструментов, таких как анализаторы кода, появлялись на панели issues.

<http://qt-project.org/doc/qtcreator-2.8/creator-os-supported-platforms.html>

*перечислены поддерживаемые платформы.*

<http://qt-project.org/doc/qtcreator-2.8/creator-project-other.html>

вместо qmake можно использовать для построения CMake. Также Qt Creator можно использовать просто как редактор кода и всё.

#### CMAKE

<http://qt-project.org/doc/qtcreator-2.8/creator-project-cmake.html>

CMake – это другое средство построения, которое также можно настраивать. Оно отличается от qmake принципом работы и генерируемыми файлами. Для данного строителя следует указать путь для исполняемого файла. *Как это делается, показано в данной части.*

<http://qt-project.org/doc/qtcreator-2.8/creator-project-qbs.html>

*рассказано про некоторое экспериментальное средство qt.*

[*http://qt-project.org/doc/qtcreator-2.8/creator-projects-autotools.html*](http://qt-project.org/doc/qtcreator-2.8/creator-projects-autotools.html)

*не читал, что здесь конкретно сказано, но считаю, что в будущем это можно изучить.*

[*http://qt-project.org/doc/qtcreator-2.8/creator-project-generic.html*](http://qt-project.org/doc/qtcreator-2.8/creator-project-generic.html)

*рассказывается, как сделать так, чтобы Qt Creator выступал просто как редактор кода.*

#### ИСПОЛЬЗОВАНИЕ НАСТРОЕК КОМАНДНОЙ СТРОКИ

[*http://qt-project.org/doc/qtcreator-2.8/creator-cli.html*](http://qt-project.org/doc/qtcreator-2.8/creator-cli.html)

*рассказывается про использование параметров командной строки для работы с построением при помощи создателя. Можно также попробовать данные возможности.*

#### ГОРЯЧИЕ КЛАВИШИ

[*http://qt-project.org/doc/qtcreator-2.8/creator-keyboard-shortcuts.html*](http://qt-project.org/doc/qtcreator-2.8/creator-keyboard-shortcuts.html)

*перечислены горячие клавиши для qt.*

#### ВНЕШНИЕ ИНСТРУМЕТНЫ

[*http://qt-project.org/doc/qtcreator-2.8/creator-editor-external.html*](http://qt-project.org/doc/qtcreator-2.8/creator-editor-external.html)

*рассказано про использование внешних инструментов, в том числе и Qt Linguist.*

#### ПОКАЗ ФАЙЛОВ СПИСКА ЗАДАЧ НА ПАНЕЛИ ISSUE

[*http://qt-project.org/doc/qtcreator-2.8/creator-task-lists.html*](http://qt-project.org/doc/qtcreator-2.8/creator-task-lists.html)

Эта панель показывает те вопросы, которые вам следует решить. Qt Creator позволяет вам загрузить список задач в данную панель для более лёгкой навигации.

*В данной части есть ссылка на пример. показано, как использовать файлы задач на панели issues. Как я понимаю, это особенные проблемы кода, которые специфичны для вашего приложения.*

#### СПРАВКА

[*http://qt-project.org/doc/qtcreator-2.8/creator-help-overview.html*](http://qt-project.org/doc/qtcreator-2.8/creator-help-overview.html)

*рассказывается, как пользоваться справкой.*

[*http://qt-project.org/doc/qtcreator-2.8/creator-help.html*](http://qt-project.org/doc/qtcreator-2.8/creator-help.html)

##### МОДА СПРАВКИ

*использование моды справки.*

##### FAQ

[*http://qt-project.org/doc/qtcreator-2.8/creator-faq.html*](http://qt-project.org/doc/qtcreator-2.8/creator-faq.html)

*содержатся наиболее часто задаваемые вопросы и ответы на них.*

##### СОВЕТЫ И ХИТРОСТИ

[*http://qt-project.org/doc/qtcreator-2.8/creator-tips.html*](http://qt-project.org/doc/qtcreator-2.8/creator-tips.html)

Переключение между модами – Ctrl + число от 1 до 7. Ctrl + tab – быстрое передвижение между открытыми в настоящий момент файлами. Открытие панелей вывода – Alt + цифры от 1 до 4. Можно настраивать горячие клавиши. *Следует обратить внимание на создание лицензионного шаблона в лицензионном заголовочном файле С++.*

##### ИЗВЕСТНЫЕ БАГИ

[*http://qt-project.org/doc/qtcreator-2.8/creator-known-issues.html*](http://qt-project.org/doc/qtcreator-2.8/creator-known-issues.html)

*перечислены некоторые хорошо известные баги qt, которые затем следует более подробно изучить.*

#### СЛОВАРЬ ТЕРМИНОВ QT CREATOR

[*http://qt-project.org/doc/qtcreator-2.8/creator-glossary.html*](http://qt-project.org/doc/qtcreator-2.8/creator-glossary.html)

*словарь наиважнейших терминов касательно создателя. Полезная вещь.*

# ВСЕ МОДУЛИ QT

<http://qt-project.org/doc/qt-5.1/qtdoc/qtmodules.html>

зашёл на страницу классов qt (<http://qt-project.org/doc/qt-5.1/qtdoc/classes.html#x>). Оказалось, что в данной библиотеке приблизительно около 1000 классов.

Данная часть содержит необходимое основание данной библиотеки на всех платформах. Они все будут оставаться исходно и двоично совместимыми на протяжение выпуска Qt5. Основные модули являются общими. Они полезные для большинства приложений qt. Остальные модули, которые не рассмотрены в данной части, считаются неосновными, но дополнительными.

1. QtCore (неграфические классы ядра, используемые другими модулями),
2. QtGui (основные классы для графического интерфейса пользователя. Включает OpenGL),
3. Qt Multimedia (классы для аудио, видео и функциональности камеры),
4. Qt Multimedia Widgets (основанные на виджетах классы для применения функциональности мультимедиа),
5. Qt Network (классы для программирования сетей),
6. Qt QML (классы для QML и JavaScript),
7. Qt Quick (декларативный каркас для построения высоко динамичных приложений с частным графическим интерфейсом пользователя),
8. Qt Quick Controls (повторно используемые основные элементы управления для создания классических интерфейсов пользователя),
9. Qt Quick Layouts (макеты для элементов, которые испльзуются для организации элементов, основанных на Qt Quick 2),
10. Qt SQL (классы для интегрирования в базу данных SQL),
11. Qt Test (классы для модульного тестирования приложений qt и библиотек),
12. Qt WebKit (классы для применений WebKit2 и новых QML API),
13. Qt WebKit Widgets (WebKit1 и основанные на QWidget классы из Qt4),
14. Qt Widgets (классы, расширяющие Qt Gui C++ виджеты).

Если вы используете qmake, то модули Qt Core и Qt Gui добавляются автоматически. Если вы используете СMake на Windows, вам также следует присоединить библиотеку qtmain.

Дополнительные модули qt предоставляют дополнительные возможности для решения некоторых специфических задач. Некоторые из этих модулей могут быть доступны только на некоторых платформах.

1. Active Qt – Windows классы для приложений, которые используют Active X и COM.
2. Qt Concurrent – классы для написания многопоточных программ без использования низкоуровневых поточных примитивов.
3. Qt D-Bus – Linux – классы для внутрипроцессной коммуникации над D-Bus протоколом.
4. Qt Graphical Effects – графические эффекты для использования с Qt Quick2.
5. Qt Image Formats – плагины для дополнительных форматов изображений TIFF, MNG, TGA, WBMP.
6. Qt OpenGL – классы для поддержки OpenGL (только для поддержки приложений из Qt4. В новом коде используйте классы из Qt Gui).
7. Qt Print Support – классы для того, чтобы сделать печать легче и более портативно.
8. Qt Declarative – обеспечивается для совместимости с Qt4.
9. Qt Script – классы для представления приложений Qt в виде сценариев. Обеспечивается для совместимости с Qt4. Для нового кода используйте QtQml.
10. Qt Script Tools – дополнительные компоненты для приложений, которые используют Qt Script.
11. Qt Sensors – для мобильных устройств.
12. Qt XML – С++ реализация SAX и DOM. Устарел. Использовать вместо него классы QXmlStreamReader и QXmlStreamWriter.
13. Qt XML Patterns – поддержка XPath, XQuery, XSLT и XML схем.

Также в qt есть некоторые инструменты: Qt Designer, справка и Qt UI Tools для динамической загрузки виджетов во время выполнения.

# МОДУЛЬ ЯДРА

<http://qt-project.org/doc/qt-5.1/qtcore/qtcore-index.html>

Все модули qt полагаются на модуль ядра. При помощи ядра qt добавляет к С++ следующую функциональность:

* Механизм сигналов и слотов для коммуникации между объектами;
* Свойства объектов
* Иерархические объектные деревья
* Владение объектами безопасным способом при помощи специальных указателей
* Динамическое определение типов, которое работает сквозь границы библиотек.

## ОБЪЕКТНАЯ МОДЕЛЬ

<http://qt-project.org/doc/qt-5.1/qtcore/object.html>

Стандартная объектная модель С++ обеспечивает очень эффективную поддержку объектной парадигмы во время выполнения. Но в некоторых проблемных областях её статическая природа является очень негибкой. Например, разработка графического интерфейса пользователя требует как эффективности во время выполнения, так и высокого уровня гибкости. Qt обеспечивает это, комбинируя скорость С++ с объектной моделью qt.

Qt добавляет следующие особенности к С++:

1. Сигналы и слоты
2. Объектные свойства
3. Мощные события и фильтры событий
4. Контекстуальный перевод строк для интернационализации
5. Изысканные таймеры, которые позволяют элегантно интегрировать много задач в управляемое событиями приложение с графическим интерфейсом пользователей
6. Иерархические и запрашиваемые деревья объектов, которые организуют владение объектом естественным образом
7. Охраняемые указатели, которые автоматически установливаются в 0, когда ссылаемые объекты уничтожаются, в отличие от нормального С++ указателя, который становится свисающим, когда его объект уничтожается.
8. Динамическое приведение типов, которое работает сквозь границы библиотек (*не понимаю, причём здесь границы библиотек*)
9. Поддержка создания частных типов

Многие из этих особенностей qt применяются со стандартными техниками qt, основанными на наследовании от класса QObject. Другие, такие как механизм коммуникации объектов и динамическая система свойств, требуют мета-объектной системы, обеспечиваемой moc.

Мета-объектная система – это расширение С++, которое делает язык лучше пригодным к программированию компонентов графического интерфейса пользователя.

Некоторые из добавленных особенностей, перечисленных выше для объектной модели qt, требуют, чтобы мы думали об объектах qt как об идентичностях, а не как о значениях. Значения копируются или присваиваются, а идентичности клонируются. Они выглядят идентичными, но они имеют разные имена, разные локации, и могут иметь полностью разные социальные связи. Клонирование идентичности является более сложной операцией, чем копирование или присваивание значения. Мы можем видеть, что это значит в объектной модели qt.

Объект qt может

1. Иметь уникальное имя ([QObject::objectName](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html#objectName-prop)()). Если мы копируем, то какое имя мы должны дать объекту?
2. Иметь локацию в объектной иерархии. Если мы копируем объект, то где мы должны его разместить?
3. Может быть связан с другими объектами qt при помощи механизма сигналов и слотов. Как мы должны переносить данные связи на копию объекта?
4. Может иметь новые свойства, которые добавляются ему во время выполнения и которые не объявлены в С++ классе. Должна ли копия включать данные свойства?

По этим причинам объекты qt должны обрабатываться как идентичности, но не как значения. Идентичности клонируются, но не копируются или присваиваются. Поэтому класс QObject и все его подклассы (прямые или непрямые) имеют свои конструкторы копирования и операторы присваивания защищёнными.

### ОБЪЕКТНЫЕ ДЕРЕВЬЯ И ВЛАДЕНИЕ

<http://qt-project.org/doc/qt-5.1/qtcore/objecttrees.html>

QObject организует себя в дерева объектов. В каждом подклассе данного класса есть некоторые внутренние члены данных, которые отслеживают родителей и детей. Класс QWidget также расширяет данную функциональность, так как наследник также обычно становится виджетом, что означает, что он также показывается графически внутри рамки своего родителя. Этот механизм активно используется при удалении объектов. Например, при удалении некоторого родительского виджета также удаляются все его дочерние виджеты.

Когда объекты создаются в яме, то дерево может быть построено из них в любом порядке, а позднее объекты в дереве могут быть уничтожены в любом порядке. Когда в дереве удаляется любой объект QObject, то если данный объект имеет родителя, деструктор автоматически удаляет объект из его родителей. Если объект имеет наследников, то деструктор автоматически удаляет каждого наследника. Ни один объект не будет уничтожен дважды.

В стандарте С++ деструкторы вызываются в обратном порядке конструкторов. *В данной части поэтому указывается, что в обычном С++ нельзя создавать объекты в произвольном порядке, если известно в каком порядке они будут удаляться, но в qt это делать можно, как было сказано ранее.*

### МЕТА-ОБЪЕКТНАЯ СИСТЕМА QT

<http://qt-project.org/doc/qt-5.1/qtcore/metaobjects.html#meta-object-system>

Мета-объектная система обеспечивает

1. механизм сигналов и слотов для коммуникации между объектами,
2. динамическую информацию о типе,
3. динамическую систему свойств.

Мета-объектная система основа на трёх вещах:

1. QObject класс обеспечивает базовый класс для объектов, которые пользуются преимуществами мета-объектной системы.

2. Q\_OBJECT макрос внутри секции закрытых членов класса используется, чтобы обеспечить возможности особенностей мета-объектной системы, такие как динамические свойства, сигналы и слоты.

3. moc, который обеспечивает каждый подкласс QObject необходимым кодом, чтобы выполнять особенности мета-объектной системы.

Помимо всех вышеуказанных особенностей мета-объектная система предоставляет несколько функций:

1. для возврата всего связанного мета-объекта для данного класса;
2. для получения во время выполнения имени класса без использования средств RTTI,
3. для возврата информации о том, является ли данный объект экземпляром класса, который наследует определённый класс в иерархии QObject;
4. функции для интернационализации;
5. функции для динамических сеттеров и геттеров по имени свойства;
6. для построения экземпляра некоторого класса.

Также можно использовать специальную функцию динамического приведения типов для объектов QObject: qobject\_cast(). Эта функция работает подобно dynamic\_cast(), но при этом не требует поддержки RTTI и работает сквозь границы динамических библиотек (*что значит, сквозь границы динамических библиотек?)*

Строго рекомендуется, чтобы все подклассы QObject содержали макрос Q\_OBJECT, чтобы к ним была подключена мета-объектная система.

#### СИСТЕМА СВОЙСТВ

<http://qt-project.org/doc/qt-5.1/qtcore/properties.html#qt-s-property-system>

qt обеспечивает изысканную систему свойств, подобную тем системам, которые поддерживают некоторые производители компиляторов. Но решение qt работает с любым стандартным компилятором на любой поддерживаемой платформе. Она основана на мета-объектной системе.

Для объявления свойств используйте макрос Q\_PROPERTY() в классе, который наследует QObject.

Свойство ведёт себя как обычный член класса, но имеет ряд особенностей, обеспечиваемых мета-объектной системой:

1. READ: требуется, если не была определена переменная MEMBER. Эта функция для считывания значения свойства. Функция может возвращать тип самого свойства, или указатель на него, или ссылку.
2. WRITE: необязательный. Она предназначена для установки свойств. Она должна возвращать void и должна иметь только один аргумент того же типа, что и свойство.
3. MEMBER: требуется, если нет функции доступа READ. Это делает данную переменную класса считываемой и записываемой без необходимости использования функций доступа (*видимо, её можно изменять не только при помощи функций доступа*).
4. RESET: настраиваемая функция. Возвращает значение свойства в значение по умолчанию. Данная функция должна возвращать void и не иметь параметров.
5. NOTIFY: необязательный. Определяет некоторый сигнал, который эмитируется каждый раз, как свойство изменяется.
6. REVISION: необязательный. Определяет свойства и их уведомления при исползьовании их в определённой ревизии (для qml).
7. DESIGNABLE: необязательный. Должно ли свойство быть видимым в дизайнере. По умолчанию да. Можно также определить булеву функцию-член.
8. SCRIPTABLE: необязательный. Атрибут указывает, должен ли иметь доступ к данному свойству скриптовый движок.
9. STORED: необязательный. Указывает, зависит ли свойство от других или существует само по себе. Также используется для того, чтобы указать, следует ли сохранять данное свойство, когда сохраняется состояние объекта.
10. USER: атрибут указывает, предназначено ли это свойство для использования пользователями. В классе может быть только одно свойство.
11. CONSTANT: необязательный. Атрибут указывает, что значение свойства постоянно.
12. FINAL: необязательный. Атрибут указывает, что свойство не может быть переписано в производном классе. Лучше никогда не следует переписывать данный атрибут свойства.

Функции READ, WRITE и RESET могут быть унаследованы, а также быть виртуальными.

Если используется множественное наследование, то они наследуются из первого класса. Типом свойства может быть любой тип, поддерживаемый QVariant или любой пользовательский тип. Заметьте, что внутри макросов не должно быть внутренних строк с запятыми, так как они используются системой для обработки макросов. Например, QMap<QString, QVariant> будет неправильно, тогда как QMap будет правильно.

Вся польза данной системы состоит в том, что свойство может быть считано, если только известно само свойство, но ничего неизвестно о классе. Для этого используются функции QObject::setProperty() и QObject::property(). *Пример их использования показан в данной части.*

В данном случае во время компиляции можно знать только имя свойства, но не знать имени класса. Но как узнать имя данного свойства??? Для этого необходимо запросить у объекта во время выполнения его QObject, QMetaObject или QMetaProperties.

*Пример кода показан в данной части. В самом коде я ссылаюсь только на указатель на объект QObject, но во время выполнения данным объектом может быть любой его подкласс. Если же известно имя свойства, то его можно установить для любого подкласса, который имеет данное свойство.*

Q\_ENUMS позволяет функции QObject::property() видеть перечисления, создаваемые в классе.

Обычно сигнал NOTIFY высылается в теле функции setProperty() для некоторого свойства.

Внутри функций чтения и записи имя свойства употребляется с префиксом m\_.

Затем демонстрируется, что есть две возможности установить свойства:

1. через функции доступа и
2. через функцию QObject::setProperty().

Q\_FLAGS() – это макрос, который также определяет перечисление, но данное перечисление состоит из флагов, которые могут быть логически объединены.

Динамические свойства. Здесь описано поведение функции QObject::setProperty(). Если такое свойство существует в данном объекте и переданное значение соответствует его типу, то свойство записывается и функция возвращает истину. Если тип значения несовместим с данным свойством, но оно существует, то функция возвращает ложь, но свойство не переписывается. Если же свойства с таким именем нет в объекте, то новое свойство с данным именем и значением автоматически добавляется в данный объект. *Интересно, оно добавляется именно в данный класс или в данный экземпляр? Это похоже на JS.*

Частные типы, используемые свойствами, должны быть зарегистрированы с использованием макроса Q\_DECLARE\_METATYPE(), чтобы их значения могли бы быть сохранены в объектах QVariant. Это делает возможным их использование как для свойств, определённых при помощи макроса Q\_PROPERTY() и при использовании динамического определения свойств.

В дополнение к системе свойств есть некоторый дополнительный макрос Q\_CLASSINFO(), который может быть использован, чтобы добавить дополнительные пары имя – значение к мета-объекту класса, например, версию, дату или автора. Возможно, кстати, даты создавать было бы даже неплохо для отслеживания процесса разработки. А затем при помощи специально написанного анализатора кода убирать данные вставки из конечной версии.

#### МЕХАНИЗМ СИГНАЛОВ И СЛОТОВ

<http://qt-project.org/doc/qt-5.0/qtcore/signalsandslots.html#signals-and-slots>

сигналы и слоты используются для коммуникации между объектами. Это одна из наиболее отличительных особенностей qt.

Раньше проблема коммуникации между объектами решалась при помощи функций обратного вызова. Это указатели на функции, которые передавались работающей функции, которая вызывает функцию обратного вызова затем, когда необходимо. У такого решения есть два потенциальных недостатка: во-первых, это небезопасно, так как управляющая функция может передать функции обратного вызова произвольные аргументы. Во-вторых, управляющая функция должна знать, какую функцию обратного вызова вызывать.

В qt существует альтернатива: сигнал эмитируется каждый раз, как происходит некоторое событие. Есть предопределённые сигналы, но можно создавать и собственные сигналы. Слоты – это функции, которые вызываются при инициировании сигнала.

Moc при анализе кода создаёт мета-объект, который содержит список всех сигналов и слотов, а также указатели на данные функции. Q\_OBJECT макрос обрабатывается препроцессором, чтобы объявить некоторые функции-члены, которые используются moc. Moc игнорирует остальные функции-члены.

1. Это безопасный механизм с точки зрения типа, так как сигнатура сигнала должна совпадать с сигнатурой слота-получателя.
2. Слот может иметь и меньше аргументов, чем сигнал, игнорируя дополнительные аргументы.
3. Они слабо связаны между собой, так как класс, эмитирующий сигнал может ничего не знать о классе, который является его получателем.
4. Слоты и сигналы могут иметь сколько угодно аргументов любого типа, но использовать их могут только объекты QObject.
5. Слоты являются в остальном обычными функциями-членами и могут использоваться отдельно.
6. Сигналы генерируются moc и не должны определяться в исполняемом файле. Также они не должны возвращать значение.
7. Слоты можно делать виртуальными.
8. Аргументы сигналов и слотов могут иметь значения по умолчанию.
9. В qt5 был введён новый синтаксис, чтобы можно было перегружать сигналы.

Сигналы можно присоединять ко многим слотам и наоборот. Более того, сигналы можно соединять с сигналами. Для подключения данной возможности и используется макрос Q\_OBJECT(). Функция QObject::connect() используется для соединения сигналов и слотов.

По умолчанию, для каждого соединения испускается один сигнал. Если соединение повторяется, то испускаются два сигнала и т.д. Для того чтобы устранить дополнительные сигналы, следует использовать функцию QObject::disconnect(), которая устраняет одно из соединений, или обеспечить тип Qt::UnicConnection, когда при повторном случае встречи данного соединения, оно не реализуется; то есть, если уже есть соединение одинаковых сигналов и слотов от одинаковых объектов.

Для испускания сигналов в коде используется слово emit signal().

В документации рекомендуется испускать сигнал только из того класса, который его определил.

При вызове слотов при помощи посылки сигналов этот процесс происходит независимо от цикла обработки событий графического интерфейса пользователя. Код, который следует за эмиссией сигнала, будет выполняться только тогда, когда сигналы будут выполнены. Другая ситуация имеет место для очерёдных соединений, когда код за эмиссией происходит немедленно.

Если сигнал соединён с несколькими слотами, то они будут выполнены в том порядке, в котором были соединены с сигналом.

Для того чтобы сигналы могли соединяться со слотами различных типов, они не должны иметь некоторые специфические для данного класса типы.

Вызов слотов через сигналы происходит в примерно 10 раз медленнее, чем их прямой вызов. Это время тратится на поиск нужного объекта и на проверку безопасности соединения (объект существует). Но зато механизм очень простой. Пользователи едва ли заметят это замедление, так как слоты вызываются через сигналы сравнительно редко.

*Также показан пример, как проверять, является ли данный класс наследником некоторого другого класса.*

Для связи сигналов и слотов при помощи функции QObject::connect() можно использовать как указатели на функции, так и макросы SIGNAL() и SLOT(), но при этом количество аргументов слота должно быть меньше или равно количеству аргументов сигнала.

Если необходимо знать информацию о том, кто послал сигнал, то есть функция QObject::sender(), которая возвращает указатель на объект, который прислал сигнал.

Класс QSignalMapper используется на тот случай, если к слоту присоединено много сигналов и все их слот обрабатывает по-разному. *Данный класс работает так: сначала в маппер помещаются отображения объектов, которые могут высылать сигнал, а также некоторая сопроводительная информация в виде строки или числа, по которой затем можно будет отличить отправителя. После этого сигналы отправителей соединяются с сигналами маппера. А определённый сигнал маппера с сопроводительным параметром, который будет инициироваться каждый раз, как будет инициироваться сигнал настоящего объекта, связывается со слотом объекта. Фактически маппер используется для преобразования сигнала отправителя в другой сигнал, который содержит информацию об отправителе*.

Также можно использовать сторонние системы сигналов и слотов. Для этого следует в файле проекта задать в настройках конфигурации CONFIG += no\_keywords. Данная строка просто говорит moc не обращать внимание на ключевые слова, которые теперь будут использованы третьей стороной. Если к некоторым компонентам нужно использовать систему сигналов и слотов qt, то следует использовать макросы Q\_SIGNALS(), Q\_SLOTS(), Q\_EMIT() вместо обычных ключевых слов.

#### СИСТЕМА РЕСУРСОВ

<http://qt-project.org/doc/qt-5.1/qtcore/resources.html>

Система ресурсов qt является платформенно-независимым механизмом для сохранения двоичных файлов в исполняемом приложении. Это удобно, если ваше приложение нуждается в наборе определённых файлов (иконки, файлы перевода и т.д.) и вы не желаете увеличивать риск потери файлов. Система ресурсов основана на тесном взаимодействии между qmake, rcc и QFile.

Ресурсы, связанные с приложением, определяются в файле .qrc – xml-подобном формате файлов, который перечисляет файлы на диске и может присваивать им имена ресурсов, которые использует приложение для доступа к данным ресурсам. Все пути в данном файле являются относительными к той директории, в которой находится данный файл.

Данные ресурсов могут либо быть скомпилированными в двоичный файл, и тогда доступ к ним осуществляется немедленно, либо бинарный ресурс может быть создан, а позднее зарегистрирован системой ресурсов. Ресурсы в приложении доступны под тем же именем, которое они имеют в дереве источников, но с префиксом :/, или при помощи URL с qrc схемой: :/images/cut.png или qrc:///images/cut.png.

В файле ресурсов можно специфицировать идентификатор к любому файлу на своё усмотрение. *Пример того, как это делается, показан в данной части.*

Для внешнего двоичного ресурса вы должны создать данные ресурса (обычно в формате .rcc). Данный ресурс следует регистрировать в приложении следующим образом: [QResource](http://qt-project.org/doc/qt-5.1/qtcore/qresource.html)::registerResource("/path/to/myresource.rcc");

*Чем отличается внешний ресурс от внутреннего ресурса.*

Для ресурса, который компилируется в бинарный, файл .qrc должен быть упомянут в файле проекта приложения .pro. Этот файл содержит все данные для изображений и других ресурсов как статические массивы С++, состоящие из сжатых двоичных данных. Файл qrc\_application.cpp автоматически регенерируется каждый раз, когда .qrc файл изменяется или один из файлов, на которые он ссылается, изменяются. Если вы не используете .pro, то вы должны запрашивать rcc вручную или добавлять правила построения в вашу систему построения.

В настоящее время, qt всегда сохраняет данные прямо в исполняемом файле, даже если операционная система обеспечивает нативную поддержку ресурсов. Это может измениться в будущих выпусках Qt.

Ресурсы сжимаются по умолчанию (в формате ZIP). Также возможно выключить сжатие. Это полезно тогда, когда ваши ресурсы уже содержат сжатый формат, такой как .png файлы. Это делается при помощи аргумента командной строки –no-compress. Также можно задавать уровень сжатия и пороговый уровень.

В приложении, пути ресурсов могут быть использованы во многих местах вместо обычных путей файлов системы. Его можно передать QIcon, QImage или QPixmap конструкторам.

В памяти ресурсы представлены деревом объектов ресурсов. Это дерево автоматически строится при запуске и используется QFile для решения путей для ресурсов. Вы можете использовать QDir (устарел), чтобы осуществлять навигацию через дерево ресурсов. Ресурсы Qt поддерживают концепцию поиска в списке путей.

Если вы имеете ресурсы в статической библиотеке, вам может быть необходимо инициализировать ваши ресурсы при помощи вызова макроса Q\_INIT\_RESOURCE() с основным именем .qrc файла (*видимо из библиотеки*).

Если надобности в ресурсах более нет, то можно освободить ресурсы при помощи макроса Q\_CLEANUP\_RESOURCE() с базовым именем, совпадающим с .qrc файлом данного ресурса. В этом может возникнуть необходимость при удалении, например, плагина или если ресурсы более недействительны.

### КЛАССЫ ОБЪЕКТНОЙ МОДЕЛИ

Класс QObject является базовым классом всех объектов qt. Это сердце объектной модели qt.

Q\_DECLARE\_METATYPE позволяет регистрировать частные типы для использования системой свойств. Класс QMetaObject содержит мета-информацию об объектах qt. класс QMetaMethod обеспечивает метаданные о функциях членах. Класс QMetaEnum обеспечивает метаданные о перечислителях. класс QMetaProperty обеспечивает метаданные о свойстве.

Класс QMetaTypeInfo предоставляет дополнительную информацию о классе. Это просто пары имя-значение, которые задаются при помощи макроса Q\_CLASSINFO().

QVariant выступает как объединение для наиболее общих типов данных Qt.

Объектная система Qt предоставляет несколько типов различных указателей. Класс QSharedPointer удерживает сильную ссылку на разделяемый указатель. Он устанавливается в нуль, если объект выходит из области видимости. Этот класс выполняет подсчёт ссылок на данный указатель. QWeakPointer (устарел) является слабой ссылкой на разделяемый указатель. Он может ссылаться уже на несуществующий объект. QPointer – это шаблонный класс, который ведёт себя как обычный С++ указатель, но устанавливается в нуль при выходе объекта из области видимости. QScopedPointer (устарел). QSharedDataPointer представляет указатель на неявно разделяемый объект. Класс QExplicitlySharedDataPointer класс представляет указатель на явно разделяемый объект.

QObjectCleanUpHandler класс следит за временем жизни множества объектов QObject. Данный класс полезен всякий раз, когда вам необходимо знать, когда некоторое количество объектов QObject, которыми владеет некто, также были удалены.

#### Qt пространство имён

<http://qt-project.org/doc/qt-5.1/qtcore/qt.html>

пространство имён Qt, которое содержит разнообразные идентификаторы, используемые в qt библиотеке. *Было бы интересно их просмотреть более подробно.*

#### Q\_DECLARE\_METATYPE()

<http://qt-project.org/doc/qt-5.1/qtcore/qmetatype.html#Q_DECLARE_METATYPE>

Данный макрос делает некоторый тип известным мета-объектной системе. Тип должен предоставлять как конструктор по умолчанию, так и конструктор копирования. Класс используется как помощник к маршаловским типам в QVariant и для соединений сигналов и слотов.

Если объявить новые типы через макрос Q\_DECLARE\_METATYPE(), то они становятся доступны для QVariant и других, основанных на шаблонах функций. Всё это в конченом итоге необходимо, чтобы использовать данный тип как частный тип для QVariant.

Этот макрос требует, чтобы данный тип был полностью определён к тому моменту, как макрос используется. Если это касается указателей, то требуется, чтобы указатель на тип был также полностью определён. Для этого следует использовать макрос Q\_DECLARE\_OPAQUE\_POINTER(), чтобы зарегистрировать указатель.

В идеальном случае данный макрос должен быть помещён под определением структуры или класса.

Использование данного макроса позволяет сделать частный тип известным всем классам, основанным на шаблонах. Если вы также хотите использовать данный тип в системе сигналов или слотов, следует также вызывать функцию qRegisterMetaType() из класса QMetaType.

Q\_DECLARE\_OPAQUE\_POINTER(). Данный макрос разрешает указатели для объявленных впереди типов.

qRegisterMetaType() регистрирует некоторый тип и возвращает некоторый идентификатор, который использует QMetaType для данного типа. Зарегистрированными могут быть любые классы, которые имеют публичные конструктор по умолчанию, конструктор копирования и деструктор. Функцию нужно вызвать только для зарегистрированных типов. После этого указанный тип может быть использован для создания объектов во время выполнения. Эта функция поточно безопасная.

#### QVariant

<http://qt-project.org/doc/qt-5.1/qtcore/qvariant.html>

QVariant выступает как объединение для наиболее общих типов данных Qt.

Поскольку С++ запрещает включать в объединения типы, которые имеют конструкторов не по умолчанию, наиболее интересные классы Qt не могут быть использованы в объединениях. Без данного класса будут проблемы с функцией QObject::property() (*она может возвращать разные типы данных*), а также с работой с базой данных, и т.д. В одно время данный класс может содержать значение некоторого конкретного типа type(). Используя функцию convert(), можно преобразовать данный тип в другой. Все функции toT() возвращают копию, не затрагивая сам объект.

Также можно сохранять объекты данного типа в контейнерах, создавая сложные структуры с данными разного типа. Т*акже там в конце детального описания указаны некоторые особенности преобразования самого данного типа в другие типы: в некоторых случаях этого сделать не получится.*

Это часть модуля QCore, поэтому QVariant не может обеспечить преобразование в типы данных, определённые в модуле GUI. Поэтому здесь нет функций типа toColor(), но преобразование осуществить можно при помощи функции value(). Обратное преобразование является автоматическим для всех типов данных, поддерживаемых QVariant.

Важными являются функции canConvert() и convert().

#### QMetaObject

<http://qt-project.org/doc/qt-5.1/qtcore/qmetaobject.html#details>

класс QMetaObject содержит мета-информацию об объектах qt. Для каждого класса приложения, являющегося подклассом QObject, создаётся один экземпляр QMetaObject, который содержит информацию о нём. Для получения данной информации следует воспользоваться методом QObject::metaObject(). Этот класс обычно не используется, но он полезен, если вы строите скриптовые движки или среды для разработки графических интерфейсов.

Класс предоставляет множество методов для получения метаинформации о другом классе. Также существует класс QMetaClassInfo, в котором сохраняется дополнительная, созданная пользователем метаинформация о классе в виде пары имя/значение. *В классе описаны также некоторые макросы, изучение которых я пока нахожу излишне детальными: Q\_ARG(), Q\_RETURN\_ARG().*

#### QMetaMethod

<http://qt-project.org/doc/qt-5.1/qtcore/qmetamethod.html>

класс QMetaMethod обеспечивает метаданные о функциях членах.

#### QMetaEnum

<http://qt-project.org/doc/qt-5.1/qtcore/qmetaenum.html>

класс QMetaEnum обеспечивает метаданные о перечислителях.

#### QMetaProperty

<http://qt-project.org/doc/qt-5.1/qtcore/qmetaproperty.html>

класс QMetaProperty обеспечивает метаданные о свойстве. Объекты данного класса могут быть скопированы по значению. Однако каждое значение будет ссылаться на одно и то же свойство.

*Как я понимаю все данные классы содержат информацию о некоторых свойствах объектов qt, которые используются в программе. Обычно, насколько я помню, объекты данных классов получаются при возврате метаданных из объектов класса. А сами эти классы полезны, когда необходимо извлечь некоторую информацию о программных объектах во время выполнения, либо когда нужно создать дополнительные свойства во время выполнения. Также, как я понимаю, для всех особенностей класса, к которым применимы данные классы метаданных, следует применять соответствующие макросы, предоставляемые qt.*

#### QMetaTypeInfo

<http://qt-project.org/doc/qt-5.1/qtcore/qmetaclassinfo.html#details>

класс QMetaTypeInfo предоставляет дополнительную информацию о классе. Это просто пары имя-значение, которые задаются при помощи макроса Q\_CLASSINFO(). Обычно туда можно поместить имя создателя или адреса. *Кстати было бы интересно сделать практику, чтобы каждый разработчик использовал данную возможность, чтобы потом можно было определить, кто создал тот или иной класс.* Сама библиотека нигде не использует данный класс. Он исключительно для вас.

#### QObject

<http://qt-project.org/doc/qt-5.1/qtcore/qobject.html>

класс QObject является базовым классом всех объектов qt. Это сердце объектной модели qt. Данный класс обеспечивает механизм сигналов и слотов. Есть специальные функции для временного запрещения сигналов и слотов, чтобы прерывать нескончаемые циклы, для того, чтоб получать информацию о том, присоединено ли что-либо к данному сигналу и не отсоединено ли от него что-либо (*кстати, очень полезные функции для тестирования*).

QObject организуют сами себя в дерево объектов. Есть также функции, которые возвращают все объекты, которые являются объектами данного класса. (*это также блестящая функция для тестирования*). Также каждый объект имеет своё имя, а также можно при помощи класса мета-объекта вернуть имя класса заданного объекта. Имя объекта можно получать и задавать. Когда объект удаляется, то он высылает сигнал destroyed(), который можно перехватить, чтобы убрать висячие ссылки. События доставляются в том же потоке, в котором объект был создан. Следует иметь в виду, что обработка событий не происходит, если функция thread() возвращает нуль. Также есть функции, которые позволяют изменять поток существования объектов.

Макрос Q\_OBJECT является обязательным для предоставления всех возможностей qt. Рекомендуется использовать данный макрос во всех подклассов QObject независимо от того, имеют ли они сигналы или слоты, так как некоторые функции тогда будут вести себя немного странно.

Также есть функция, которая возвращает информацию о том, является ли данный объект действительно виджетом. Она намного быстрее, чем qobject\_cast<>().

В данном классе нет ни конструктора копирования, ни оператора присваивания. Это из-за дизайна. Они объявлены, но в закрытой части с макросом Q\_DISABLE\_COPY(). Все подклассы также используют данный макрос, чтобы задекларировать данные методы закрытыми. О причинах данного решения было сказано ранее.

Очень классным методом является такой: [QMetaObject::connectSlotsByName](http://qt-project.org/doc/qt-5.1/qtcore/qmetaobject.html#connectSlotsByName)(). Данный метод в качестве параметра имеет некоторый объект QObject, слот которого, написанный в соответствии с конвенцией об именах слотов, соединяется с соответствующим сигналом всех его дочерних объектов автоматически.

К объектам данного класса могут во время выполнения добавляться и удаляться динамические свойства. Для их считывания и установки значения используются все те же функции, которые используются для свойств, объявленных во время компиляции. *Данные методы объявлены в данной части.*

Операция tr() позволяет все объекты данного класса делать переводимыми на другие языки.

#### QObjectCleanUpHandler

<http://qt-project.org/doc/qt-5.1/qtcore/qobjectcleanuphandler.html#details>

QObjectCleanUpHandler класс следит за временем жизни множества объектов QObject. Данный класс полезен всякий раз, когда вам необходимо знать, когда некоторое количество объектов QObject, которыми владеет некто, также были удалены. Это важно, например, когда ссылаешься на память в приложении, которая была динамически выделена в разделяемой библиотеке.

Для слежения за некоторыми объектами следует создать данный класс и добавить в него объекты, которыми вы интересуетесь. Если объект удаляется, то он автоматически удаляется и из данного объекта. Также можно объекты убирать из него самостоятельно.

#### QPointer

<http://qt-project.org/doc/qt-5.1/qtcore/qpointer.html#details>

QPointer – это шаблонный класс, который обеспечивает охраняемые указатели на объекты QObject. Он ведёт себя как обычный указатель C++ за исключением того, что автоматически устанавливается в нуль, если соответствующий объект удаляется. *В описании класса есть некоторые особенности для поведения данного класса, которые были введены в библиотеке qt5.* Также qt предоставляет класс QSharedPointer, который представляет из себя объект указателя с возможностью подсчёта ссылок, который может быть использован для удержания коллекции ссылок на один указатель.

Охраняемые указатели можно передавать функциям, которые требуют обычный указатель. То есть обычные указатели и охраняемые указатели являются взаимозаменяемыми. Обязательным для данного класса является условие того, чтобы класс, на который он ссылается был бы подклассом QObject.

#### QSharedPointer

<http://qt-project.org/doc/qt-5.1/qtcore/qsharedpointer.html>

класс QSharedPointer удерживает сильную ссылку на разделяемый указатель. Это автоматический разделяемый указатель в С++. Он ведёт себя точно также как и обычный указатель для обычных целей. Данный класс будет удалять указатель, который он удерживает, когда тот выходит из области видимости, обеспечивая, чтобы ни один другой объект QSharedPointer не мог ссылаться на него. Данный объект можно создать из обычного указателя, из такого же разделяемого указателя, или при помощи продвижения слабого указателя.

Данный класс является поточно безопасным и работает автоматически со значением указателя. Разные потоки могут обращаться к данному классу, указывающему на один и тот же объект, не используя механизм блокировки.

Также qt предоставляет два других обёрточных класса для указателей: QPointer и QSharedDataPointer. Они несовместимы друг с другом, так как каждый из них имеет разные случаи использования.

Данный класс удерживает разделяемый указатель при помощи внешнего счётчика ссылок (счётчик находится вне объекта). Исходя из названия, указатель разделяется между всеми экземплярами классов QSharedPointer и QWeakPointer. Содержание объекта, на который указывает указатель, не рассматривается как разделяемое, однако существует только один объект. Поэтому класс QSharedPointer не обеспечивает способ отделения или копирования указываемого объекта.

QSharedDataPointer удерживает указатель на разделяемые данные (класс, производный от DeruvedData). Он делает это при помощи внутреннего счётчика ссылок. Поэтому данный класс может отделить основанные на типе доступа данные, которые охраняются: если доступ не константный, он создаёт копию автоматически для завершения операций.

QScopedPointer просто удерживает яму, которая динамически выделит память для объекта и удаляет его в деструкторе. Этот класс полезен тогда, когда объект должен быть создан в яме и удалён автоматически, но не более. Этот указатель очень прост, очень лёгок. (*как я понял, это просто избавление программиста от написания ключевых слов new и delete*). Он лёгок, так как не использует дополнительную структуру для подсчёта ссылок.

Класс QPointer удерживает указатель на производный класс от QObject, но он делает это слабо. QWeakPointer имеет такую же функциональность, но его использование для этой функции устарело.

Особенность класса QSharedPointer, которая может быть доступна во время компиляции для целей отладки – это механизм отслеживания. Когда доступен, данный класс регистрирует в Глобальной коллекции все указатели, за которыми он следит. Это позволяет ему словить ошибки, например, присваивание одного и того же указателя двум объектам QSharedPoiner. Эта функция доступна при объявлении QT\_SHAREDPOINTER\_TRACK\_POINTERS макроса перед включением заголовочного файла QSharedPointer. *Также указан случай, когда данный механизм работает неправильно, но я не понял, о чём идёт речь. Разберусь в будущем.*

*Я пока понимаю данный класс так: есть один указатель, который может быть «захвачен многими» экземплярами данного класса. Если все они удаляются, то удаляется и сам указатель. Причём, как я понимаю, выполнять присваивание двум экземплярам данного класса нельзя.*

*Мне следует экспериментально поработать над данным классом. Со всеми непонятными для меня аспектами.*

#### QWeakPointer (устарел)

<http://qt-project.org/doc/qt-5.1/qtcore/qweakpointer.html#details>

класс QWeakPointer (устарел) является слабой ссылкой на разделяемый указатель. Это автоматическая слабая ссылка в С++. Она не может разыменовать указатель, но она позволяет проверить, был ли удалён указатель или нет в другом контексте. Объекты данного класса могут быть созданы только при помощи присваивания из QSharedPointer.

Класс не может рассматриваться как указатель сам по себе, так как он не может гарантировать, что объект доступен. Поэтому для доступа к данному объекту для начала следует продвинуть данный класс в QSharedPointer и проверить, существует ли объект или нет. Также данный класс может возвращать указатель, но он не гарантирует, что он будет доступен.

QScopedPointer (устарел)

<http://qt-project.org/doc/qt-5.1/qtcore/qscopedpointer.html#details>

QScopedPointer (устарел) сохраняет указатель на динамически сохранённый объект и удаляет его в деструкторе. Класс не имеет конструктора копирования и оператора присваивания. Классы, которые предварительно объявлены, могут быть использованы с данным типом указателя. *Но для них должны выполняться особенные требования, которые описаны в данной части.* Данный класс устарел. Для QObject лучше использовать класс QPointer.

#### QSharedDataPointer

<http://qt-project.org/doc/qt-5.1/qtcore/qshareddatapointer.html>

QSharedDataPointer представляет указатель на неявно разделяемый объект. QSharedDataPointer<T> делает лёгким написание ваших собственных неявно разделяемых классов. Данный класс является поточно безопасным при подсчёте ссылок.

Неявное разделение используется многими классами qt, чтобы объединить скорость и эффективность памяти указателей с лёгкостью использования классов.

Для того чтобы создать неявно разделяемый класс, следует задать в нём член данных QSharedDataPointer<T>, а класс Т должен быть производным классом от QSharedData.

*В данной части есть некоторый демонстрационный код.*

Данный класс работает так. Предположим, что вы имеете некоторый один член данных у многих классов, который является разделяемыми данными. Если вы в одном классе изменяете данный член, то автоматически создаётся его копия, чтобы это не затронуло остальные классы-владельцы этих данных.

Если в классе владельце находится такой указатель, то все обращения к разделяемому объекту происходят через него. Если при этом будет записываться некоторая информация в разделяемый объект, то оператор ->() автоматически вызовет операцию detach(), которая создаст копию разделяемого объекта, если счётчик разделяемого объекта будет более одного. Это гарантирует, что запись в один объект владелец не повлияет на любые другие данные в другом объекте-владельце, которые разделяют тот же самый объект данных.

Класс разделяемых данных имеет конструктор по умолчанию, конструктор копирования и деструктор. Обычно это всё, что необходимо для класса данных для неявного разделения. Здесь под классом данных понимается и тот класс, который хранит указатель на разделяемые данные, и тот класс, где непосредственно находятся сами данные, *как я понял из примера.*

Также далее идёт разговор о закрытых и открытых классах, о публичных и нет файлах.

Что такое закрытые заголовочные файлы???

*Как это всё реализуется. Также в конце идёт понятная и очень интересная часть, посвящённая потенциальным проблемам при работе с данным классом.*

#### QExplicitlySharedDataPointer

<http://qt-project.org/doc/qt-5.1/qtcore/qexplicitlyshareddatapointer.html>

класс QExplicitlySharedDataPointer класс представляет указатель на явно разделяемый объект.

Данный класс позволяет вам писать ваш собственный явно разделяемый класс легко. Он поточно безопасный. За исключением одного большого недостатка. Операции данного класса не производят автоматическое копирование члена данных при его перезаписывании. Но функция detach() там всё равно доступна, однако вам придётся использовать её самостоятельно. Они ведут себя как обычные указатели С++ за тем исключением, что не оставляют висячих ссылок при удалении объекта.

*Вы явно сами определяете, как данный класс копируется при изменении.*

Также отмечается, что есть функции отладки [QObject::dumpObjectTree](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html#dumpObjectTree)() and [QObject::dumpObjectInfo](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html#dumpObjectInfo)(), которые очень полезны, когда приложение ведёт себя странно.

### СИСТЕМА ВВОДА/ВЫВОДА

<http://qt-project.org/doc/qt-5.1/qtcore/io.html>

класс QImageReader предоставляет независимый от формата интерфейс для считывания изображений из файлов или других устройств.

класс QImageWriter обеспечивает независимый от формата интерфейс для записи изображений в файлы и а другие устройства.

класс QTextDocumentWriter обеспечивает независимый от формата интерфейс для записи [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html) в файлы или на другие устройства.

Класс QFileDevice обеспечивает интерфейс для чтения и записи открытых файлов. Это базовый класс для QFile, а также некоторых других файловых классов.

класс QFileInfo обеспечивает независимую от системы информацию о файле: размер, положение в файловой системе и другие данные о файле.

класс QFileSystemWatcher обеспечивает интерфейс для мониторинга файлов и директорий на наличие модификаций.

класс QIODevice является базовым интерфейсным классом для всех устройств ввода/вывода в qt.

класс QResource обеспечивает интерфейс для чтения напрямую из ресурсов. Это объект, который представляет набор данных, относящихся к единичной сущности ресурса.

класс QTemporaryDir создаёт уникальную директорию для временного пользования. Он делает это безопасно. Имя новой директории гарантированно будет уникальным, и она удалится при удалении объекта данного класса.

класс QSaveFile обеспечивает интерфейс для записи текстовых и двоичных файлов, без потери существующих данных, если операция записи отказывает. *Всегда используйте данный файл при записи полных документов на диск.*

класс QTemporaryFile является устройством ввода/вывода, которое управляет временными файлами.

класс QUrl обеспечивает удобный интерфейс для работы с url. Он может анализировать и строить url как в закодированной, так и в незакодированной форме. Также данный класс имеет поддержку интернационализированных имён доменов.

класс QUrlQuery обеспечивает способ манипулирования парами ключ-значение в запросе url. Это используется для анализа запроса, который находится в url.

класс QSocketNotifier обеспечивает поддержку для мониторинга активности дескриптора файла.

класс QFile обеспечивает интерфейс для считывания и записи файлов. Это инструмент для считывания текстовых файлов, двоичных файлов и ресурсов. Этот класс может быть использован сам по себе или, что более удобно, с классами QTextStream или QDataStream.

класс QTextStream предоставляет удобный интерфейс для чтения и записи текста. QTextStream конвертирует восьмибитные данные, сохранённые на диске в 16-битную юникод строку. При этом по умолчанию предполагается, что пользователь использует восьмибитную локальную кодировку. Для изменения этого используйте класс QTextCodec. Также данный класс использует буфер, из-за чего вам не следует считывать информацию из потока с использованием функциональности его суперкласса, например QFile, так как например положение в потоке и положение в файле будут отличаться.

класс QDataStream обеспечивает сериализацию двоичных данных на QIODevice.

QBuffer класс предоставляет интерфейс QIODevice для класса QByteArray. Массив обрабатывается как обычный файл с произвольным доступом.

QByteArray обеспечивает массив байтов.

класс QDir (*устарел?*) предоставляет доступ к структурам директорий их содержанию.

QBitArray – это класс, который обеспечивает массив битов. Это массив, который даёт доступ к отдельным битам и обеспечивает выполнение над ними логических операций.

QBitArray – это класс, который обеспечивает массив битов. Это массив, который даёт доступ к отдельным битам и обеспечивает выполнение над ними логических операций.

#### QImageReader

<http://qt-project.org/doc/qt-5.1/qtgui/qimagereader.html>

класс QImageReader предоставляет независимый от формата интерфейс для считывания изображений из файлов или других устройств. Этот класс даёт больший контроль при считывании изображений, нежели [QImage](http://qt-project.org/doc/qt-5.1/qtgui/qimage.html) and [QPixmap](http://qt-project.org/doc/qt-5.1/qtgui/qpixmap.html) классы. Например, можно изменить размер изображения или считать только его часть. *Далее показано, как пользоваться данным классом*. Есть функция для получения информации об ошибке при считывании. Данный класс поддерживает все встроенные форматы изображений, а также форматы от плагинов, в которых обеспечено чтение. Также можно разрешить или запретить автоматическое детектирование формата файла.

#### QImageWriter

<http://qt-project.org/doc/qt-5.1/qtgui/qimagewriter.html#details>

класс QImageWriter обеспечивает независимый от формата интерфейс для записи изображений в файлы и а другие устройства. Он поддерживает настройки, зависимые от формата: гамма уровень, уровень сжатия и уровень качества. Если вам не нужны такие настройки, то воспользуйтесь [QImage::save](http://qt-project.org/doc/qt-5.1/qtgui/qimage.html#save)() или [QPixmap::save](http://qt-project.org/doc/qt-5.1/qtgui/qpixmap.html#save)() вместо вместо данного класса.

#### QTextDocumentWriter

<http://qt-project.org/doc/qt-5.1/qtgui/qtextdocumentwriter.html#details>

класс QTextDocumentWriter обеспечивает независимый от формата интерфейс для записи [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html) в файлы или на другие устройства. Для записи следует построить объект данного класса с именем файла или устройства и определить формат документа для записи. Можно получать информацию об ошибках. *Следует экспериментально изучить данный класс, который представляет для меня довольно большой интерес.*

#### QFileDevice

<http://qt-project.org/doc/qt-5.1/qtcore/qfiledevice.html#details>

Класс QFileDevice обеспечивает интерфейс для чтения и записи открытых файлов. Это базовый класс для QFile, а также некоторых других файловых классов.

#### QFileInfo

<http://qt-project.org/doc/qt-5.1/qtcore/qfileinfo.html#details>

класс QFileInfo обеспечивает независимую от системы информацию о файле. Он обеспечивает информацию об имени и позиции файла в файловой системе, права доступа к нему и является ли он директорией или символической ссылкой. Также доступны размер файла и время последней модификации/чтения. Также данный класс может быть использован для получения информации о ресурсе qt. Есть функции для работы с разными типами путей. На юникс системе информация о символической ссылке совпадает с информацией о файле, на который она ссылается. На Windows вся информация соответствует файлу ссылки, а не целевому файлу. *Функции данного класса, которые не обращаются к файловой системе, работают быстрее.*

Также для ускорения работы данный класс кэширует информацию о файле. Но файлы могут изменяться. Поэтому есть функция, которая обновляет информацию о файле. Также можно отменить кэширование.

#### QFileSystemWatcher

<http://qt-project.org/doc/qt-5.1/qtcore/qfilesystemwatcher.html#details>

класс QFileSystemWatcher обеспечивает интерфейс для мониторинга файлов и директорий для модификаций. Класс проводит мониторинг файловой системы на изменения файлов и директорий при помощи просмотра списка определённый путей. Пути могут добавляться и удаляться. Есть сигнал, который высылается, когда некоторый файл был модифицирован или удалён из системы. То же самое есть и для директории. Слежение за файлами останавливается при их удалении или переименовании. *Есть некоторое замечание по поводу Linux, но которое я не понял.* Действие слежения потребляет ресурсы системы. Некоторые системы, например, разрешают одновременное открытие лишь фиксированного максимального количества файлов. Если вы захотите добавить файл ещё, то функция выйдет из строя.

#### QIODevice

<http://qt-project.org/doc/qt-5.1/qtcore/qiodevice.html#details>

класс QIODevice является базовым интерфейсным классом для всех устройств ввода/вывода в qt. Он обеспечивает базовый абстрактный интерфейс и общее определение для устройств, которые поддерживают чтение и запись блоков данных. Данный класс нельзя инстанцировать, но он используется многими другими классами, которые требуют представления на различных устройствах ввода/вывода. Данный класс различает два типа устройств:

* произвольного доступа и
* последовательного доступа.

Некоторые операции данного класса являются асинхронными, и это не всегда хорошо. *В данной части они перечислены и есть пример кода.* Некоторые подклассы данного класса используют буфер.

#### QResource

<http://qt-project.org/doc/qt-5.1/qtcore/qresource.html#details>

класс QResource обеспечивает интерфейс для чтения напрямую из ресурсов. Это объект, который представляет набор данных, относящихся к единичной сущности ресурса. Класс даёт прямой доступ к байтам в их сыром формате. Это позволяет работать с ресурсом без буфера или перенаправления. Перенаправление полезно, когда с ресурсом следует работать как с файлом. В данном случае следует использовать класс QFile. Обычно данные в данном классе компилируются в приложение/библиотеку. Но также возможно загрузить ресурс во время выполнения. Когда загружается во время выполнения, файл ресурса будет загружен как один большой набор данных и затем будет дан в виде частей через ссылки на дерево ресурса. Загружать ресурс можно или через абсолютный путь или через относительный.

*Также сказана некоторая информация о возможности разжатия файлов. Есть некоторая функция, которая используется для доступа к реальным данным, если данные сжаты.* Обращение к данной функции происходит неявно через доступ из класса QFile. Ресурс, который представляет директорию, может иметь только потомков, но не данные. Есть функция для загрузки ресурса во время выполнения. Файл, который передаётся в качестве аргумента данной функции, должен быть двоичным файлом ресурса (rcc). *Также в конце есть некоторая информация по поводу динамической загрузки ресурсов. Я не понял данной информации.*

#### QSaveFile

<http://qt-project.org/doc/qt-5.1/qtcore/qsavefile.html#details>

класс QSaveFile обеспечивает интерфейс для записи текстовых и двоичных файлов, без потери существующих данных, если операция записи отказывает. *Всегда используйте данный файл при записи полных документов на диск.* Во время записи содержание будет записано во временный файл и если не случится ошибки, то при помощи специальной операции содержание можно поместить в конечный файл. Есть функции для отмены записи при проблемах с приложением. Также, если объект класса был уничтожен, то временный файл отбрасывается.

#### QTemporaryDir

[*http://qt-project.org/doc/qt-5.1/qtcore/qtemporarydir.html#details*](http://qt-project.org/doc/qt-5.1/qtcore/qtemporarydir.html#details)

класс QTemporaryDir создаёт уникальную директорию для временного пользования. Он делает это безопасно. Имя новой директории гарантированно будет уникальным, и она удалится при удалении объекта данного класса. Имя директории либо генерируется автоматически, либо создаётся на основе шаблона вручную. Также очень важно протестировать, что временная директория была создана. Но при этом важно, чтобы использовалась именно функция isValid(), а не exists(), так как последняя представляет текущую директорию.

*Также в данной части в конце речь идёт о формировании имени временной директории.*

#### QTemporaryFile

<http://qt-project.org/doc/qt-5.1/qtcore/qtemporaryfile.html#details>

класс QTemporaryFile является устройством ввода/вывода, которое управляет временными файлами. Описание данного класса подобно описанию класса QTemporaryDir.

#### QUrl

<http://qt-project.org/doc/qt-5.1/qtcore/qurl.html#details>

класс QUrl обеспечивает удобный интерфейс для работы с url. Он может анализировать и строить url как в закодированной, так и в незакодированной форме. Также данный класс имеет поддержку интернационализированных имён доменов. Url может отображаться в закодированном и незакодированном виде. Незакодированный вид – это то, что обычно видят пользователи, тогда как закодированный вид обычно используется для отправления на веб-сервер. Также данный класс обеспечивает функции для построения url часть за частью. Для кодирования и раскодирования url используются специальные функции. *Также в данной части перечислены все стандарты, которых придерживается данный класс.* Данный класс способен обнаруживать много ошибок в url во время анализа или когда компоненты url установлены при помощи индивидуальных методов сеттеров. *Далее в данном классе содержится информация об детектировании ошибок в url, которую тяжело воспринимать без знаний в данной предметной области.* Также при создании QString, содержащей url, из QByteArray или char\* всегда вызывайте QString::fromUtf8().

#### QUrlQuery

<http://qt-project.org/doc/qt-5.1/qtcore/qurlquery.html#details>

класс QUrlQuery обеспечивает способ манипулирования парами ключ-значение в запросе url. Это используется для анализа запроса, который находится в url. *В данной части есть иллюстративный пример.* Запросы используются для передачи настроек в url и обычно декодируются в множественные пары ключ-значение. *Также в данной части рассматриваются некоторые вопросы кодировки. Также рассматриваются специфические для данной области вопросы кодирования знаков пробела.* По умолчанию данный класс использует знак «=» для отделения ключа от своего значения, а знак “&”для отделения пар ключ-значение друг от друга. Можно задавать собственные символы разделителей. *Сказано пользоваться для выбора нестандартных разделителей следующим стандартом* RFC 3986. *Прежде использования данного класса мне следует более подробно изучить такую тему, как url.*

#### QSocketNotifier

[*http://qt-project.org/doc/qt-5.1/qtcore/qsocketnotifier.html#details*](http://qt-project.org/doc/qt-5.1/qtcore/qsocketnotifier.html#details)

класс QSocketNotifier обеспечивает поддержку для мониторинга активности дескриптора файла. Данный класс делает возможным интеграцию цикла обработки событий qt с другими циклами обработки событий, основанными на мониторинге дескриптора. Действия дескриптора файла детектируются в цикле обработки событий приложения. Открыв однажды устройство с использованием низкоуровневого программного интерфейса, вы можете создать сокет уведомитель, чтобы проводить мониторинг дескриптора файлов. Есть три типа сокетов уведомителей: чтение, запись и исключение. Тип должен быть определён при построении сокета. Невозможно установить два уведомителя сокета одинакового типа для одного и того же сокета. Также в данной части советуют всегда явно разрешать или запрещать конкретный уведомитель сокета. Также можно получить идентификатор сокета при помощи специальной функции. Сокет, переданный данному классу, станет неблокирующимся, даже если он был создан как блокирующийся сокет. *Также в последней части есть некоторая специфическая информация по использованию данного класса в ОС Windows. Об этом я почитаю подробнее, когда изучу, что такое сокет, и захочу использовать данный класс в своих приложениях.*

#### QFile

<http://qt-project.org/doc/qt-5.1/qtcore/qfile.html#details>

класс QFile обеспечивает интерфейс для считывания и записи файлов. Это инструмент для считывания текстовых файлов, двоичных файлов и ресурсов. Этот класс может быть использован сам по себе или, что более удобно, с классами QTextStream или QDataStream. Имя файла обычно передаётся через конструктор, но может быть установлено и при помощи специальной функции. *Также данный класс ожидает разделителя «/» независимо от операционной системы.* Класс обеспечивает средства для проверки существования файла, для его удаления, открытия, закрытия, а также flush(). Данные обычно записываются и считываются при помощи QDataStream и QTextStream, но вы можете также использовать QIODevice-наследованные методы. Также данный класс наследует методы, работающие с отдельными символами. Класс обеспечивает различные функции навигации по файлу. По умолчанию класс работает с файлом как с двоичным файлом. *В классе показан пример построчного считывания файла.* Для считывания файла можно использовать потоки.

Все классы qt используют названия в кодировке Unicode. Если вы хотите воспользоваться другими средствами работы с файлами, например, стандартными средствами С++, которые используют восьмибитные кодировки, то вам необходимо использовать функции encodeName() и decodeName(). Ссылки на них указаны в описании данного класса.

*В системе Unix есть некоторые файлы, которые ведут себя по-особенному. Они описаны в описании к данному классу*. Также данный класс вовсе не подходит для считывания некоторых файлов, таких как файлы устройств в системе Unix. Также файлы зависят от платформы. Например, в Linux в некоторые директории нельзя записывать файлы.

#### QTextStream

<http://qt-project.org/doc/qt-5.1/qtcore/qtextstream.html#details>

класс QTextStream предоставляет удобный интерфейс для чтения и записи текста. QTextStream конвертирует восьмибитные данные, сохранённые на диске в 16-битную юникод строку. При этом по умолчанию предполагается, что пользователь использует восьмибитную локальную кодировку. Для изменения этого используйте класс QTextCodec. (<http://qt-project.org/doc/qt-5.1/qtcore/qfile.html#details>).

Данный класс может работать на QIODevice, QByteArray или QString. Вы можете средствами данного класса удобно считывать слова, строки и числа. Этот класс обладает функциональностью форматирования текста, а также чисел. Также данный класс используют для консоли. Он осведомлён о локализации и автоматически декодирует стандартный ввод, используя правильный кодек. Также можно установить прибор или строку, с которыми работает данный класс. Также данный класс обеспечивает функции навигации над файлом и т.д. Этот класс внутренне использует QTextCodec, чтобы автоматически поддерживать разные наборы символов. Но установить кодек можно также и самостоятельно. Когда данный класс работает с QString, то кодек становится недоступным.

Есть три общих способа использования данного класса при чтении файла:

* Кусочек за кусочком (readLine(), readAll());
* Слово за словом, причём слова отделяются пробелами, а передний пробел автоматически отбрасывается.
* Символ за символом. Этот метод часто используется для удобного управления анализом файла.

Также данный класс использует буфер, из-за чего вам не следует считывать информацию из потока с использованием функциональности его суперкласса, например QFile, так как например положение в потоке и положение в файле будут отличаться. Данный класс автоматически детектирует систему счисления чисел, но её можно также устанавливать вручную. Данный класс обеспечивает также некоторые манипуляторы.

#### QInputMethodEvent::Attribute

[*http://qt-project.org/doc/qt-5.1/qtgui/qinputmethodevent-attribute.html#details*](http://qt-project.org/doc/qt-5.1/qtgui/qinputmethodevent-attribute.html#details)

*класс* QInputMethodEvent::Attribute сохраняет атрибуты метода ввода.

#### QDataStream

<http://qt-project.org/doc/qt-5.1/qtcore/qdatastream.html#details>

класс QDataStream обеспечивает сериализацию двоичных данных на QIODevice. Поток данных – это двоичный поток закодированной информации, который на 100 % независим от операционной системы компьютера, центрального кода или кода байтов. Поэтому потоки данных, написанные на одной платформе, могут быть использованы и на другой платформе тоже.

Данный класс обеспечивает сериализацию базовых классов типов данных C++. Выполнение сериализации для более сложных типов выполняется разбиением последних на примитивные типы. Данный класс тесно сотрудничает с классом QIODevice, который представляет I/O среду, которая может считывать и записывать данные. QFile класс – это пример I/O устройства. *В данной части указаны примеры чтения и записи двоичных файлов из потока и в поток.* Каждый элемент, записанный в поток, записывается в предопределённом двоичном формате, который изменяется в зависимости от типа элемента. Qt поддерживает сериализацию многих типов, которые перечислены в <http://qt-project.org/doc/qt-5.1/qtcore/datastreamformat.html>.

Для целых чисел лучше всего всегда использовать типы целых для Qt, так как это гарантирует, что вы получаете целые того размера, который вы желаете и изолирует вас от отличий платформ и компиляторов.

Также данный класс позволяет создавать устройство для считывания данных.

Двоичный формат Qt развивается вместе с самой средой разработки. Поэтому важно обеспечить, чтобы для считывания и записи одной и той же информации использовались одни и те же версии qt. В приложении также можно установить версию. *Как это сделать, показано в данной части.*

Если вы создаёте специальные структуры данных, пригодные для вашего приложения, то вам следует писать в файле специальный заголовок, который бы содержал магическое число, а также версию. Это даст вам возможность к дальнейшему расширению вашей системы. При считывании данного файла в следующий раз следует вначале проверять данные индикаторы совместимости, после чего только приступать к обработке данного файла.

*Вы также можете изменить порядок байтов при сериализации, но делать этого не рекомендуется.*

Вы также можете пожелать записать ваши собственные данные в поток данных напрямую. Данные могут быть считаны из потока в некоторый выделенный массив символов. Подобно этому данные могут быть записаны в поток при помощи другой функции writeRawData(). Но заметьте, что любое кодирование/декодирование данных должно быть сделано вами. Также сериализованы могут быть контейнерные классы.

Вдобавок к перегруженным операторам потоков любой класс Qt, который вы желаете сериализовать к потоку данных, будет иметь подходящие потоковые операторы, объвленные как не члены данного класса.

*Вообще класс полезный. Он позволяет переводить любую структуру данных в двоичный формат, а затем считывать её в приложение, возможно, выполняя кодирование и декодирование.*

#### СЕРИАЛИЗАЦИЯ ТИПОВ ДАННЫХ

<http://qt-project.org/doc/qt-5.1/qtcore/datastreamformat.html>

класс QDataStream позволяет сериализовать некоторые из типов данных qt. Список в данной части показывает эти данные, а также то, как они представляются.

#### QBuffer

<http://qt-project.org/doc/qt-5.1/qtcore/qbuffer.html#details>

QBuffer класс предоставляет интерфейс QIODevice для класса QByteArray. Массив обрабатывается как обычный файл с произвольным доступом. Массив можно передавать буферу, также как и устанавливать буфер для массива. По умолчанию у буфера уже есть свой внутренний массив. Также буфер можно использовать для записи различной информации в массив при помощи потока данных. При этом массив передаётся буферу, а буфер – соответствующему потоку. Буфер также высылает некоторые сигналы каждый раз, как в него поступают или записываются новые данные.

#### QByteArray

<http://qt-project.org/doc/qt-5.1/qtcore/qbytearray.html#details>

QByteArray обеспечивает массив байтов. Он может быть использован для сохранения как raw bytes (включая «\0»), так и традиционные 8-битные, «\0» terminated bytes. Использование данного класса значительно более удобнее, чем использование const char \*. Кроме прочего он всегда гарантирует, что данные заканчиваются ограничителем «\0» и использует неявное разделение (копирование при записи), чтобы уменьшить использование памяти и избежать самого копирования данных.

*Плохо понимаю, чем отличаются два приведенных выше типа байтов.* Вдобавок к данному классу qt также предоставляет класс QString, чтобы хранить строковые данные. Данный класс хранит 16-битные символы в кодировке Unicode, делая возможным легко сохранять в вашем приложении символы не из кодировок ASCII/Latin-1.

Есть два важных случая, когда вам необходимо использовать данный класс QByteArray:

* чтобы хранить двоичные данные и
* когда сохранение памяти критично.

Теперь я понял, в чём суть ограничителя. Данные класс предоставляет возможность, которая реализуется по умолчанию, заканчивать любой массив const char \* символом ограничителем «/0». Если затем создать указатель, который будет указывать на данный массив, то будет гарантированно, что данные, на который он указывает, ограничены символом-ограничителем. Данный класс совершает глубокое копирование данных const char \*, так что вы можете модифицировать их позднее, без ожидания побочных эффектов.

Также есть оператор at(), который также предоставляет доступ к членам массива байтов, но при этом работает быстрее, так как не выполняет глубокое копирование. Массив байтов может включать и ограничительные символы. *В данной части показано, как сделать так, чтобы они воспринимались, как обычные символы, а не как окончание строки.*

Для сортировки строк интерфейса пользователя наиболее подходит функция QString::localeAwareCompare().

Различают нулевой массив байтов и пустой массив байтов. Нулевой массив байтов –это такой массив байтов, который инициализируется строкой (const char\*)0. Пустой массив байтов – это массив, размер которого равен 0. Нулевой массив байтов всегда является пустым, но пустой массив байтов не всегда является нулевым. Рекомендуется всегда избегать использования функции isNull(). Для использования преобразований строк и чисел из-за локализации лучше годится класс QString.

*Не понял раздел* ***8-bit Character Comparison***

QDir (*устарел?*)

<http://qt-project.org/doc/qt-5.0/qtcore/qdir.html>

класс QDir (*устарел?*) предоставляет доступ к структурам директорий их содержанию. Класс используется для управления путями, доступом к файлам, а также управлением файловой системой. Здесь также поддерживаются абсолютные и относительные пути. Для объекта данного класса можно изменять директорию, создавать директорию или удалять директорию, а также переименовывать её. Также можно использовать функцию refresh() для обновления информации о директории. Директории содержат некоторое содержание, состоящее из файлов, директорий и символических ссылок. Можно просматривать содержимое директории, применять разные фильтры для его отображения. Также в данном классе содержатся некоторые статические функции, которые обеспечивают доступ к некоторым специфическим директориям. Также в классе предусмотрен набор функций для обработки имени пути, например, приведение пути к виду, специфическому для данной платформы и т.д.

#### QBitArray

<http://qt-project.org/doc/qt-5.1/qtcore/qbitarray.html#details>

QBitArray – это класс, который обеспечивает массив битов. Это массив, который даёт доступ к отдельным битам и обеспечивает выполнение над ними логических операций. Он также использует неявное разделение, как и класс QByteArray. Для доступа и проверки битов следует использовать функции setBit() и testBit(). Этот класс поддерживает все логические операции, как и обычный язык С++. *Примеры использования данных операций показаны в данной части.*Также здесь есть нулевой массив (который инициализирован при помощи конструктора по умолчанию) и пустой массив (длина которого равняется нулю). Также нужно избегать использования функции isNull().

### КОНТЕЙНЕРНЫЕ КЛАССЫ

[*http://qt-project.org/doc/qt-5.1/qtcore/containers.html*](http://qt-project.org/doc/qt-5.1/qtcore/containers.html)

qt обеспечивает контейнерные шаблонные классы. Они разработаны, чтобы быть легче, безопаснее и проще в использовании, чем контейнеры STL. Все контейнерные классы являются **неявно разделяемыми**, реентерабельными, а также они оптимизированы по скорости, затратам памяти, количеству расширения кода, который в конченом счёте приводит к меньшим в размере исполняемым файлам. Также они являются потокобезопасными, когда они используются только для чтения.

Для получения элементов контейнеров можно использовать два типа итераторов:

* в java стиле и
* в стл стиле.

Первый тип итераторов является более лёгким в использовании и обеспечивает высокоуровневую функциональность, тогда как второй тип итераторов является более эффективным и совместим с алгоритмами qt и стл. Также qt обеспечивает ключевое слово foreach, которое позволяет легко обходить элементы в контейнере.

Qt обеспечивает следующие последовательные контейнерные классы: [QList](http://qt-project.org/doc/qt-5.1/qtcore/qlist.html), [QLinkedList](http://qt-project.org/doc/qt-5.1/qtcore/qlinkedlist.html), [QVector](http://qt-project.org/doc/qt-5.1/qtcore/qvector.html), [QStack](http://qt-project.org/doc/qt-5.1/qtcore/qstack.html), and [QQueue](http://qt-project.org/doc/qt-5.1/qtcore/qqueue.html). Для большинства приложений QList является наиболее лучшим типом контейнера. Также qt обеспечивает следующие ассоциативные контейнерные классы: [QMap](http://qt-project.org/doc/qt-5.1/qtcore/qmap.html), [QMultiMap](http://qt-project.org/doc/qt-5.1/qtcore/qmultimap.html), [QHash](http://qt-project.org/doc/qt-5.1/qtcore/qhash.html), [QMultiHash](http://qt-project.org/doc/qt-5.1/qtcore/qmultihash.html), and [QSet](http://qt-project.org/doc/qt-5.1/qtcore/qset.html). Мульти контейнеры поддерживают наличие множества значений для одного и того же ключа. Контейнер хэш обеспечивает более быстрый поиск, используя хэш функцию вместо двоичного поиска в сохранённом ряде. Также есть дополнительные классы [QCache](http://qt-project.org/doc/qt-5.1/qtcore/qcache.html) and [QContiguousCache](http://qt-project.org/doc/qt-5.1/qtcore/qcontiguouscache.html), которые позволяют выполнять быстрый поиск объектов в ограниченных КЭШах.

Следует обратить внимание, что основное отличие мэпа от хэша заключается в том, что первый сохраняет значения в порядке ключей, а второй в некотором внутреннем произвольном порядке.

Контейнеры могут быть вложенными. При этом необходимо между одинаковыми угловыми скобками ставить пробел, чтобы компилятор С++ не интерпретировал это как оператор >>.

Все члены контейнеров должны иметь один и тот же присваиваемый тип данных. Чтобы это гарантировать, тип должен обеспечивать конструктор по умолчанию, конструктор копирования и оператор присваивания. Для сохранения виджетов в контейнерах их следует сохранять как указатели (обычно выскакивает ошибка об отсутствии конструктора копирования и оператора присваивания). *В данной части есть пример объявления кода класса, который может использоваться в качестве элемента контейнера.*

Некоторые контейнеры предъявляют дополнительные требования к типу данных. В мэпе (в ключе) должен быть оператор <(). Также некоторые специальные функции имеют некоторые специальные требования. Также контейнеры обеспечивают операторы <<() и >>() для использования считывания и записи данных контейнера классами QDataStream, но для этого следует перегрузить в классах типов данных эти операторы (*есть пример кода*). Также в некоторых классах есть значения по умолчанию, которые используются в некоторых конкретных случаях. Для примитивных типов данных данное значение равняется 0.

Итераторы обеспечивают однородные средства для доступа к элементам контейнера. *Итераторы аннулируются, если данные в контейнере модифицируются или отделяются от неявно сохранённой копии, для того чтобы вызывать неконстантные функции члены.*

Итераторы java стиля более удобные и введены в qt4. Обычно следует использовать именно их. В каждом классе данных итераторов существует два типа:

* которые обеспечивают только чтение или
* чтение/запись.

*В данной части есть таблица соответствия между итераторами и контейнерными классами.*

Итераторы в стиле java, в отличие от итераторов стл, указывают между элементами, а не на сами элементы. Поэтому они могут ссылаться либо на самое начало контейнера, либо на самый его конец, либо посредине между элементами. *Есть пример использования данного итератора. Это очень удобная вещь. Далее есть таблица для обобщения программного интерфейса итератора списка в стиле java.*

Но данный итератор не обеспечивает функций для удаления или вставки. Для выполнения данных операций нужно вместо QListIterator использовать класс QMutableListIterator. Причём модифицируется или удаляется то значение, которое было перепрыгнуто последний раз. Примерно такие же возможности есть у итераторов для ассоциативных контейнеров, но в данном случае также обеспечиваются функции для извлечения ключа и значения, а также итерации между значениями с одинаковым ключом. Что касается стл итераторов, то их также есть два типа, а также для них *приводится таблица соответствия с контейнерными классами.* Данные итераторы могут использоваться с алгоритмами qt и стл. Для данных итераторов программный интерфейс совпадает с интерфейсом для указателей массивов (++, --). Они указывают на конкретный элемент. Функция end() указывает на недействительную позицию, а функция begin() тоже. В результате для пустого контейнера мы никогда не выйдем из цикла. При декрементировании итератора это следует сделать прежде доступа к элементу контейнера. *Также есть таблица операторов над стл итераторами.*

*Из-за неявного разделения вовсе не затратно возвращать контейнеры по значению. В qt многие функции делают это.*

На стл итераторы из-за неявного разделения накладываются некоторые ограничения:

* нельзя использовать итератор сразу же при возврате контейнера по значению (нужно использовать его копию – *есть пример кода*);
* нельзя взять копию контейнера, пока непостоянные итераторы активны в нём.

Итераторы в стиле java лишены данных недостатков. Следует лучше понять этот аспект.

Для итерации через все элементы контейнера можно использовать ключевое слово foreach. Оно обрабатывается препроцессором. Код с данным словом значительно меньше, чем код с итераторами.

Если тип данных не содержит запятой, то его можно определить внутри данного ключевого слова (foreach). Также можно использовать фигурные скобки для блока в цикле с данным ключевым словом. При использовании данного ключевого слова (foreach) создаётся копия контейнера. Поэтому вы модифицируете не сам контейнер, а его копию.

Также qt обеспечивает псевдо ключевое слово forever для бесконечных циклов.

Также данные макросы можно убрать при помощи такой строки в файле проекта: CONFIG += no\_keywords. Qt содержит также несколько классов, которые в некотором отношении походят на контейнеры. Но эти классы не обеспечивают итераторы и не могут использоваться с ключевым словом foreach: [QVarLengthArray](http://qt-project.org/doc/qt-5.1/qtcore/qvarlengtharray.html)<T, Prealloc>, [QCache](http://qt-project.org/doc/qt-5.1/qtcore/qcache.html)<Key, T>, [QContiguousCache](http://qt-project.org/doc/qt-5.1/qtcore/qcontiguouscache.html)<T>, [QPair](http://qt-project.org/doc/qt-5.1/qtcore/qpair.html)<T1, T2> (*о них подробнее будет вестись речь ниже*).

Также есть некоторые нешаблонные типы, которые соревнуются с контейнерами: [QBitArray](http://qt-project.org/doc/qt-5.1/qtcore/qbitarray.html), [QByteArray](http://qt-project.org/doc/qt-5.1/qtcore/qbytearray.html), [QString](http://qt-project.org/doc/qt-5.1/qtcore/qstring.html), and [QStringList](http://qt-project.org/doc/qt-5.1/qtcore/qstringlist.html).

Сложность алгоритма – это то, насколько быстро (или медленно) работает функция, если размер контейнера возрастает. Например, вставка элемента в середину списка является очень быстрой, тогда как для вектора это очень затратная операция, которая требует перемещения в памяти половины элементов вектора. Для описания алгоритма используется следующая терминология:

* Постоянное время: время выполнения функции не зависит от числа элементов в контейнере.
* Логарифмическое время: время выполнения пропорционально логарифму количества элементов контейнера.
* Линейное время: время выполнения пропорционално количеству элементов контейнера.
* Линейно-логарифмическое время: функция асимптотически медленнее, чем линейная, но бытсрее чем логарифмическая.
* Квадратичное время: время выполнения пропорционално квадрату количества элементов контейнера.

*В данной части есть таблица, которая показывает скорость некоторых операций по отношению к некоторым контейнерам.*

[QVector](http://qt-project.org/doc/qt-5.1/qtcore/qvector.html)<T>, [QString](http://qt-project.org/doc/qt-5.1/qtcore/qstring.html), and [QByteArray](http://qt-project.org/doc/qt-5.1/qtcore/qbytearray.html) сохраняют свои значения рядом в памяти, [QList](http://qt-project.org/doc/qt-5.1/qtcore/qlist.html)<T> удерживает масив указателей на данные, что обеспечивает основанный на индексах доступ. [QHash](http://qt-project.org/doc/qt-5.1/qtcore/qhash.html)<Key, T> содержит хэш-таблицу, чей размер пропорционален числу элементов в хэше.

*Также там описана стратегия перевыделения памяти по мере увеличения размера строки.*

[QVector](http://qt-project.org/doc/qt-5.1/qtcore/qvector.html)<T>, [QHash](http://qt-project.org/doc/qt-5.1/qtcore/qhash.html)<Key, T>, [QSet](http://qt-project.org/doc/qt-5.1/qtcore/qset.html)<T>, [QString](http://qt-project.org/doc/qt-5.1/qtcore/qstring.html), and [QByteArray](http://qt-project.org/doc/qt-5.1/qtcore/qbytearray.html) обеспечивают три функции для контроля алгоритма перераспределения памяти: capacity(), reserve(), squeeze(). Если вы знаете, сколько точно элементов в вашем контейнере, то вы можете вначале вызвать функцию reserve(), а затем заполнить ваш контейнер.

#### QVector

<http://qt-project.org/doc/qt-5.1/qtcore/qvector.html#details>

QVector класс – это шаблон, который обеспечивает динамический массив. Это один из общих контейнерных классов Qt. Он сохраняет элементы в соседних позициях в памяти и обеспечивает к ним быстрый доступ. В данном классе оператор at() работает быстрее, чем [], так как он не выполняет никогда копирования. Заметьте, что если вы используете функцию resize(), то так как вектор использует соседние ячейки памяти, то ему может понадобиться перекопировать себя в новое место целиком. Если вы знаете, сколько вам понадобится элементов, вы можете вызвать функцию reserve(), которая зарезервирует нужное количество элементов для вектора. Данный класс также обеспечивает Java-итераторы и STL-итераторы. Они используются редко. Он не поддерживает вставку, представку, добавление сзади или замещение по отношению к его собственным значениям. Это вызовет прекращение работы приложения. Для большинства целей QList является более правильным классом для использования. Операции вставки и добавления с ним работают быстрее, чем с вектором, из-за способа хранения элементов, а его основанный на индексах интерфейс более удобен, чем QLinkedList. Он также образует меньше кода в исполняемом файле. Если вы желаете иметь настоящий связанный список с гарантией вставок постоянного времени в середину списка, а также итераторы для списка более предпочтительны, чем индексы, используйте QLinkedList. Если вы желаете, чтобы элементы занимали ближайшие позиции в памяти, или если ваши элементы больше, чем указатель и вы желаете избежать переполнения при выделении для них памяти в яме во время вставки, используйте вектор. Если вы желаете низкоуровневый массив с переменным размером, то используйте QVarLengthArray.

#### QList

<http://qt-project.org/doc/qt-5.1/qtcore/qlist.html#details>

QList – это шаблонный класс для списков. Он обеспечивает быстрый доступ к элементам, также как и быстрое их добавление или вставку. Внутренне данный класс представляется как массив указателей на некоторый данный тип T. Если это сам по себе указатель или один из разделяемых классов Qt, то они сохраняются в список напрямую. Для списков размером тысячи элементов этот массив позволяет очень быструю вставку в середину, а также обеспечивает основанный на индексах доступ. Если есть необходимость выделять память в яме для отдельных элементов, то лучше использовать QVector.

Класс QStringList обеспечивает хранение строк. Это просто QList<QString>, который обеспечивает дополнительные удобные функции для более удобной работы с данным классом.

В данном классе операция [] является очень быстрой, так как данный класс организован как массив указателей. Но операция at() всё равно работает быстрее. Также здесь есть различные типы итераторов. Также следует иметь в виду, что функции списка всегда полагают, что для них указан правильный индекс. Поэтому они могут приводить к ошибке. Если к тому же вы определите во время компиляции QT\_NO\_DEBUG, то эти ошибки не будут зафиксированы.

#### QLinkedList

<http://qt-project.org/doc/qt-5.1/qtcore/qlinkedlist.html>

Класс QLinkedList – это шаблонный класс, который обеспечивает связанные списки. Он сохраняет список значений и обеспечивает основанный на итераторах доступ, также как и вставки за постоянное время, также как и удаления за постоянное время.

Из википедии: связный список — базовая динамическая [структура данных](http://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), состоящая из [узлов](http://ru.wikipedia.org/wiki/%D0%A3%D0%B7%D0%B5%D0%BB_%28%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29), каждый из которых содержит как собственно [данные](http://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85_%28%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29), так и одну или две [ссылки](http://ru.wikipedia.org/wiki/%D0%A1%D1%81%D1%8B%D0%BB%D0%BA%D0%B0_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) («связки») на следующий и/или предыдущий узел списка.

#### QVarLengthArray

<http://qt-project.org/doc/qt-5.1/qtcore/qvarlengtharray.html#details>

QVarLengthArray – это класс, который обеспечивает низкоуровневый массив переменного размера. Язык С++ не поддерживает массивы с изменяемым размером в стеке.

*Там приведён классный код демонстрации данной ошибки, который раскрыл мне Глаза на мои предыдущие проблемы.* При выделении памяти в яме данное явление вполне возможно.

Данный класс обеспечивает следующую возможность: он выделяет массив в стеке фиксированной величины. Если вы увеличиваете размер массива, то для новых элементов автоматически выделяется память в яме. Выделение памяти в стеке является намного более быстрым, чем выделение памяти в яме. Во всех контейнерных классах могут помещаться только присваиваемые типы данных.

В отличие от QVector данный класс является намного более низкоуровневым и не обеспечивает всей той функциональности, которую обеспечивает вектор. Данный класс не поддерживает неявное разделение, но при этом он гораздо более быстрый, чем вектор, так как не требует лишних накладных расходов. Данный класс следует использовать лишь в очень специализированных случаях с целью некоторой оптимизации.

#### QStack

<http://qt-project.org/doc/qt-5.1/qtcore/qstack.html#details>

класс QStack является шаблонным классом, который обеспечивает стек. Это один из основных контейнерных классов qt. Он реализует стек для данных одинакового типа. Стек – это структура типа последний пришёл, первый ушёл (last in, first out (LIFO)). Другими словами, элементы передаются наверх и забираются также сверху. Вся функциональность QVector также применима к данному классу. Также типом данных данного класса должен быть присваевамый тип данных.

#### QQueue

<http://qt-project.org/doc/qt-5.1/qtcore/qqueue.html#details>

QQueue – это шаблонный класс, который обеспечивает очередь. Это один из базовых контейнерных классов. Очередь – это контейнер типа «первый пришёл, последний ушёл» (FIFO – first in, first out). Элементы добавляются к хвосту очереди, а извлекаются из её начала. Данный класс наследует QList. Вся функциональность данного класса применима и к очереди. Также очередь может содержать только присваиваемые типы данных.

#### QStringList

<http://qt-project.org/doc/qt-5.1/qtcore/qstringlist.html#details>

описаны основные особенности класса QStringList

#### QMap

<http://qt-project.org/doc/qt-5.1/qtcore/qmap.html>

класс QMap – это шаблон, который обеспечивает красно-чёрно-древовидный словарь. Данный класс обеспечивает почти такую же функциональность, что и класс QHash, но есть и отличия:

* Хэш обеспечивает более быстрый поиск
* В хэше при использовании итератора элементы имеют произвольный порядок, а в мэпе они упорядочены по ключу
* Тип ключа хэша должен обеспечивать оператор ==() и Глобальную функцию qHash(), а в мапе должен обеспечивать оператор <().

Для вставки в мап можно использовать оператор [] (*смотри код*). Но авторы рекомендуют использовать функции contains() и value(). Так как этот оператор молча вставляет в мап значение для данного ключа, если он не был найден. Также в данном контейнере есть различные типы итераторов. Одному ключу можно присваивать много различных значений, которые затем можно получить. Для извлечения значений, но не ключей можно использовать foreach.

##### QHash

<http://qt-project.org/doc/qt-5.1/qtcore/qhash.html>

класс QHash является шаблоном для обеспечения словаря, основанного на хэш-таблице. Он обеспечивает очень быстрый поиск значения. Всё остальное справедливо как для класса QMap. При использовании итератора порядок элементов непредсказуем. Также можно сохранять для одного значения ключа несколько значений. Для извлечения значений, но не ключей можно использовать foreach.

Тип данных элемента (или ключа???) должен обеспечивать оператор ==(), а также глобальную функцию qHash(), *возможные сигнатуры которой показаны в данной части*. Данная функция рассчитывает численное значение, основанное на ключе. Есть некоторые типы ключей, *которые перечислены в данной части,* для которых в заголовочном файле <QHash> есть реализации данной функции. Для многих классов в их заголовочных файлах также есть данная функция. Для этого обращайтесь к заголовочным файлам данных классов. *В данной части показан пример кода вашего собственного класса, который вы желаете использовать в качестве ключа.*

Все хэш-таблицы уязвимы для определенного класса атак отказа в обслуживании, в котором злоумышленник тщательно предварительно вычисляет набор различных ключей, которые будут хэш-функцией в том же ведре хэш-таблицы. Атака направлена на получение наихудшего поведения алгоритма, когда данные подаются в таблицу. С целью избежать данного плохого поведения расчёт хэш значения, который производит функция qHash(), может быть улучшено произвольным сидом, который обнуляет степень атаки. Этот сид автоматически генерируется классом QHash единожды за процесс, а затем передаётся данным классом в качестве второго аргумента функции qHash(). Такая произвольность данного класса гарантируется по умолчанию. Для того чтобы отменить произвольность сида с целью отладки можно использовать переменную среды QT\_HASH\_SEED, которая будет задавать значение сида.

#### QSet

<http://qt-project.org/doc/qt-5.1/qtcore/qset.html>

класс QSet является шаблоном для обеспечения набора, основанного на хэш-таблице. Он сохраняет элементы в неопределённом порядке, но обеспечивает очень быстрый их поиск. Требования к типу данных для элементов данного контейнера такие же, как и для контейнера QHash. Внутренне данный класс использует хэш-таблицу для осуществления поиска.

#### QMultiMap

<http://qt-project.org/doc/qt-5.1/qtcore/qmultimap.html#details>

класс QMultiMap является удобным подклассом QMap для обеспечения многозначных мапов. В данном классе нет оператора [](). При требовании значения возвращается наиболее позднее. Но можно и получить все значения для данного ключа. *Как это делается, показано в данной части.*

#### QMultiHash

<http://qt-project.org/doc/qt-5.1/qtcore/qmultihash.html>

тоже самое, что и QMultiMap, но для класса QHash.

#### QCache

<http://qt-project.org/doc/qt-5.1/qtcore/qcache.html#details>

класс QCache является шаблонным классом, который обеспечивает кэш. Преимуществом данного класса перед контейнерными классами является то обстоятельство, что данный класс осуществляет собственность над объектами и удаляет их для освобождения места для других объектов, если необходимо. Для каждого объекта определяется приблизительный его объём памяти. Также определяется максимальный объём КЭШа. Если память объектов превышает объём КЭШа, то они начинают удаляться, начиная с тех, доступ к которым был осуществлён наиболее давно. После получения доступа к любому объекту, им начинает владеть кэш и он решает, когда его удалять. Также данный класс обеспечивает некоторые функции для получения объектов из КЭШа.

#### QContiguousCache

<http://qt-project.org/doc/qt-5.1/qtcore/qcontiguouscache.html#details>

класс QContiguousCache является шаблоном, который обеспечивает смежный кэш. Этот класс удобен для кэширования элементов, которые отображаются в интерфейсе пользователя. В отличие от обычного КЭШа он накладывает ограничение, что элементы в КЭШе должны быть смежными. Это позволяет данному классу расходовать меньше памяти и циклов процессора. Перед использованием КЭШа следует установить его ёмкость, после чего следует выполнять добавление в него элементов. Кэш при переполнении удаляет элементы с обратного конца по отношению к тем, которые были недавно добавлены (спереди или сзади).

#### QPair

[*http://qt-project.org/doc/qt-5.1/qtcore/qpair.html#details*](http://qt-project.org/doc/qt-5.1/qtcore/qpair.html#details)

класс QPair является шаблонным классом для сохранения пары элементов. Данный класс может быть использован для возвращаемого значения некоторой функции, если она должна вернуть два значения. Типы данных в данном шаблоне должны быть присваиваемыми.

### ПОДДЕРЖКА ПОТОКОВ В КТ

<http://qt-project.org/doc/qt-5.1/qtcore/threads.html>

qt обеспечивает поддержку потоков в форме

* платформенно-независимых поточных классов,
* поточно-безопасного способа проводки событий и
* соединения сигналов и слотов через потоки.

Это делает лёгким разработку портативных многопоточных приложений qt, которые имеют преимущество на многопроцессорных машинах. Многопроцессорное приложение также является полезной парадигмой для выполнения время затратных операций без замораживания интерфейса пользователя приложения. *на данной странице приводятся ссылки на полезные книги для введение в многопоточное программирование приложения. Мне затем их следует прочитать.*

#### ИНИЦИИРОВНАИЕ ПОТОКОВ ПРИ ПОМОЩИ QTHREAD

<http://qt-project.org/doc/qt-5.1/qtcore/threads-starting.html>

экземпляр QThread представляет поток и обеспечивает средства для инициирования потока, который затем выполнит функцию QThread::run(). Данная функция для потока имеет то же самое значение, что и функция main() для приложения. Весь код, который выполняется в стеке вызовов, который начинается в функции run(), будет выполнен в новом потоке, и поток закончится, когда функция вернёт некоторое значение. Данный класс испускает сигналы, чтобы показать, что он начал или закончил выполнение. Для создания потока следует создать подкласс класса QThread и переопределить функцию run(). Затем следует создать экземпляр данного объекта и вызвать функцию QThread::start(). *Также следует иметь в виду, что перед созданием любого потока следует создать объект приложения.* Затем запустится новый поток, а главный поток продолжит выполнение. Затем автоматически будет выполнен код, который написан в функции run().

Также следует отметить, что функция [QCoreApplication::exec](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html#exec)() должна всегда вызываться из главного потока приложения, но не из любого другого потока приложения. В графических приложениях главный поток также называется GUI потоком, потому что это единственный поток, которому позволено выполнять связанные с графическим интерфейсом пользователя операции.

#### СИНХРОНИЗАЦИЯ ПОТОКОВ

<http://qt-project.org/doc/qt-5.1/qtcore/threads-synchronizing.html>

классы QMutex, QReadWriteLock, QSemaphore и QWaitCondition обеспечивают синхронизацию потоков. Пока главная идея потоков – это то, что им следует быть одновременными настолько, насколько это возможно. Есть точки, в которых потоки должны останавливаться и ждать другие потоки. Например, если два потока пытаются получить доступ к одинаковой глобальной переменной одновременно, то результаты обычно не определены. Мьютекс обеспечивает взаимно исключающую блокировку или мьютекс. Максимум один поток может удерживать в единицу времени мьютекс. Если поток попытается получить мьютекс, пока тот занят, то он переведётся в режим сна, до тех пор пока мьютекс не освободится. Они часто используются, чтобы защитить доступ к разделяемым данным (к которым могут обратиться много потоков одновременно). Его используют для организации поточно безопасных классов. QReadWriteLock похож на QMutex за тем исключением, что он отличает доступ для чтения и доступ для записи к разделяемым данным. Его также используют для написания поточно безопасных классов. QSemaphore – это обобщение QMutex, которое защищает определённое число одинаковых ресурсов. В отличие от мьютекса, который защищает только один ресурс. Типичное применение данного класса – это синхронизованный доступ к циклическому буферу между производителем и покупателем. QWaitCondition позволяет потоку пробуждать другие потоки, когда встретилось некоторое условие. Один или много потоков могут блокировать ожидание для данного класса, чтобы установить условие. Также следует заметить, что данные классы синхронизации полагаются на использование качественно организованных указателей. Например, вы не можете использовать упакованные классы с вижуал студио (тоже очень плохо понял данную тему).

#### РЕЕНТЕРАБЕЛЬНОСТЬ И ПОТОКОБЕЗОПАСНОСТЬ

<http://qt-project.org/doc/qt-5.1/qtcore/threads-reentrancy.html>

сквозь документацию постоянно используются два термина:

* reentrant (реентерабельный) и
* thred-safe (потокобезопасный).

Они используются для классов, чтобы показать, как они могут использоваться в многопоточных приложениях. Потокобезопасная функция может быть вызвана одновременно из множества потоков, даже когда вызовы используют разделяемые данные, так как все ссылки на разделяемые данные упорядочены. Реентерабельная функция обозначает, что она также может быть вызвана одновременно из множества потоков, но только если каждый вызов использует свои собственные данные. Следовательно потокобезопасная функция является всегда реентерабельной, но реентерабельная функция не всегда потокобезопасная. Реентерабельный класс обозначает, что его функции могут быть вызваны из различных потоков безопасно, пока каждый поток использует различные экземпляры данного класса. Потокобезопасный класс обозначает, что его функции могут быть вызваны из различных потоков, даже если все они используют один и тот же экземпляр данного класса. Также следует заметить, что если функция не отмечена как потокобезопасная или реентерабельная, то её не следует использовать в разных потоках. Что касается класса, то для него данное обстоятельство обозначает, что нельзя запрашивать доступ к членам данного класса из разных потоков.

**Реентерабельность**

Классы С++ часто являются реентерабельными, просто потому, что они используют их собственные члены данных. Любой поток может вызвать функцию-член экземпляра многопоточного класса, также как и ни один другой поток не может вызвать функцию член того же экземпляра в одинаковое время. Например, некоторый класс является реентерабельным. В нём определены операции инкремента и декремента. Но данный класс не является потокобезопасным, так как если многие потоки попытаются модифицировать данные, результат будет не определён. Это из-за того, что операторы ++ и -- не являются атомарными. Действительно, данные операции обычно вызывают три машинные операции: загрузку значения переменной в регистр; инкрементирование или декрементирование данного значения; сохранение значения из регистра в главную память. Если два потока загрузят старое значение переменной одновременно, инкрементируя его в регистре, и сохранят его назад, то они совершат переписывание друг друга, в результате переменная инкрементируется только один раз. Очевидно, что доступ может быть упорядоченным: один поток может гарантированно выполнить данные три шага перед тем, как второй поток выполнить такие же самые шаги. Наиболее простой способ сделать это – это использовать мьютекс.

Многие классы qt не являются потокобезопасными, так как это повлекло бы переполнение из-за повторяющегося закрытия и открытия мьютексов. Некоторые классы qt всё-таки являются потокобезопасными. Это касается самих потоковых классов, а также некоторых фундаментальных функций, например, некоторых функций QCoreApplication.

*Также в данной части отмечается, что в многопоточной литературе нет устоявшейся терминологии, из-за чего вам всегда необходимо удостовериться, что вы правильно понимаете*

*терминологию данной книги.*

#### ПОТОКИ И QOBJECTS

<http://qt-project.org/doc/qt-5.1/qtcore/threads-qobject.html>

QThread наследует QObject. Он высылает сигналы, чтобы продемонстрировать, что поток начался или окончился. Более интересным свойством QObject является тот факт, что данные классы могут быть использованы во мгогих потоках, испускать сигналы, которые запрашивают слоты из других потоков. Это возможно, так как каждому потоку позволено иметь свой собственный цикл обработки событий. QObject является реентерабельным. Многие его подклассы, которые не связаны с графическим интерфейсом пользователя, также являются реентерабельными. Но необходимо отметить, что все данные классы проектировались с той целью, чтобы они обрабатывались только в одном потоке. Создание одного объекта в одном потоке и запрашивание его функций-членов из других потоков не гарантирует правильной работы данного объекта.

Есть некоторые ограничения, которые необходимо держать в уме:

* дочерний объект QObject должен всегда создаваться в том же потоке, где находится и родитель. Поэтому нельзя класс QThread делать родительским классом объекта, который будет создан в другом потоке.
* Объекты, управляемые событиями, должны быть использованы в одном и том же потоке.
* Также вы должны гарантировать, что все объекты, которые созданы в потоке, будут уничтожены прежде, чем будет уничтожен сам поток. Это просто обеспечить, создавая объекты в стеке в определении функции run().
* Все графические классы qt не являются реентерабельными и должны быть вызваны из Главного потока приложения, откуда также обязательно должна вызываться функция QCoreApplication::exec(). Обычно это позволяет время затратные операции производить в отдельном потоке, а результат отображать графически лишь в главном потоке.

Каждый поток может иметь свой собственный цикл обработки событий. Для начала собственного цикла обработки событий поток должен использовать функцию QThread::exec(). Также можно и заканчивать цикл обработки событий. Данная возможность позволяет потоку использовать некоторые неграфические классы qt, которые требуют наличия цикла обработки событий: QTimer, QTcpSocket, QProcess. Данная возможность также позволяет соединять сигналы из любого потока к слотам некоторого определённого потока.

Обычно объекты QObject живут в том же потоке, в котором они были созданы. События к этому объекту отправляются циклом обработки событий данного потока. Получить данный поток можно при помощи функции: QObject::thread(). Следует заметить, что данная функция возвращает нуль для тех объектов, которые были созданы перед созданием QApplication. Следует отметить, что обрабатывать события таких объектов может только Главный поток приложения. Но в классе QObject есть функция которая позволяет изменять поток для данного объекта и его наследников. Удаление объекта из потока, к которому он не принадлежит, является небезопасным, так как данный объект может обрабатывать некоторые события. Но вызвав функцию QObject::deleteLater() будет в конечном счёте захвачен циклом обработки событий того потока, в котором находится данный объект.

Если цикл обработки событий в потоке не запущен, то события не могут быть доставлены к объекту. Это ограничение также относится и к главному потоку приложения.

Вы можете вручную создать событие к любому объекту в любом потоке в любое время с использованием поточно безопасной функции QCoreApplication::postEvent(). События автоматически будут отправлены циклом обработки событий того потока, где объект был создан. Фильтры событий поддерживаются во всех потоках с тем лишь условием, что следящий объект должен жить в том же потоке, что и поток, за которым он следит. Это же касается функции QCoreApplication::sendEvent().

Qobject и все его подклассы не являются потокобезопасными. Это включает полностью систему доставки событий. Нужно помнить, что цикл обработки событий может доставлять события вашему объекту, пока вы получаете доступ к нему из других потоков. Если вы планируете высылать вашему объекту некоторые события из других потоков, то вам необходимо гарантировать единственность доступа к нему при помощи мьютекса. С другой стороны вы можете высылать сигналы объекту из функции run(), так как высылание сигналов является потокобезопасным.

Qt обеспечивает следующие типы соединений между сигналами и слотами:

* Авто-соединение (по умолчанию). Если сигнал испускается в потоке, где живёт объект-получатель, то поведение является таким же как при прямом соединении. Иначе поведение будет таким же, как и у соединения по очереди.
* Прямое соединение. Слот запрашивается немедленно, когда сигнал высылается. Слот выполняется в потоке испускателя сигнала, но это не обязательно поток получателя сигнала.
* Соединение по очереди. Слот запрашивается, когда управление возвращается к циклу обработки событий получателя. Слот выполняется в потоке получателя.
* Блокирующее соединение по очереди. Слот запрашивается как в соединении по очереди, за исключением того, что текущий поток блокируется, пока слот не возвратит значение. Использование данного типа соединения в таком же потоке вызовет тупик.
* Уникальное соединение. Поведение такое же, как и при автоматическом соединении, но соединение происходит, только если нет его дубликата.

Тип соединения определяется при передаче дополнительного аргумента функции connect(). Имейте в виду, что соединение объектов, которые живут в различных потоках, является небезопасным, если в потоке получателя работает цикл обработки событий. По этой же причине вызов любой функции объекта из другого потока также является небезопасным. Сама по себе функция QObject::connect() является потокобезопасной.

#### ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ

<http://qt-project.org/doc/qt-5.1/qtcore/threads-qtconcurrent.html>

пространство имён QtConcurrent обеспечивает высокоуровневый интерфейс программирования приложений, который делает возможным написание многопоточных приложений без использования низкоуровневых потоковых примитивов, таких как мьютексы, блокировки чтения-записи, соединения ожидания или семафоры. Программы, написанные при помощи данного пространства имён автоматически приводят в порядок количество используемых потоков в соответствии с числом ядер процессора. Это значит, что приложения, написанные сегодня, будут продолжать масштабирование, когда будут развёрнуты на многопроцессорных системах в будущем.

QtConcurrent включает интерфейс программирования приложений для параллельной обработки, включая MapReduce и FilterReduce для систем с разделяемой памятью, а также классов для проведения асинхронных вычислений в приложениях с графическим интерфейсом пользователя.

QtConcurrent поддерживает несколько контейнеров STL, а также типов итераторов. Но данный модуль лучше работает с контейнерами qt, которые имеют итераторы произвольного доступа.

#### ПОДДЕРЖКА ПОТОКОВ В МОДУЛЯХ QT

<http://qt-project.org/doc/qt-5.1/qtcore/threads-modules.html>

SQL модуль. Соединение с базой данных может быть сделано только в том же потоке, в котором оно было создано. Перебрасывание соединений из потока в поток, а также создание запросов из других потоков не поддерживается. Также сторонние библиотеки, которые используются при написании кода для QSqlDriver, могут накладывать определённые ограничения на использование данного модуля в многопоточных приложениях.

QPainter может быть использован в потоке для рисования QImage, QPrinter и QPicture. Рисование на QPixmaps и QWidget не поддерживается. Много потоков могут одновременно рисовать, но рисовать на данном устройстве может только один единственный поток.

Классы [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html), [QTextCursor](http://qt-project.org/doc/qt-5.1/qtgui/qtextcursor.html) и все связанные классы являются реентерабельными.

Qt использует оптимизацию, называемую неявное разделение для многих своих классов, таких как [QImage](http://qt-project.org/doc/qt-5.1/qtgui/qimage.html) and [QString](http://qt-project.org/doc/qt-5.1/qtcore/qstring.html). Неявно разделяемые классы, начиная с qt4, могут безопасно копироваться между потоками. Они полностью реентерабеольные, а неявное разделение классов является реально неявным. В qt интегрирование многопоточности и неявного разделения происходит за счёт того, что счётчик ссылок является атомарным. При этом это не гарантирует потокобезопасности. Используйте поточные примитивы для использования одного неявно разделяемого класса во многих потоках. Вообще авторы документации рекомендуют использовать для передачи данных между потоками механизм сигналов и слотов, так как они не нуждаются в явном блокировании. Таким образом, можно сделать вывод, что неявно разделяемые классы можно использовать точно таким же образом, как если бы они были обычными классами со значениями, в многопоточных приложениях.

#### QAtomicInt

<http://qt-project.org/doc/qt-5.1/qtcore/qatomicint.html>

класс QAtomicInt обеспечивает платформонезависимые атомарные операции над целыми числами. Атомарная операция – это комплексная операция, которая выполняется без прерываний. В данном классе есть функции для инкрементирования и декрементирования целого числа, которое может быть использована для подсчёта ссылок и для создания ваших собственных неявно разделяемых классов. Класс предоставляет несколько определений функций для проверки-установки значений, выбора-сохранения значений, выбора-добавления значений. Каждое определение определяет семантику упорядочивания памяти, которая описывает, как память (*там дальше идёт перевод, но я не понимаю, что он конкретно обозначает*). Так как современные процессоры позволяют выполнение вне очереди и упорядочивание памяти, то использование правильной семантики необходимо для гарантии, что ваши приложения будут работать на всех процессорах. *Далее приводятся некоторые настройки, которых я до конца не понимаю. А также последнее высказывание я также осознал довольно смутно.*

Функции проверки работают так: если текущее значение совпадает с некоторым, то ему присваивается некоторое другое значение, а функция возвращает правду. Иначе она не делает ничего, а возвращает ложь. Функции выборки и сохранения возвращают текущее значение и присваивают данному числу новое значение. Функции выборки и добавления возвращают текущее значение и добавляют к нему некоторое другое значение.

Данный класс будет выполнять атомарные операции на всех процессорах, но так как не все процессоры обладают возможностями выполнения всех операций данного класса, то необходимо изучать процессоры, прежде чем использовать данный класс. В qt есть различные макросы, при помощи которых можно проверить во время компиляции, обладает ли ваше железо необходимыми свойствами.

Свободные от ожидания операции (wait-free) – это операции, которые всегда выполняются за одно и то же время. Они обычно не выполняются с использованием любых блокировок и циклов. Для данных операций также существуют некоторые макросы.

#### QAtomicPointer

<http://qt-project.org/doc/qt-5.1/qtcore/qatomicpointer.html#details>

класс QAtomicPointer является шаблоном для обеспечения платформонезависимых атомарных операций на указателях. Далее весь текст полностью совпадает с таковым для класса QAtomicInt, но макросы здесь, конечно же другие. *К изучению данных классов мне следует вернуться после освоения некоторой информации о многопоточных приложениях.*

#### QMutex

<http://qt-project.org/doc/qt-5.1/qtcore/qmutex.html#details>

класс QMutex обеспечивает упорядочивание доступа между потоками. Цель данного класса – защитить объект, структуру данных и секцию кода таким образом, чтобы только один поток в единицу времени мог обращаться к ним. Обычно лучше использовать мьютекс с классом QMutexLocker, так как это делает лёгкой гарантию того, что блокировка и разблокировка выполнятся соответственно. *Далее есть одна очень прекрасная демонстрация использования данного класса, а также того, когда синхронизации не обеспечивается.*

#### QMutexLocker

<http://qt-project.org/doc/qt-5.1/qtcore/qmutexlocker.html#details>

класс QMutexLocker является удобным классом, который упрощает блокирование и разблокированіе мьютекса. В сложных функциях использование мьютекса является сложной задачей. Поэтому иногда тяжело понять, в каком состоянии находится мьютекс. Данный класс позволяет это однозначно определить. Объект данного класса следует создать внутри той функции, которую должен заблокировать мьютекс. Когда данный объект создаётся, то мьютекс автоматически блокируется, а когда разрушается, то мьютекс автоматически разблокируется. *В данной части демонстрируется пример, из которого сразу же становится очевидной выгода использования данного класса.*

#### QReadWriteLock

<http://qt-project.org/doc/qt-5.1/qtcore/qreadwritelock.html#details>

класс QReadWriteLock обеспечивает блокирование чтения-записи. Это защитный инструмент, который предотвращает тот факт, когда к ресурсу одновременно обращаются за чтением и записью. Данный класс применим в том случае, когда вы хотите, чтобы много потоков могли обращаться для чтения к некоторому ресурсу, но если кто-то захочет записать в него что-то, то все остальные должны быть заблокированы. Данный класс является конкурентом мьютекса и полезен, когда существует много одновременных считываний и записей, которые случаются непериодически. Также блокировка для чтения не случится, если в настоящее время ожидает своей очереди некоторая запись. *Также там далее идёт речь о некоторой рекурсии, но этого пока я не понял.*

#### QReadLocker

<http://qt-project.org/doc/qt-5.1/qtcore/qreadlocker.html#details>

класс QReadLocker является удобным классом для упрощения блокировки и разблокировки при доступе для чтения. Действует также, как и QMutexLocker.

#### QWriteLocker

<http://qt-project.org/doc/qt-5.1/qtcore/qwritelocker.html#details>

класс QWriteLocker является тем же самым, что и QReadLocker, но только для записи.

#### QRunnable

<http://qt-project.org/doc/qt-5.1/qtcore/qrunnable.html>

класс QRunnable является базовым классом для всех работоспособных объектов. Это интерфейс, который представляет задачу или кусок кода, который следует выполнить, который находится в функции run(). Вы можете использовать класс QThreadPool для выполнения данного кода в отдельном потоке. Последний класс удаляет QRannable автоматичекски, если autoDelete() возвращает правду (при завершении последнего потока данный объект будет удалён). Также последний поддерживает выполнение QRannable несколько раз. *Как это сделать, показано в данной части.* Вызов несколько раз QThreadPool::start() с одними и тем же QRannable не рекомендуется, так как это может вызвать условие гонки. *Что такое условие гонки?*

#### QThreadPool

<http://qt-project.org/doc/qt-5.1/qtcore/qthreadpool.html>

класс QThreadPool управляет набором QThreads. Он позволяет уменьшить затраты на создание потоков в программах, которые используют потоки. Каждое приложение qt имеет один глобальный объект данного класса, который может быть вызван при помощи вызова globalInstance(). Для использования одного из потоков пула нужно создать подкласс QRunnable и переопределить виртуальную функцию run(). Затем создать объект этого класса и передать его в пул, после чего вызвать функцию начала из пула. Данный класс удаляет по умолчанию QRunnable. А ранее указанный флаг позволяет это предотвратить. Также можно выполнять один и тот же QRunnable много раз (*показано, как*). Также если поток не используется некоторое определённое количество времени, то он удаляется. Можно устанавливать данное количество времени. Также можно получать информацию о количестве потоков, а также выделять потоки для некоторых внешних операций.

Это низкоуровневый класс управления потоками. Высокоуровневым классом является Qt Concurrent.

#### QSemaphore

<http://qt-project.org/doc/qt-5.1/qtcore/qsemaphore.html>

класс QSemaphore обеспечивает общий обсчитывающий семафор. Семафор является обобщением мьютекса. Если мьютекс может быть заблокирован лишь один раз, то овладевать семафором можно много раз. Семафоры обычно используются для защиты определённого количества идентичных ресурсов. Они поддерживают две фундаментальные операции:

* освобождение и
* получение.

Получение пытается получить несколько ресурсов. Пока нужное количество ресурсов не будет свободно, то вызов заблокируется, пока ресурсы не освободятся. Функция освобождения освобождает некоторое количество ресурсов. Есть функции, которые проверяют количество доступных ресурсов, а также функция, которая немедленно возвращает вызов, если требуемого количества ресурсов не обнаруживается. Типичное применение семафора – это управление доступом к буферу, который разделяется потоком производителя и потоком покупателя.

*Также в данной части есть прекрасный пример с рестораном, который наглядно демонстрирует применение семафора.*

#### QThreadStorage

<http://qt-project.org/doc/qt-5.1/qtcore/qthreadstorage.html#details>

класс QThreadStorage обеспечивает сохранение данных для некоторого потока. Данный класс хранит некоторые локальные для потока данные и удаляет их при выходе из потока. Если данные являются указателем, то ликвидируются данные, на которые тот ссылается.

Деструктор данного класса не уничтожает все данные, но они уничтожаются, когда оканчивается их поток. Данный класс может использоваться для хранения данных для главного потока приложения.

Непонятно назначение данного класса.

#### QWaitCondition

<http://qt-project.org/doc/qt-5.1/qtcore/qwaitcondition.html>

класс QWaitCondition обеспечивает условную переменную для синхронизирующихся потоков. Этот класс позволяет одному потоку сообщить другому потоку о том, что некоторый тип условия был встречен. *Данный класс позволяет перевести некоторый мьютекс в режим ожидания, а затем, когда будет угодно пользователю, вывести его оттуда при помощи некоторых функций. Если в режим ожидания переводятся несколько мьютексов, то порядок их освобождения не определён. Одновременно, если освобождается один поток, то предсказать, какой именно, нельзя. Поэтому, если необходимо пробудить некоторый конкретный поток, то используйте для разных потоков разные объекты условия.*

## Классы интернационализации и локализации

класс QLocale позволяет преобразовывать числа и их строковое представление в разных языках.

класс QStringRef обеспечивает тонкую обёртку вокруг подстрок QString. Он обеспечивает поднабор QString только для чтения.

класс QString обеспечивает строку из символов в кодировке Unicode. Данный класс сохраняет строку 16-битных символов, где каждый символ соответствует кодировке Unicode 4.0.

класс QChar обеспечивает 16-битные символы в кодировке юникод. Большинство компиляторов обрабатывают его как короткий целочисленный тип без знака.

класс QLatin1Char обеспечивает 8-битный символ в кодировке ASCII/Latin-1.

класс QLatin1String обеспечивает тонкую обёртку вокруг закодированного в кодировке US-ASCII/Latin1 строкового литерала.

### QLocale

<http://qt-project.org/doc/qt-5.0/qtcore/qlocale.html#details>

класс QLocale позволяет преобразовывать числа и их строковое представление в разных языках. В конструкторе данного класса задаются страна и язык, а затем данный класс предоставляет функции для преобразования чисел в строки. Данный класс поддерживает понятие локации по умолчанию, которая берётся из установок системы при запуске приложения. Локацию по умолчанию можно изменять при помощи специальной функции. *С локацией по умолчанию связано несколько моментов, которые описаны в описании данного класса*. Если при создании объекта данного класса не обнаруживается страна, то выбирается наиболее подходящая страна для данного языка; если же нет и языка, то выбирается локация по умолчанию. Для того чтобы проверить текущую локацию ввода с клавиатуры, используйте функцию QInputMethod::locale().

### QString

<http://qt-project.org/doc/qt-5.1/qtcore/qstring.html#details>

класс QString обеспечивает строку из символов в кодировке Unicode. Данный класс сохраняет строку 16-битных символов, где каждый символ соответствует кодировке Unicode 4.0. Юникод – это международный стандарт, который перекрывает большинство письменных систем. Данный класс использует неявное разделение. Вдобавок к данному классу qt обеспечивает класс QByteArray для сохранения байтов и обычных 8-битных строк с нулём на конце. Обычно используйте QString, второй класс используйте для сохранения байтных данных или если затраты памяти очень критичны. Везде, где с данным классом используется константная срока, она интерпретируется как обычная строка с нулём на конце в кодировке UTF-8.

Для доступа к символам строки лучше использовать оператор at(), который работает быстрее. Класс позволяет расширять размер строки, заполнять пустые места заданными символами, хранить символы «\0» внутри строки, различные функции для манипулирования данными строки, для обработки пробелов, для управления выделением памяти для строки, для нахождения местоположения некоторого символа или строки. Также класс предоставляет много функций для конвертации чисел в строки и наоборот. Строки можно сравнивать.

Класс содержит три функции для конвертации строк между кодировками (*одна из них соответствует локальной кодировке данной машины*). Другие кодировки поддерживаются через класс QTextCodec. Также есть некоторые макросы, которые запрещают автоматическую конвертацию между данным классом и С-строками. *В данной части показано, как их использовать.*

*Показана разница между нулевой и пустой строкой.* Рекомендуется использовать только пустые строки. *Часть More Efficient String Construction я пока не понял.*

### QStringRef

<http://qt-project.org/doc/qt-5.1/qtcore/qstringref.html#details>

класс QStringRef обеспечивает тонкую обёртку вокруг подстрок QString. Он обеспечивает поднабор QString только для чтения. Данный класс указывает на некоторую порцию строки и может вернуть её как отдельную строку. Этот класс создан для оптимизации производства подстрок из некоторой общей строки. Данный класс избегает выделения памяти и подсчёта ссылок, но только ссылается на часть изначальной строки. Данный класс является полезным для низкоуровневого кода, например, для анализатора строк. Объект данного класса ссылается на строку, которая существует, но если данной строки не существует, то данный объект ссылается на некоторое неизвестное место в памяти и будет возвращать некоторый мусор. *Мы рекомендуем использовать данный класс в устойчивом коде, где это может вызвать увеличение производительности.*

### QChar

<http://qt-project.org/doc/qt-5.1/qtcore/qchar.html#details>

класс QChar обеспечивает 16-битные символы в кодировке юникод. Большинство компиляторов обрабатывают его как короткий целочисленный тип без знака. Данный класс содержит множество функций для символов: проверка типа символа, направление написания символа, функции преобразования символов из прописных в строчные, а также функции преобразования между различными кодировками.

### QLatin1Char

<http://qt-project.org/doc/qt-5.1/qtcore/qlatin1char.html>

класс QLatin1Char обеспечивает 8-битный символ в кодировке ASCII/Latin-1.

### QLatin1String

<http://qt-project.org/doc/qt-5.1/qtcore/qlatin1string.html>

класс QLatin1String обеспечивает тонкую обёртку вокруг закодированного в кодировке US-ASCII/Latin1 строкового литерала. Многие функции класса QString перегружены с тем, чтобы обрабатывать простые С-строки. Этот класс используется для определения константных С-срток. Д*анный класс может быть использован везде, где ожидается использование QString, в отличие от простой константной строки.*

## АНИМАЦИИ

<http://qt-project.org/doc/qt-5.1/qtcore/animation-overview.html>

каркас по анимации преследует цель обеспечить лёгкий способ создания анимированного и плавного графического интерфейса пользователя. Данный каркас может быть также использован с каркасом графических представлений. Основание данного каркаса состоит из базового класса QAbstractAnimation и его двух подклассов [QVariantAnimation](http://qt-project.org/doc/qt-5.1/qtcore/qvariantanimation.html) and [QAnimationGroup](http://qt-project.org/doc/qt-5.1/qtcore/qanimationgroup.html). Первый класс представляет все базовые свойства, которые являются общими для всех типов анимации: способность начинаться, останавливаться и становиться на паузу. Класс QPropertyAnimation выполняет анимацию свойства, которое является частью мета-объектной системы. Он интерполирует изменяющееся свойство. Если вы хотите его использовать, то объявите его как свойство, а класс сделайте QObject. Сложную анимацию можно построить при помощи древовидной структуры классов QAbstractAnimation. Дерево строится при помощи QAnimationGroup, который работает как контейнер для других анимаций. Заметьте, что группы могут также содержать внутри и другие группы, так как они также являются подклассами QAbstractAnimation.

Анимация контролируется глобальным таймером, который посылает сообщения об обновлении всем анимациям, которые в настоящее время играют.

*Далее продемонстрирован код того, как можно осуществить перемещение кнопки по экрану.*

Для анимации класса QGraphicsItem, который не является подклассом QObject, следует использовать множественное наследование и уже использовать анимацию для соответствующего подкласса.

При анимации свойств вы можете задавать начальное и конечное значение свойства, продолжительность анимации, задавать некоторые ключевые точки, через которые должна проходить анимация, а также задавать функцию, которая управляется скоростью изменения анимации во времени. Также есть некоторые классы, которые выступают как контейнеры для анимации. Эти классы позволяют проигрывать анимацию параллельно или последовательно одна за одной. *В данной части есть пример проигрывания анимаций параллельно и последовательно.*

Можно строить деревья анимаций. При использовании машины состояний мы можем связать одну или более анимаций в переход между состояниями с использованием классов QSignalTransition и QEventTransition. Оба эти класса являются производными от QAbstractTransition. Также можно ассоциировать свойства с состоянием вместо задания начального и конечного значений вручную.

### QAbstractAnimation

[*http://qt-project.org/doc/qt-5.1/qtcore/qabstractanimation.html#details*](http://qt-project.org/doc/qt-5.1/qtcore/qabstractanimation.html#details)

класс QAbstractAnimation является базовым классом всех анимаций. При наследовании данного класса вы можете создать собственную анимацию. Прогресс анимации задаётся её текущим временем, которое измеряется в миллисекундах с момента начала анимации. Значение обновляется автоматически во время работы анимации. В любой момент времени анимация находится в одном из трёх состояний:

* работа,
* пауза или
* остановка.

Состояния могут изменяться при помощи специальных функций. Также есть сигнал об изменении состояния. Анимация может работать циклически, что устанавливается также специальной функцией. Для создания работающих наследников данного класса следует переопределить функции duration() и updateCurrentTime().

### QAnimationGroup

<http://qt-project.org/doc/qt-5.1/qtcore/qanimationgroup.html#details>

класс QAnimationGroup является абстрактным базовым классом для группы анимаций. Это контейнер для анимаций, т.е. он решает, когда начать, остановить или поставить на паузу их. Также можно создавать сложные графы анимаций. При этом следует отметить, что начинание подгрупп в таком графе не поддрерживается и может привести к неправильному поведению. Все анимации в контейнере уничтожаются при удалении контейнера. Также класс предоставляет различные функции для управления содержимым контейнера.

### QParallelAnimationGroup

<http://qt-project.org/doc/qt-5.1/qtcore/qparallelanimationgroup.html>

класс QParallelAnimationGroup обеспечивает группу параллельных анимаций. Группа заканчивается, когда заканчивается наиболее продолжительная анимация.

### QPauseAnimation

<http://qt-project.org/doc/qt-5.1/qtcore/qpauseanimation.html>

класс QPauseAnimation обеспечивает паузу для класса [QSequentialAnimationGroup](http://qt-project.org/doc/qt-5.1/qtcore/qsequentialanimationgroup.html). Если вы желаете ввести задержку между анимациями, вы можете вставить в контейнер объект данного класса. Продолжительность паузы задаётся либо в конструкторе, либо при помощи специальной функции. *Но в самом классе* [*QSequentialAnimationGroup*](http://qt-project.org/doc/qt-5.1/qtcore/qsequentialanimationgroup.html) *есть функции для добавления пауз, поэтому вам не нужно самостоятельно создавать данный класс.*

### QPropertyAnimation

<http://qt-project.org/doc/qt-5.1/qtcore/qpropertyanimation.html#details>

класс QPropertyAnimation анимирует свойства qt. Класс интерполирует свойства qt. Так как свойства сохраняются в виде типа QVariant, класс наследует QVariantAnimation и поддерживает анимацию таких же метатипов, что и его суперкласс. Класс, который объявляет свойства, должен быть QObject. Имя свойства и экземпляр, который будет анимаироваться, помещаются в конструктор.

### QSequentialAnimationGroup

<http://qt-project.org/doc/qt-5.1/qtcore/qsequentialanimationgroup.html#details>

класс QSequentialAnimationGroup обеспечивает последовательную группу анимаций. Он запускает анимации последовательно. Данная группа может быть обработана как другая простая анимация.

### QVariantAnimation

<http://qt-project.org/doc/qt-5.1/qtcore/qvariantanimation.html>

класс QVariantAnimation обеспечивает абстрактный базовый класс для анимаций. Это абстрактный класс. Он имеет чисто виртуальный метод updateCurrentValue(). Данный класс выполняет интерполяцию QVariant, но использование интерполяции он оставляет своим подклассам. Для реакции на изменения текущего значения вы должны переопределить функцию updateCurrentValue(). Также возможно установить значения на некоторые определённые шаги при анимации. Интерполяция будет тогда учитывать данные точки в определённые моменты времени. Функции setEasingCurve() и setDuration() позволяют настроить способ интерполяции. Также можно переопределить функцию interpolated(). Этот класс является альтернативой, когда вы желаете анимировать простые значения, не делая их свойствами qt. Но в большинстве случаев лучше объявить ваше значение как свойство. *Также показано, как зарегистрировать функцию интерполяции для своих собственных типов данных QVariant, так как не все они поддериваются при анимации. Там перечислены те, которые поддерживаются.*

### QEasingCurve

<http://qt-project.org/doc/qt-5.1/qtcore/qeasingcurve.html#details>

класс QEasingCurve обеспечивает упрощающие кривые для управления анимацией. Эта функция контролирует, как изменяется скорость интерполяции между 0 и 1. Данный класс используется вместе с [QPropertyAnimation](http://qt-project.org/doc/qt-5.1/qtcore/qpropertyanimation.html).

### QTimeLine

<http://qt-project.org/doc/qt-5.1/qtcore/qtimeline.html>

класс QTimeLine обеспечивает сроки для управления анимацией. Наиболее часто он используется для анимации элементов управления графического интерфейса пользователя путём вызова периодически слота. Линию можно построить, передав её продолжительность в миллисекундах конструктору. Эта величина описывает, как долго анимация будет работать. *Как я понимаю данный класс высылает через определённые промежутки времени некоторый сигнал, который можно связать с виджетом, так что периодически определённое количество раз будет выполняться некоторый слот.* Также данный класс содержит функции для циклического запуска времени и т.д.

## КАРКАС МАШИНЫ СОСТОЯНИЙ

[*http://qt-project.org/doc/qt-5.1/qtcore/statemachine-api.html*](http://qt-project.org/doc/qt-5.1/qtcore/statemachine-api.html)

машина состояний обеспечивает классы для создания и выполнения графов состояний. Семантика машины состояний основана на State Chart XML (SCXML). *Тут прямо указывается ссылка на 2 источника, которые можно использовать. Именно их я и буду изучать.* Диаграммы состояния обеспечивают графический способ моделирования того, как система реагирует на стимулы. Это делается при помощи определённых состояний, в которых может находится система, и как она может переходить из одного состояния в другое. Ключевой характеристикой систем, основанных на событиях, является тот факт, что поведение часто зависит не только от последнего события, но и от событий, предшествовавших ему. При помощи диаграмм состояния данная информация легко выражается. Машина состояний обеспечивает интерфейс и модель выполнения, которая может быть использована для эффективного включения элементов и семантик диаграмм состояний в qt приложения. Данный модуль тесно интегрирован с мета-объектной системой qt.

Граф состояний в данном случае является иерархическим. Одни состояния могут быть вложены в другие состояния, а текущая конфигурация машины состояний состоит из набора состояний, которые сейчас активны. Все состояния должны иметь общего прародителя. Далее приводится небольшой код для демонстрации реализации простой машины состояний: вначале создаётся машина и сами состояния, затем создаются переходы между ними, затем состояния передаются машине, устанавливается начальное состояние, после чего машина запускается. Также есть пример кода, по которому можно в данном состоянии присваивать некоторому свойству класса QObject некоторое значение. *Машина состояний является асинхронной, что значит, что она становится частью цикла обработки событий вашего приложения.*

Чтобы машина остановилась, она должна иметь некоторое конечное специальное состояние. Можно в состояниях задавать иерархию, передавая родительское состояние в конструкторе дочернего состояния. При этом сигналы, связанные с родительским состоянием, будут также работать и для дочернего состояния. Дочерние состояния наследуют переходы, но они также могут их и переписывать. Также можно создавать состояния истории, которые используются для того, чтобы можно было затем вернуться к некоторому состоянию. Оно делается дочерним для данного состояния. Затем, возвращаясь к нему, мы просто возвращаемся к его родителю. Но почему нельзя вернуться к родителю напрямую???

Предположим, что у некоторой сущности есть два свойства, каждое из которых булево. Тогда возможных комбинаций состояний такой машины – 4, а возможных переходов – 8. Если добавлять новое свойство, то количество комбинаций будет расти экспоненциально. Но если использовать концепцию параллельных состояний, то только линейно.

Когда параллельная группа состояний вводится, все дочерние состояния одновременно также вводятся. Переходы внутри отдельных дочерних состояний работают нормально. Однако, любое дочернее состояние может осуществить переход, который выйдет из родительского состояния. Когда это случается, родительское состояние и все его дочерние состояния также покидаются. Параллелизм машины состояний следует семантике чередований. Все параллельные операции будут выполнены за один, атомарный шаг обработки события, таким образом, ни одно событие не может прервать параллельные операции. Однако события будут всё ещё обрабатываться последовательно, так как машина является однопоточной. Например: рассмотрим ситуацию, где есть два перехода, которые выходят из одной и той же группы параллельных состояний, и их условия становятся истиной одновременно. В этом случае событие, которое обработается последним из двух не даст никакого эффекта, так как первое событие уже вызовет выход машины из параллельного состояния.

Также можно создавать конечные состояния конкретных групп состояний, что замечательно инкапсулирует их от внешнего мира, так что всё, что нужно знать о данном сложном состоянии – это когда машина вошла в него, а когда она вышла из него. Переход не обязательно должен иметь целевое состояние. Такой переход не вызывает никакого изменения состояния. Это даёт вам возможность реагировать на некоторое событие без оставления данного состояния. Машина состояний запускает свой собственный цикл обработки событий. Вы можете также постить свои собственные события для машины состояний. Для этого следует создать подкласс класса абстрактного перехода и переопределить метод eventTest(), в котором вы проверяете то, совпадает ли событие с событием вашего типа. *Есть пример кода, но я пока разобрался с данным примером кода лишь частично.*

Может так случится, что состояние не будет обработано никаким переходом. В результате оно будет просто молча израсходовано. Но лучше объединить все состояния и создать переход, который обрабатывает все такие состояния (переход по умолчанию).

В Глубоко вложенных диаграммах состояния можно также добавить такой переход для возврата на наиболее подходящий уровень.

Если вы желаете, чтобы при выходе из некоторого состояния, которое явно задавало свойство, в другое состояние, где этого нет, свойство возвращалось в начальное значение, то вы можете установить Глобальную политику перезаписи. *Есть пример кода.* Если при этом значение свойства в некотором состоянии не задано, то машина автоматически найдёт ближайший прародитель, в котором данное свойство задаётся. Если такого не найдено, то свойство будет перезаписано в начальное значение (до любого присваивания в состояниях). *Данное свойство я понял нормально.*

Программный интерфейс машины состояний соединён с программным интерфейсом анимации. Это позволяет автоматически анимировать свойства, которые присваиваются в состояниях. Для этого необходимо в объект перехода между состояниями добавить объект анимации. При использовании политики Глобальной перезаписи можно также добавлять анимацию при перезаписи свойства.

При использовании анимации может так произойти, что нельзя определить, чему равняется сейчас свойство, так как оно непрерывно изменяется. Но иногда бывает полезно определить тот момент, когда свойство достигнет значения, которое задаётся данным состоянием. Там есть для этих целей специальный сигнал. *Также там показан понятный мне пример кода.*

Что произойдёт, если состояние будет оставлено прежде окончания анимации? В этом случае поведение машины состояний зависит от целевого состояния перехода. Если данное состояние явно определяет значение свойства, то никаких дополнительных действий не производится. Свойству будет присвоено значение данного состояния. Если целевое состояние не присваивает свойству значение, то есть две настройки: значение будут таким, как в состоянии, которое было оставлено (по умолчанию); при установке Глобальной политики перезаписи свойство будет сохранено как обычно.

Если вы желаете, чтобы изменение значения некоторого свойства сопровождалось некоторой конкретной анимацией независимо от состояния, то к машине можно добавить анимацию по умолчанию. Но заметьте, что анимации, явно установленные для переходов, имеют приоритет перед анимациями по умолчанию.

Машина состояний является подклассом класса состояния. Поэтому можно создавать вложенные машины состояний. Настоящая машина состояний обрабатывает вложенную машину состояний, как атомарное состояние. Дочерняя машина состояний имеет свою собственную очередь событий и конфигурацию. Но конфигурация дочерней машины не является частью конфигурации родительской машины. Состояния дочерних и родительских машин состояний не могут быть целевыми для переходов из друг друга соответственно.

### QKeyEventTransition

<http://qt-project.org/doc/qt-5.1/qtwidgets/qkeyeventtransition.html>

класс QKeyEventTransition обеспечивает переходы для событий от клавиатуры.

### QMouseEventTransition

<http://qt-project.org/doc/qt-5.1/qtwidgets/qmouseeventtransition.html#details>

класс QMouseEventTransition обеспечивает переходы для событий мыши.

### QAbstractState

<http://qt-project.org/doc/qt-5.1/qtcore/qabstractstate.html#details>

класс QAbstractState является базовым классом для состояний класса [QStateMachine](http://qt-project.org/doc/qt-5.1/qtcore/qstatemachine.html). Это абстрактный базовый класс. Он определяет общий для всех объектов состояний интерфейс. Есть два сигнала, испускаемые при выходе и входе в состояние. Также есть функция для возврата родительского состояния. А также функция для машины состояния. Для подкласса следует определить функцию onEntry(), которая вызывается, когда осуществляется вход в состояние. Также переопределите функцию onExit().

### QAbstractTransition

<http://qt-project.org/doc/qt-5.1/qtcore/qabstracttransition.html#details>

класс QAbstractTransition является базовым классом для переходов между объектами состояниями. Также есть функции для возврата машины, а также начального и конечного состояний. Также есть сигнал о том, что переход был вызван. Также есть функция для добавления анимаций в переход. Функция eventTest() вызывается машиной состояний, чтобы определить, следует ли вызывать переход в ответ на некоторое событие. Также есть функция, переопределив которую, можно добиться частной обработки перехода.

### QEventTransition

<http://qt-project.org/doc/qt-5.1/qtcore/qeventtransition.html#details>

класс QEventTransition обеспечивает специфические для QObject переходы из-за событий qt. Данный класс связывает некоторое событие с конкретным объектом. *Есть пример кода*. *также в данной части показано, как создавать подкласс данного класса. Особый акцент делается на механизме переопределения функции eventTest().*

### QFinalState

[*http://qt-project.org/doc/qt-5.1/qtcore/qfinalstate.html#details*](http://qt-project.org/doc/qt-5.1/qtcore/qfinalstate.html#details)

класс QFinalState обеспечивает конечное состояние. Данный класс используется для сообщения о том, что данная машина закончила работу. Когда данное состояние достигается, то машина испускает сигнал об окончании работы. В общем случае, когда конечное подсостояние достигается, то испускается сигнал родительского состояния об окончании. Для использования данного класса следует создать конечное состояние и добавить его в переход в качестве целевого, а сам переход добавить в некоторое другое состояние.

### QHistoryState

<http://qt-project.org/doc/qt-5.1/qtcore/qhistorystate.html#details>

класс QHistoryState обеспечивает средства для возврата в предыдущее активное подсостояние. Это псевдо-состояние, которое представляет дочернее состояние, в котором родительское состояние было в последний раз, когда из него вышли. Также можно определить специальную функцию setDefaultState(), чтобы установить состояние, в которое следует войти, если в родительское состояние никогда не выполнялось входа. По умолчанию состояние истории является неполным в том смысле, что оно не помнит вложенных состояний. Это можно настроить через определённое свойство данного класса.

### QSignalTransition

<http://qt-project.org/doc/qt-5.1/qtcore/qsignaltransition.html#details>

класс QSignalTransition обеспечивает переход, основанный на сигнале qt. Обычно вам следует использовать перегруженную функцию [QState::addTransition](http://qt-project.org/doc/qt-5.1/qtcore/qstate.html#addTransition)(), которая принимает отправителя и сигнал как аргументы, чем создавать объект данного класса напрямую. Этот класс можно специализировать, чтобы сделать переход по сигналу условным.

### QState

<http://qt-project.org/doc/qt-5.1/qtcore/qstate.html#details>

класс QState обеспечивает состояние общего назначения для машины состояний. Объекты данного класса могут иметь дочерние состояния, а также могут иметь переходы в другие состояния. Также состоянию можно приписывать некоторое конкретное значение некоторого свойства. Состояния наиболее высокого уровня должны задавать машину состояний как своё родительское состояние или должны быть добавлены к машине состояний. Также в данном классе есть свойство, которое определяет, как мода дочернего состояния обрабатывается. Дочерние состояния являются взаимно исключающими и машине нужно знать, в какое состояние войти (из дочерних) при входе в данное состояние. Когда выполняется вход в конечное состояние данного состояния, то вызывается сигнал о завершении состояния. Также можно устанавливать состояние ошибки данного состояния. Туда осуществляется переход, если возникает ошибка при входе в состояние.

### QStateMachine

<http://qt-project.org/doc/qt-5.1/qtcore/qstatemachine.html#details>

класс QStateMachine обеспечивает машину иерархических конечных состояний. Данный класс базируется на понятиях и обозначениях диаграмм состояния. Машина управляет набором состояний и набором переходов между состояниями. Эти состояния и переходы определяют граф состояний. Если данный граф был построен, то машина состояний выполнит его. Не рекомендуется удалять состояния, пока машина состояний работает. Также перед началом работы должно быть установлено начальное состояние. Машина управляется событиями и содержит свой собственный цикл обработки событий. События постятся в машину при помощи определённой функции. Это означает, что машина работает асинхронно и она не будет развиваться без работающего цикла обработки событий (*не понимаю*). Вам не нужно, как правило, постить события напрямую как переходы qt. Но для собственных переходов, управляемых событиями, данная функция может оказаться полезной (*для понимания частных событий следует реализовать свой собственный пример*). Машина работает, пока не осуществится вход в конечное состояние наивысшего уровня. Также можно остановить машину состояний явно. Если вдруг встретится некоторая ошибка, то машина будет искать состояние ошибки, и если оно окажется доступным, то машина войдёт в данное состояние. Возможные типы ошибок описываются перечислением Error. После входа в состояние ошибки можно также получить тип ошибки. Если осуществится вход в состояние ошибки, то выполнение машины не прервётся. Если этого не случится, то машина остановится и на консоли появится сообщение об ошибке.

### SignalEvent

<http://qt-project.org/doc/qt-5.1/qtcore/qstatemachine-signalevent.html#details>

класс SignalEvent представляет сигнальное событие qt. Данное событие генерируется машиной состояний в ответ на сигнал qt. Можно получить объект, который выслал сигнал, а также индекс сигнала, а также аргументы сигнала.

### WrappedEvent

<http://qt-project.org/doc/qt-5.1/qtcore/qstatemachine-wrappedevent.html#details>

класс WrappedEvent наследует QEvent и удерживает клон события, связанного с QObject. Событие в обёртке генерируется машиной состояний в ответ на событие qt. Здесь можно вернуть объект, который сгенерировал событие, а также клон самого оригинального события.

## ПЛАГИНЫ В QT

<http://qt-project.org/doc/qt-5.1/qtcore/plugins-howto.html>

qt предоставляет два программных интерфейса для создания плагинов:

* Высокоуровневый для написания расширений для самого qt: частные драйверы базы данных, форматы изображений, текстовые кодеки, частные стили и т.д.
* Низкоуровневый интерфейс для расширения qt приложений.

Например, вы желаете написать частный класс QStyle и желаете, чтобы приложения загружали его динамически. Для этого вам следует использовать высокоуровневый интерфейс. Так как высокоуровневый интерфейс построен на основе низкоуровневого, то некоторые вопросы являются общими для обоих. Написание плагина, который расширяет сам qt достигается при помощи специализации подходящих базовых классов плагинов, определении некоторых функций, добавлением некоторых макросов. Существует несколько базовых классов плагинов. Производные плагины по умолчанию сохраняются в поддиректории стандартной директории плагинов. Qt не найдёт плагины, если они сохранены в неправильной директории. Также отмечается, что для большинства плагинов требуется .json файл, содержащий метаданные, описывающие плагин, которые требуются для большинства плагинов. Информация в данном файле является зависимой от плагина. Драйверы базы данных, форматы изображений, кодеки текста и большинство других типов плагинов не требуют явного создания объектов. Qt найдёт и создаст их сам, когда потребуется. Стили представляют исключение, так как вы можете захотеть установить стиль явно в коде.

**The Lower-Level API: Exyending Qt Applications**

не только qt, но также и qt приложения могут быть расширены при помощи плагинов. Это требует того, чтобы приложение задетектировало и загрузило плагины с использованием QPluginLoader. В этом контексте плагины могут обеспечить произвольную функциональность и не ограничены только драйверами к базе данных, форматами изображений, текстовыми кодеками, стилями и другими типами, которые расширяют функциональность qt.

Создание приложений, расширяемых при помощи плагина, включает следующие шаги:

1. Определение набора интерфейсов для коммуникации с плагином.
2. Использование макроса Q\_DECLARE\_INTERFACE() для сообщения мета-объектной системе qt об интерфейсе.
3. Использование класса QPluginLoader в приложении для загрузки плагинов.
4. Использование qobject\_cast() для тестирования, может ли плагин применить данный интерфейс.

Написание плагина включает следующие шаги:

1. Объявление класса плагина, который наследует QObject и интерфейсы, которые плагин желает обеспечить.
2. Использование макроса Q\_INTERFACES(), чтобы сказать мета-объектной системе qt об интерфейсах.
3. Экспортирование плагина с использованием макроса Q\_PLUGIN\_METADATA().
4. Построение плагина с использованием подходящего .pro файла.

Также отмечается, что QCoreApplication должен быть инициализирован прежде, чем плагины будут загружены.

Приложения qt автоматически знают, какие плагины доступны, так как плагины сохраняются в стандартных поддиректориях плагинов. Во время разработки директория для плагинов – это QTDIR/plugins (QTDIR – директория, где установлен qt), причём каждый тип плагина находится в своей поддиректории.

Если вы желаете, чтобы ваше приложение использовало плагины и вы не желаете использовать стандартный путь плагинов, то при разработке процесса установки определите путь, который вы желаете использовать для плагинов, и сохраните путь, т.е. с использованием QSettings, которое считает приложение при работе. Тогда приложение вызовет функцию [QCoreApplication::addLibraryPath](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html#addLibraryPath)() с этим путём и ваши плагины будут доступны для приложения. Заметьте, что конечная часть пути не может быть изменена (*вот этого я не совсем понял*). Это очень важная тема для инсталятора.

Если вы желаете, чтобы плагин был загружаемым, то единственный подход – это создать поддиректорию под приложение и помесить плагин в эту поддиректорию. Если вы распространяете любой из плагинов, который приходит вместе с qt, вы должны копировать поддиректорию под плагины, где плагин находится в корневой папке ваших приложений. (*последнего я не понял*)

Наиболее обычный и более гибкий способ включить плагин в приложение – это скомпилировать его в динамическую библиотеку, которая предоставляется отдельно, обнаруживается и загружается динамически во время выполнения. Также плагины могут связываться статически с вашим приложением. Использование статических плагинов делает развёртывание менее подверженным ошибкам, но также и имеет недостаток, который связан с тем, что никакая функциональность не может быть добавлена при помощи плагина без полной пересборки и перераспределения приложения. *Далее приведён перечень статических плагинов, которые предоставляет qt при установке.*

Для статической связи с данными плагинами вам необходимо добавить требуемые плагины в ваше построение с использованием QTPLUGIN. Q\_IMPORT\_PLUGIN() макрос также необходим в коде приложения, но те автоматически генерируются qmake и добавляются в проект вашего приложения. если вы желаете, чтобы все плагины, добавленные в переменную QTPLUGIN, были автоматически связаны, удалите remove\_plugins из переменной CONFIG. данная тема также важна, так как позволит включить в приложение статические плагины.

Также можно создать собственные статические плагины, проделывая следующие шаги:

1. Добавить CONFIG += static в .pro файл приложения.
2. Используйте Q\_IMPORT\_PLUGIN() макрос для вашего приложения.
3. Свяжите ваше приложение с вашим плагином с использованием LIBS в файле проекта.

*Также есть замечание о том, на что следует обратить внимание, если вы не используете для построения qmake. Это касается наличие директивы препроцессора QT\_STATICPLUGIN. также в данной части ведётся речь о некотором QPA плагине.*

### РАЗВЁРТЫВАНИЕ ПЛАГИНОВ

<http://qt-project.org/doc/qt-5.1/qtdoc/deployment-plugins.html>

При запуске приложения qt вначале обрабатывает выполняемую директорию приложения, как базовую директорию для поиска плагинов. *Есть пример.* также qt просматривает директорию, определяемую через QLibraryInfo::location([QLibraryInfo::PluginsPath](http://qt-project.org/doc/qt-5.1/qtcore/qlibraryinfo.html#LibraryLocation-enum)), которая обычно находится в QTDIR/plugins. Если вы желаете, чтобы qt смотрел в другие места, то вы можете добавить столько путей, сколько вам нужно при помощи вызова [QCoreApplication::addLibraryPath](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html#addLibraryPath)(). Также можно использовать файл qt.conf, чтобы переписать закодированные пути, которые компилируются в qt библиотеку. Есть ссылка на дальнейшее чтение. Также ещё есть другая возможность – это установка переменной среды QT\_PLUGIN\_PATH перед запуском приложения. Если установлено, то qt будет искать плагины в этих путях, определённых в данной переменной.

При загрузке плагинов qt библиотека делает некоторые санитарные проверки, чтобы определить, может или нет плагин быть загружен и использован. Это обеспечивает возможность иметь множественные версии и конфигурации qt библиотеки, установленной бок о бок. *Далее перечислены ограничения на загрузку плагинов qt.*

Если вы настраиваете qt быть построенным как в моде отладки, так и в моде выпуска, но только строите приложения в моде выпуска, вам необходимо гарантировать, что ваши плагины также построены в моде выпуска. По умолчанию, если доступно построение qt в моде отладки, то плагины будут строиться только в моде отладки. Чтобы заставить плагины строиться в моде выпуска, добавьте следующую строку к файлу проекта плагина: CONFIG += release.

Это будет гарантировать, что плагин будет совместим с версией библиотеки, используемой в приложении. Есть некоторые вопросы, которые могут помешать работать корректно написанным плагинам в приложениях, которые разработаны для их использования. Многие из них относятся к отличиям в способе, которым плагины и приложения были построены, часто вырастая из систем построения и процессов. *Далее приводится таблица, в которой описаны проблемы и способы их решения.*

Вы также можете использовать переменную среды QT\_DEBUG\_PLUGINS, чтобы получить диагностическую информацию из qt о каждом плагине, который она пытается загрузить. Установите эту переменную в ненулевое значение в среде, из которой ваше приложение запускается.

### ИСПОЛЬЗОВАНИЕ QT.CONF

<http://qt-project.org/doc/qt-5.1/qtdoc/qt-conf.html>

qt.conf файл переписывает закодированные пути, которые компилируются в qt библиотеке. Эти пути доступны с использованием класса [QLibraryInfo](http://qt-project.org/doc/qt-5.1/qtcore/qlibraryinfo.html). Без данного файла функции в [QLibraryInfo](http://qt-project.org/doc/qt-5.1/qtcore/qlibraryinfo.html) возвращают эти закодированные пути; иначе они возвращают пути так, как они определены в qt.conf. Без этого файла библиотеки qt будут использовать закодированные пути, чтобы искать плагины, переводы и так далее. Эти пути могут не существовать в целевой системе, или они могут быть недоступными. Из-за этого вам нужен qt.conf, чтобы заставить qt библиотеки смотреть где-то ещё. [QLibraryInfo](http://qt-project.org/doc/qt-5.1/qtcore/qlibraryinfo.html) будет загружать данный файл из следующих локаций:

* :/qt/etc/qt.conf с использованием системы ресурсов
* Мак
* В директории, которая содержит исполняемый файл приложения, то есть [QCoreApplication::applicationDirPath](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html#applicationDirPath)() + [QDir::separator](http://qt-project.org/doc/qt-5.1/qtcore/qdir.html#separator)() + "qt.conf".

Данный файл является INI текстовым файлом, как это указано в классе [QSettings](http://qt-project.org/doc/qt-5.1/qtcore/qsettings.html#Format-enum). Данному файлу следует иметь группу Paths, которая содержит вводы, которые соответствуют каждому значению перечисления [QLibraryInfo::LibraryLocation](http://qt-project.org/doc/qt-5.1/qtcore/qlibraryinfo.html#LibraryLocation-enum). *Полезный файл! Следует воспользоваться им экспериментально. Далее приведена некоторая таблица с непонятным мне содержанием.*

Абсолютные пути используются в данном файле как специфические. Все пути являются относительными по отношению к префиксу. *А в той таблице были как раз префиксы.*

В Windows и на Х11 префикс является относительным к директории исполняемого файла приложения. *Есть пример файла данного формата.*

### КЛАССЫ ПЛАГИНОВ В КТ

<http://qt-project.org/doc/qt-5.1/qtcore/plugins.html>

#### QLibraryInfo

<http://qt-project.org/doc/qt-5.1/qtcore/qlibraryinfo.html#details>

QLibraryInfo класс обеспечивает информацию о qt библиотеке. Многие кусочки информации установлены, когда qt настраивается и строится. Данный класс предоставляет абстракцию для доступа к той информации. При помощи использования статической функции данного класса приложение может получить информацию о экземпляре qt библиотеки, который приложение использует во время выполнения. Также можно использовать файл qt.conf для переписывания закодированных путей, которые компилируются в qt библиотеке.

#### QIconEnginePlugin

<http://qt-project.org/doc/qt-5.1/qtgui/qiconengineplugin.html#details>

класс QIconEnginePlugin обеспечивает абстрактный базовый класс для частных плагинов QIconEngine. Плагин движка иконок является простым интерфейсом плагина, который делает более лёгким создание частных движков иконок, которые могут быть загружены динамически в приложения через класс QIcon. Последний класс использует суффикс имени файла или ресурса, чтобы определить, какой движок иконок использовать. Написание плагина достигается при помощи специализации этого базового класса, переопределив чисто виртуальную функцию create(), и экспортируя класс при помощи макроса Q\_PLUGIN\_METADATA(). Json метаданные должны содержать список ключей движка иконок, который поддерживает данный плагин. Ключ соответствует суффиксу файла или имени ресурса, используемого, когда плагин создаётся. Ключи являются чувствительными к регистру.

#### QImageIOPlugin

<http://qt-project.org/doc/qt-5.1/qtgui/qimageioplugin.html#details>

класс QImageIOPlugin определяет интерфейс для написания плагина формата изображения. Это фабрика для написания объектов QImageIOHandler, которые используются внутренне классом QImageReader и QImageWriter, чтобы добавить поддержку разных форматов изображений в qt. *Также описано содержание файла json для данного плагина.* Qt использует возвращаемое значение функции capabilities(), чтобы использовать правильный плагин для работы. Если данное имя поддерживают несколько плагинов, то qt выбирает их случайно.

#### QGenericPlugin

<http://qt-project.org/doc/qt-5.1/qtgui/qgenericplugin.html#details>

класс QGenericPlugin является абстрактным базовым классом для плагинов. Определение по умолчанию класса QGenericPluginFactory автоматически обнаруживает плагин и загружает драйвер на ***сервер приложения*** во время выполнения. Json-файл метаданных должен содержать список ключей, поддерживаемых при помощи данного плагина. *Следует в будущем попрактиковаться с данным классом.*

#### QGenericPluginFactory

<http://qt-project.org/doc/qt-5.1/qtgui/qgenericpluginfactory.html#details>

класс QGenericPluginFactory создаёт драйверы плагинов. *И в данном классе, и в предыдущем классе почему-то говорится в некоторых функциях и описаниях о драйверах мыши.*

#### QStylePlugin

<http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleplugin.html#details>

класс QStylePlugin обеспечивает абстрактный базовый класс для плагинов частных стилей QStyle. Данный класс является простым интерфейсом плагинов, который делает лёгким создание частных стилей, которые могут загружаться динамически в приложения с использованием класса QStyleFactory. *Также в данной части указано, как создавать такие плагины и что должен содержать json файл описания плагина.*

<http://qt-project.org/doc/qt-5.1/qtdoc/groups.html> - ПРОСТО НАИПОЛЕЗНЕЙШАЯ ССЫЛКА.

#### QLibrary

<http://qt-project.org/doc/qt-5.1/qtcore/qlibrary.html#details>

класс QLibrary загружает разделяемые библиотеки во время выполнения. Экземпляр объекта данного класса управляет единичным разделяемым объектным файлом (DLL). Данный класс предоставляет доступ к функциональности в этой библиотеке независимым от платформы способом. Для этого данному классу сообщается имя файла (в конструкторе или явно, при помощи специальной функции). Когда загружается библиотека, то данный класс осуществляет поиск во всех определённых для библиотек местах системы, если только не указан абсолютный путь. При указании абсолютного пути класс вначале обращается по нему, но если там нет библиотеки, то класс пробует имена с разными специфическими для платформы префиксами и суффиксами. Это делает возможным определять разделяемые библиотеки только с использованием их основных имён.

К физически одной и той же библиотеке могут обращаться несколько экземпляров данного класса. Типичное использование данного класса – это перерешать экспортируемый символ в библиотеке и вызвать С функцию, которую представляет данный символ. Это называется явным связыванием в отличие от неявного связывания, которое производится при помощи шага связывания в процессе построения, когда происходит связывание исполняемого файла с библиотекой.

#### <QtPlugin>

<http://qt-project.org/doc/qt-5.1/qtcore/plugins.html>

заголовочный файл <QtPlugin> определяет макросы для определения плагинов.

Q\_DECLARE\_INTERFACE( *ClassName, Identifier* ). Данный макрос связывает данный идентификатор с классом интерфейса. Идентификатор должен быть уникальным. Данный макрос обычно используется прямо за определением данного класса в заголовочном файле. Также не следует помещать данный макрос в пространства имён.

Q\_IMPORT\_PLUGIN (*PluginName)* макрос импортирует плагин, соответствующий TARGET, определённой в файле проекта плагина. Если вставить данный макрос в исходный код приложения, то можно использовать статический плагин. Статические плагины также должны быть включены сборщиком в файл проекта в соответствующую переменную.

Q\_PLUGIN\_METDATA(…) макрос используется, чтобы объявить метаданные, которые являются частью плагина, который инстанцирует данный объект. Данный макрос содержит идентификатор используемого интерфейса и ссылку на файл, который содержит метаданные для плагина. В исходном коде плагина должно быть только одно появление данного макроса. Также данный макрос может использоваться только с классом, имеющим конструктор по умолчанию. Файл должен находится в одной из директорий, определённых системой построения. moc завершится с ошибкой, когда он не найдёт определённый файл. Данная функция была введена только в qt кор5.

## СИСТЕМА СОБЫТИЙ

<http://qt-project.org/doc/qt-5.1/qtcore/eventsandfilters.html>

в qt события – это объекты, производные от класса QEvent, который представляет из себя вещи, которые случаются или внутри приложения, или как результат внешней деятельности, о которой приложению нужно знать. События могут быть посланы или обработаны любым экземпляром подкласса QObject, но они особенно подходят для виджетов. Когда событие происходит, qt создаёт объект события, чтобы представить его при помощи конструирования экземпляра соответствующего подкласса QEvent, и доставляет его в некоторый экземпляр QObject при помощи вызова его функции event(). Эта функция не обрабатывает событие самостоятельно; но она на основании типа полученного события вызывает обработчик события для данного конкретного типа и высылает ответ, основанный на том, было ли событие принято или проигнорировано. Некоторые события приходят из системы окон (QMouseEvent, QKeyEvent); другие, такие как QTimerEvent, приходят из других источников; некоторые приходят из самого приложения. Каждый специализированный класс события наследует QEvent и добавляет некоторую собственную функциональность. Некоторые классы поддерживают в действительности несколько типов событий: одинарный и двойной щелчок мыши. Каждое событие имеет связанный с ним тип: QEvent::Type и это может быть использовано как удобный источник информации о типе во время выполнения, чтобы быстро определить, из какого подкласса данный объект события был сконструирован. Так как программам необходимо реагировать способами, которые являются сложными и изменяющимися, то механизм доставки qt является гибким. Документация [QCoreApplication::notify](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html#notify)() кратко рассказывает полную историю. *Также в данной части есть ссылка на статью, которая рассказывает данную историю менее кратко.* Здесь же авторы объяснят информацию, достаточную для 95 % приложений.

Обычный способ для доставки событий – это вызов виртуальной функции. Данные функции являются подходящими для реакции на данное событие. Если вы не выполняете всю работу в реализации вашей функции, то вы можете вызвать функцию базового класса. Если вы желаете уж очень специфического поведения, которое не обеспечивается текущими функциями, то вам необходимо переопределить функцию event().

Иногда так случается, что объекту необходимо смотреть и перехватывать события, которые послали другим объектам. для этого предназначены некоторые функции, которые могут ловить события, которые ещё не дошли до целевого объекта. эти функции являются фильтрами. Если установить фильтр для объекта QApplication или QCoreApplication, то фильтроваться будут все сообщения приложения. Многие приложения желают создавать и посылать свои собственные события. Вы можете посылать события точно так же, как это делает собственный цикл обработки событий qt, при помощи построения подходящих объектов-событий и посылки их при помощи функций [QCoreApplication::sendEvent](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html#sendEvent)() and [QCoreApplication::postEvent](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html#postEvent)(). Первая функция обрабатывает событие немедленно. Когда она возвращает значение, то фильтр или сам объект уже обработали данное событие. Вторая функция помещает событие в очередь для дальнейшей отправки. Когда в следующий раз Главный цикл обработки событий qt запустится, то он вышлет все данные события с некоторой степенью оптимизации. Например, если существует несколько событий увеличения размера окна, то они комплектуются в одно.

При создании виджета следует обеспечивать создание всех его членов данных до того, как он может обработать некоторые события. Для этого создавайте все члены данных в его конструкторе.

При создании событий частного типа вам следует определить номер события, который должен быть больше, чем QEvent::User, а также вам может понадобиться специализировать класс QEvent для того чтобы передать определённую информацию о вашем частном событии.

### QAbstractEventDispatcher

<http://qt-project.org/doc/qt-5.1/qtcore/qabstracteventdispatcher.html#details>

класс QAbstractEventDispatcher обеспечивает интерфейс для управления очередью событий qt. Он получает сообщения от оконной системы и других источников. Далее он посылает их объектам приложения для обработки и доставки. Данный класс предоставляет очень точный контроль над доставкой событий. Для простого контроля обработки событий используйте [QCoreApplication::processEvents](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html#processEvents)(). При помощи специальной функции можно получить объект данного класса (*похоже на одиночку*), либо же можно создать свой собственный и затем установить его при помощи специальной функции класса QCoreApplication. Главный цикл обработки событий начинается при вызове функции [QCoreApplication::exec](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html#exec)() и заканчивается при вызове функции [QCoreApplication::exit](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html#exit)(). Локальный цикл обработки событий может быть создан с использованием класса QEventLoop. Программы, которые выполняют длинные операции, могут вызвать функцию processEvents() вместе с побитовой ИЛИ комбинацией разных флагов QEventLoop::ProcessEventsFlag, чтобы управлять тем, какие события следует доставить. Также данный класс разрешает интегрирование внешнего цикла обработки событий и цикла обработки событий qt.

### QBasicTimer

<http://qt-project.org/doc/qt-5.1/qtcore/qbasictimer.html#details>

класс QBasicTimer обеспечивает события таймера для объектов. Это быстрый, лёгкий и низкоуровневый класс, который используется qt внутри. Авторы рекомендуют использовать в приложениях более высокоуровневый класс QTimer. Этот таймер является повторяющимся: он высылает последовательные события таймера до тех пор, пока функция stop() не будет вызвана. *Далее описаны некоторые функции данного класса, а также говорится о том, как использовать данный класс.*

### QChildEvent

<http://qt-project.org/doc/qt-5.1/qtcore/qchildevent.html#details>

класс QChildEvent содержит параметры события для событий дочерних объектов. Данные события высылаются немедленно объектам, когда дети добавляются или удаляются.

### QDynamicPropertyChangeEvent

<http://qt-project.org/doc/qt-5.1/qtcore/qdynamicpropertychangeevent.html#details>

класс QDynamicPropertyChangeEvent содержит параметры события для событий изменения динамических свойств. Они высылаются объектам, когда свойства динамически добавляются, изменяются или удаляются с использованием функции QObject::setProperty().

### QEvent

<http://qt-project.org/doc/qt-5.1/qtcore/qevent.html#details>

класс QEvent является базовым классом для всех классов событий. Объекты событий содержат параметры событий. Цикл обработки событий qt получает события нативной системы окон из очереди событий, транслирует их в QEvents и высылает переведённые события в объект QObject. В общем, события приходят из ниже лежащей системы окон, но также возможно высылать события вручную с использованием функций класса QCoreApplication. Функция spontaneous() проверяет данное обстоятельство. Объекты QObject получают события при помощи вызова их функций QObject::event(). Функция может быть переопределена в подклассах, чтобы настроить обработку событий и добавить дополнительные типы событий. По умолчанию события отправляются к обработчикам событий (специальным функциям). Функция [QObject::installEventFilter](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html#installEventFilter)() позволяет объекту перехватывать события, предназначенные для другого объекта. Базовый класс QEvent содержит только параметр типа события и флаг «принять» (можно игнорировать). Подклассы данного класса содержат дополнительные параметры, которые описывают частное событие.

### QTimerEvent

<http://qt-project.org/doc/qt-5.1/qtcore/qtimerevent.html#details>

класс QTimerEvent содержит параметры, которые описывают события таймера. Данные события высылаются через периодические интервалы объектам, которые начали один или более таймеров. Каждый таймер имеет уникальный идентификатор. Обработчик [QObject::timerEvent](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html#timerEvent)() получает события таймера.

### QTimer

<http://qt-project.org/doc/qt-5.1/qtcore/qtimer.html#details>

класс QTimer обеспечивает повторяемые и однократные таймеры. Класс обеспечивает высокоуровневый интерфейс для таймеров. Для его использования создайте объект данного класса, соедините его сигнал timeout() с соответствующим слотом и вызовите функцию start(). Из последней он будет высылать сигнал timeout() через постоянные интервалы. *Интервал задаётся в конструкторе.*

Вы можете установить таймаут таймера только раз при помощи вызова функции setSingleShot(true). В многопоточных приложениях использовать таймер можно в любом потоке, который имеет цикл обработки событий. Невозможно запустить таймер из других потоков.

Таймеры не начинаются ровно в то время, которое написано. Точность зависит от системных таймеров. Точность зависит от операционной системы и железа. Большинство платформ поддерживают разрешение в 1 мс, хотя во многих реальных ситуациях точность таймера не равняется данному разрешению. Если qt не способен доставить требуемое число кликов таймера, он просто отбрасывает их. Альтернативами данному таймеру являются [QObject::startTimer](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html#startTimer)(), переопределение [QObject::timerEvent](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html#timerEvent)(), а также использование класса QBasicTimer.

### ТАЙМЕРЫ

<http://qt-project.org/doc/qt-5.1/qtcore/timers.html>

класс QObject обеспечивает базовую поддержку таймеров в qt. Вы можете начать таймер с [QObject::startTimer](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html#startTimer)() с интервалом в миллисекундах в качестве аргумента. Функция возвращает уникальный идентификационный номер таймера. Для завершения таймера необходимо вызвать функцию [QObject::killTimer](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html#killTimer)() с идентификатором таймера в качестве аргумента. Для работы данного механизма приложение должно работать в цикле обработки событий. Вы начинаете приложение при помощи функции [QApplication::exec](http://qt-project.org/doc/qt-5.1/qtwidgets/qapplication.html#exec)(), после чего таймер загорается и приложение высылает сообщение [QTimerEvent](http://qt-project.org/doc/qt-5.1/qtcore/qtimerevent.html) и поток управления оставляет цикл обработки событий до тех пор, пока происходит обработка события таймера. Это предполагает, что таймер не может загораться, пока ваше приложение занято произведением некоторых других действий. Другими словами, точность таймера определяется детализированностью вашего приложения.

Главный программный интерфейс для функционирования таймера обеспечивается классом QTimer. Данный таймер характеризуется наличием сигналов и слотов, так как он наследует класс QObject. Также данный класс обеспечивает статическую функцию для single-shot timers. Для того чтобы таймер заработал, вы должны иметь в вашем приложении цикл обработки событий, т.е. должен быть совершён вызов функции [QCoreApplication::exec](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html#exec)(). События таймера будут доставляться, только пока работает данный цикл обработки событий. Если вы уже имеете подкласс QObject и хотите лёгкой оптимизации, вы можете использовать класс QBasicTimer вместо QTimer.

### QGestureEvent

<http://qt-project.org/doc/qt-5.1/qtwidgets/qgestureevent.html#details>

класс QGestureEvent обеспечивает описание triggered gestures. Он содержит список жестов, которые могут быть получены с использованием специальной функции. Жесты могут быть или активны, или отменены. *Также в данной части описаны некоторые функции и есть ссылка на следующую часть.*

### ЖЕСТЫ В ВИДЖЕТАХ И ГРАФИЧЕСКИХ ПРЕДСТАВЛЕНИЯХ

<http://qt-project.org/doc/qt-5.1/qtwidgets/gestures-overview.html>

Из-за отсустсвия необходимости эту часть конспекта я пока не читал. qt включает каркас для программирования жестов, который имеет возможность формировать жесты из набора событий, независимо от использованных методов ввода. Жест может быть некоторым частным движением мыши, действием по прикосновению к экрану, серией событий из другого источника. Природа ввода, интерпретация жеста и взятое действие выбираются на усмотрение разработчика. QGesture – это центральный класс, обеспечивающий контейнер для информации о жестах, выполняемых пользователем. *В данной части перечислены некоторые специальные классы жестов.* Разработчики могут также применять новые жесты при помощи создания подклассов и расширения класса QGestureRecognizer. Добавление поддержки для новых жестов включает применение кода для распознания жеста из событий ввода. Для разрешения определённого жеста на некотором устройстве следует использовать функцию grabGesture(). Когда пользователь выполняет жест, то события QGestureEvent будут доставлены к целевому объекту, а они, в свою очередь, будут обработаны при помощи переопределения функции event() для виджетов или sceneEvent() для объектов графики. Также одно такое событие может содержать много жестов (QGesture), а уже приложение решает, как ими воспользоваться. Каждый жест в данном событии может быть индивидуально либо принят, либо проигнорирован. Хорошей практикой является явное разрешение или отмена жеста. Общее правило таково, что при отмене жеста именно вы делаете это. Игнорирование жеста обозначает, что либо он направляется другому целевому объекту, либо он отменяется. Каждый объект жеста имеет несколько состояний, через которые он проходит. Обычно ввод пользователя изменяет состояние жеста. Первый раз частный жест доставляется к виджету или графическому элементу в состоянии Qt::GestureStarted.

* Принятие жеста означает, что виджет действует на жест: Qt::GestureUpdateState.
* Игнорирование жеста означает, что жест никогда не будет предложен вам снова. Он будет предложен родительскому виджету или элементу в нём.
* Вызов setWidgetCancelPolicy() на жест, когда он находится в начальном состоянии, и также принимается, может вызвать отмену других жестов.

*Там нужно просто взять и выучить функции для управления состояниями жеста, когда понадобится. Также в данной части есть пример собственного кода. Также можно создавать частные распознаватели жестов, а также частные жесты. Как это делать, написано в данной части.*

### QEnterEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qenterevent.html#details>

класс QEnterEvent содержит параметры, которые описывают событие ввода. Данное событие происходит, когда курсор мыши вводится в окно или виджет.

### QInputEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qinputevent.html#details>

класс QInputEvent является базовым классом, который описывает ввод пользователя.

### QMouseEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qmouseevent.html#details>

класс QMouseEvent содержит параметры, которые описывают события мыши. Данные события происходят, когда кнопка мыши нажимается или отпускается внутри виджета, или когда курсор мыши двигается. Событие движения мыши генерируется только тогда, когда нажата кнопка мыши. Если событие мыши не обрабатывается каким-либо из ваших виджетов, то вам следует установить игнорирование данного события. Если события мыши распространяются на виджет, для которого Qt::WA\_NoMouseProprgation макрос был установлен, тогда события мыши не распространятся дальше к родительскому виджету (по цепочке). Также можно получить состояние клавиш модификаторов, позицию курсора относительно виджета, также можно получить Глобальную позицию курсора. В классе виджета QWidget::setEnabled() можно дать возможность или отменить возможность использовать события мыши или клавиатуры в данном виджете. Для получения событий в ваших виджетах вам следует переопределить функции обработки некоторых событий мыши.

### QHoverEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qhoverevent.html#details>

класс QHoverEvent содержит параметры, которые описывают события мыши. События мыши случаются когда курсор мыши двигается в, из или внутри виджета и если виджет имеет атрибут Qt::WA\_Hover. Есть функции, которые дают текущую позицию мыши и старую позицию мыши. *Также в данной части описано отличие от QMouseEvent. Оно заключается в том, что последнее событие высылается виджету, в котором сейчас находится курсор мыши, тогда как первое событие доставляется всем виджетам (включая родительские), причём первым событие получает виджет, который является самым Главным в иерархии.*

<http://qt-project.org/doc/qt-5.1/qtgui/qwheelevent.html#details>

**КЛАССЫ СОБЫТИЙ**

класс QWheelEvent содержит параметры, которые описывают событие прокрутки.

Данные события генерируются и от колёсика мыши, и от жестов прокрутки внутри приложения.

*Далее перечислены некоторые функции данного класса, а также функция класса виджета, которая используется для получения данного события.*

[*http://qt-project.org/doc/qt-5.1/qtgui/qkeyevent.html#details*](http://qt-project.org/doc/qt-5.1/qtgui/qkeyevent.html#details)

**КЛАССЫ СОБЫТИЙ**

класс QKeyEvent описывает события клавиш. Данные события высылаются виджету вместе с фокусом ввода на клавиатуре, когда клавиши нажаты или отпущены.

*Далее указаны некоторые функции данного класса, а также функции класса виджета, которые следует использовать для получения данного события.*

### QFocusEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qfocusevent.html#details>

класс QFocusEvent содержит параметры события для событий фокуса виджета. Данные события высылаются виджету, когда изменяется фокус ввода с клавиатуры. Данные события случаются благодаря действиям мыши и клавиатуры, системы окон, всплывающих меню, горячих клавиш клавиатуры, или других специфических для приложения причин. Можно получить причину конкретного события смены фокуса. *Далее указаны некоторые функции данного класса, а также функции класса виджета, которые следует использовать для получения данного события.*

### QPaintEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qpaintevent.html#details>

класс QPaintEvent содержит параметры события для событий рисования. Данные события высылаются виджетам, которым нужно обновлять, например, когда часть виджета показывается, так как закрывающий виджет сдвинулся. Событие содержит регион, который необходимо обновить, а также ограничивающий регион прямоугольник. *Также затем идёт речь о paint clipping, но что это такое, я так и не понял.*

### QMoveEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qmoveevent.html#details>

класс QMoveEvent содержит параметры события для событий движения. Данные события посылаются виджетам, которые были сдвинуты относительно их родителей.

### QExposeEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qexposeevent.html#details>

класс QExposeEvent содержит параметры события для событий показа. Данное событие высылается окнам, когда область окна становится недействительной или изменяется видимость окна в оконной системе.

### QResizeEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qresizeevent.html#details>

класс QResizeEvent содержит параметры события для событий изменения размера. Они высылаются виджетам, когда изменяется их размер.

### QCloseEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qcloseevent.html#details>

класс QCloseEvent содержит параметры, которые описывают событие закрытия. Они высылаются окнам, которые пользователь желает закрыть (при выборе пункта «закрыть» в меню файла или при нажатии крестика в углу окна). Также они инициируются, когда вы вызываете функцию close(), когда закрываете виджет в программе. Также можно установить специальный флаг Qt::WA\_DeleteOnClose, который указывает, следует ли спрятать или удалить окно при закрытии. Если вы желаете обеспечить некоторую особенную функцию для обработки закрытия вашего окна, то вам следует переопределить функцию QWidget::closeEvent(), в которой следует проигнорировать событие закрытия. Если вы желаете, чтобы виджет был удалён, когда он закрывается, тогда создайте его с флагом Qt::WA\_DeleteOnClose. Это очень полезно для независимых окон высокого уровня в многооконном приложении. Когда удаляются, QObjects испускают сигнал destroyed(). Если последнее окно высокого уровня закрывается, то испускается сигнал QAplication::lastWindowClosed().

### QIconDragEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qicondragevent.html#details>

класс QIconDragEvent показывает, что началось перетаскивание Главной пиктограммы. Данное событие высылается виджетам, когда Главная иконка окна была перетащена.

### QContextMenuEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qcontextmenuevent.html#details>

класс QContextMenuEvent содержит параметры, которые описывают события контекстного меню. Событие высылается виджету, когда пользователь выполняет действие, связанное с открытием контекстного меню. *В данном событии есть специальный флаг, о котором можно прочитать более подробно.*

### QInpurMethodEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qinputmethodevent.html#details>

класс QInputMethodEvent обеспечивает параметры для событий метода ввода. Событие высылается, когда используется метод ввода текста. Это для языков с нелатинским алфавитом. *Класс интересен для людей, которые хотят быть способными корректно обработать языки со сложным символьным вводом.*

### QTabletEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qtabletevent.html#details>

класс QTabletEvent содержит параметры, которые описывают событие Tablet. *Класс мне не понятен, так как там находятся непонятные для меня понятия.*

### QDropEvent

[*http://qt-project.org/doc/qt-5.1/qtgui/qdropevent.html#details*](http://qt-project.org/doc/qt-5.1/qtgui/qdropevent.html#details)

класс QDropEvent обеспечивает событие, которое высылается, когда выполняется действие перетащить и опустить. Когда виджет допускает события опускания, он получает это событие, если он принял наиболее недавние события QDragEnterEvent и QDragMoveEvent, посланные ему. Также здесь есть специальные функции, которые позволяют определить, может ли виджет ответить на данное событие. Событие опускания содержит предлагаемое действие, доступное из proposedAction(), для виджета, чтобы или принять или проигнорировать его. Если действие может быть обработано виджетом, вам следует вызвать функцию acceptProposedAction(). Так как предлагаемое действие может быть комбинацией значений Qt::DropAction, оно может быть полезным или для выбора одного из данных значений как действия по умолчанию или для опроса пользователя о выборе их предпочтительных действий. Если предпочтительное действие опускания является неподходящим, возможно, из-за того, что ваш частный виджет не поддерживает это действие, вы можете заместить его при помощи любого из возможных действий опускания при помощи вызова функции setDropAction() с предпочитаемым вами действием. Если вы установите значение, которое не присутствует в битовой ИЛИ комбинации значений, возвращаемых при помощи функции possibleActions(), действие копирования по умолчанию будет использовано. Если замещение действия опускания было установлено, вызовите функцию accept() вместо acceptProposedAction(), чтобы завершить операцию опускания. Функция mimeData() обеспечивает данные, опускаемые на виджет в виде QMimeData объекта. Он содержит информацию о MIME типе данных вдобавок к самим данным. Вообще было бы очень желательно подробно экспериментально изучить данный класс, а также всю систему перетаскивания и опускания в целом.

### QDragEnterEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qdragenterevent.html#details>

класс QDragEnterEvent обеспечивает событие, которое высылается виджету, когда выполняется действия перетащить и отпустить. Данное событие всегда немедленно сопровождается событием движения при перетаскивании.

### QDragMoveEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qdragmoveevent.html#details>

класс QDragMoveEvent обеспечивает событие, которое высылается, пока перетаскивание и опускание в процессе. Виджет должен проверить событие, чтобы увидеть, какой тип данных оно обеспечивает, и вызвать функцию accept(), чтобы принять опускание, если оно подходит. Также можно вызвать функцию для ограничения прямоугольника внутри виджета, куда будут вставлены данные.

### QDragLeaveEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qdragleaveevent.html#details>

класс QDragLeaveEvent обеспечивает событие, которое высылается виджету, когда перетаскивание и опускание покидает его. Событию всегда предшествуют событие QDragEnterEvent и набор событий QDragMoveEvent. Если вместо этого высылается событие QDropEvent, то данное событие не высылается.

### QHelpEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qhelpevent.html#details>

класс QHelpEvent обеспечивает событие, которое используется для запроса справочной информации о некотором пункте виджета. Данное событие может быть перехвачено в приложениях, чтобы обеспечить подсказки типа «Что это?» для частных виджетов. *Следует экспериментально изучить работу данного класса, который может оказаться очень полезным для всех моих приложений.*

### QStatusTipEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qstatustipevent.html#details>

класс QStatusTipEvent обеспечивает событие, которое используется, чтобы показать сообщения в строке состояния. Status tips могут быть установлены на виджет с использованием QWidget::setStatusTip(). Они показываются в строке состояния, когда курсор мыши входит в виджет. Также данные отметки могут устанавливаться для действий в меню. *Как это делается, показано в примере кода в данной части.* Также данные элементы обеспечиваются для классов изображений элементов через Qt::StatusTipRole перечисление.

### QWhatsThisClickedEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qwhatsthisclickedevent.html#details>

класс QWhatsThisClickedEvent обеспечивает событие, которое может быть использовано для обработки гиперссылок в тексте “What’s This?”.

### QActionEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qactionevent.html#details>

класс QActionEvent обеспечивает событие, которое генерируется, когда QAction добавляется, удаляется или изменяется. Действия могут быть добавлены к виджету при помощи QWidget::addAction(). Эта функция генерирует событие добавления действия, которое можно обработать, чтобы обеспечить частное поведение.

### QHideEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qhideevent.html#details>

класс QHideEvent обеспечивает событие, которое высылается после того, как виджет спрятан. Это событие посылается прямо перед QWidget::hide(), а также когда окно высокого уровня прячется пользователем. *Поведение данного события связано с функцией spontaneous().*

### QShowEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qshowevent.html#details>

класс QShowEvent обеспечивает событие, которое высылается, когда виджет показывается. Есть два вида событий показа: показать события, вызванные системой окон (spontaneous), и внутренние события показа. Спонтанные события показа высылаются только после того, как система показывает окно; они также высылаются, когда окно высокого уровня перерисовывается, после того как оно свернуто. События внутреннего показа доставляются только перед тем, как виджет становится видимым.

### QFileOpenEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qfileopenevent.html>

класс QFileOpenEvent обеспечивает событие, которое будет высылаться, когда существует запрос открытия файла или url. Данные события будут высланы QApplication::instance(), когда ОС затребует, что файл или url следует открыть. Это событие высокого уровня, которое может быть вызвано разными действиями пользователя в зависимости от среды пользователя. Например, двойной щелчок на иконке файла. Данное событие используется только для уведомления приложения о запросе. Оно может быть безопасно сгенерировано.

### QShortcutEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qshortcutevent.html#details>

класс QShortcutEvent обеспечивает событие, которое генерируется, когда пользователь нажимает комбинацию клавиш. Обычно не нужно использовать данный класс напрямую, так как класс QShortcut обеспечивает весь необходимый интерфейс.

### QWindowStateChangeEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qwindowstatechangeevent.html#details>

класс QWindowStateChangeEvent обеспечивает состояние окна перед изменением данного состояния.

### QTouchEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qtouchevent.html#details>

класс QTouchEvent содержит параметры, которые описывают событие прикосновения. (*пока мне не нужно*)

### QScrollPrepareEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qscrollprepareevent.html#details>

класс QScrollPrepareEvent высылается при подготовке прокрутки. Оно высылается перед прокруткой. Получатель должен принять данное событие, чтобы прокрутка началась. Также не гарантируется, что [QScrollEvent](http://qt-project.org/doc/qt-5.1/qtgui/qscrollevent.html) будет вызвано после данного события. *В данной части есть пример.*

### QScrollEvent

<http://qt-project.org/doc/qt-5.1/qtgui/qscrollevent.html#details>

класс QScrollEvent посылается во время прокрутки. Данное событие высылается, чтобы показать, что получатель следует прокрутить. Обычно получателем является некоторый визуальный элемент. Некоторое внимание следует уделить тому, чтобы не было конфликтов двух данных событий, вызванных из разных источников. Использование QScroller::scrollTo является, однако, безопасным.

## НЕЯВНОЕ РАЗДЕЛЕНИЕ

[*http://qt-project.org/doc/qt-5.1/qtcore/implicit-sharing.html*](http://qt-project.org/doc/qt-5.1/qtcore/implicit-sharing.html)

неявное разделение используется во многих классах qt, чтобы максимизировать использование ресурсов и минимизировать копирование. Неявно разделяемые классы являются как безопасными, так и эффективными, когда передаются в качестве аргументов. Если в них производится запись, то они просто копируются.

Разделяемый класс содержит указатель на блок разделяемых данных, который содержит счётчик ссылок и данные. Когда данный объект создаётся, то он устанавливает счётчик ссылок в 1. Далее счётчик ссылок инкрементируется и декрементируется, когда новый объект начинает ссылаться на разделяемые данные и когда он заканчивает ссылаться на разделяемые данные. Разделяемые данные удаляются, когда счётчик ссылок устанавливается в нуль.

### ГЛУБОКОЕ И НЕПОЛНОЕ КОПИРОВАНИЕ

Когда имеешь дело с копированием, то есть два типа копирования объекта. Мы обычно говорим о

* глубоком копировании (deep copying) и
* неполном копировании (shallow copying).

Глубокое копирование предполагает дубликат объекта. Неполное копирование является просто указателем на блок разделяемых данных. Глубокое копирование является дорогостоящим с точки зрения памяти и процессора. Присваивание объектов (=()) для неявного разделяемых объектов выполняется с использованием неполных копий. Неявное разделение автоматически отделяет данные из разделяемого блока, если объект вот-вот должен измениться и счётчик ссылок больше единицы. Это часто называется copy-on-write value semantic. Неявно разделяемый класс имеет полный контроль над данными. При использовании функций для изменения данных он автоматически отделяет их.

*В данной части перечислены классы qt, которые используют неявное разделение.* В данной части отмечается, что они ведут себя также как и обычные объекты, а неявное разделение есть лишь внутренний механизм их работы. Для создания собственных разделяемых классов следует использовать классы qt QSharedData и QSharedDataPointer, которые ранее уже рассматривались мною.

## XML-поддержка в Qt

#### QXmlStreamReader

<http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamreader.html#details>

QXmlStreamReader класс обеспечивает быстрый анализ для чтения хорошо организованных xml при помощи простого программного интерфейса потоков. Этот класс более быстр и более удобен по сравнению с собственным SAX анализатором qt (смотрите QXmlSimpleReader). В некоторых случаях он может также быть быстрее и более удобной альтернативой для использования в приложениях, которые в противном случае использовали бы DOM дерево (QDomDocument). Он считывает данные как с [QIODevice](http://qt-project.org/doc/qt-5.1/qtcore/qiodevice.html), так и с [QByteArray](http://qt-project.org/doc/qt-5.1/qtcore/qbytearray.html). Qt обеспечивает [QXmlStreamWriter](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamwriter.html) для написания xml. Базовым понятием xml чтеца является представление xml документа как потока признаков, похожих на SAX. Основное отличие в том, как эти признаки сообщаются. При помощи SAX приложение должно обеспечить обработчики (функции обратного вызова), которые получают так называемые xml события от анализатора. При помощи данного класса приложение само управляет циклом и создаёт пул признаков из чтеца, один за другим, как оно нуждается в них. Это делается при помощи [readNext](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamreader.html#readNext)(). При помощи такого метода вы можете разбить анализ xml документа на несколько методов или классов, так как такой метод обеспечивает рекурсивный анализ. Это позволяет следить за собственным состоянием приложения при анализе xml. В функции [readNext](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamreader.html#readNext)().возвращаемое значение является некоторым перечислением, которое возвращает либо начало элемента, либо его конец и т.д. *Далее приведён пример кода обычного цикла с использованием данного класса.* Данный класс автоматически учитывает все пожелания к хорошо оформленному xml документу. Если случилась ошибка во время анализа, то [atEnd](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamreader.html#atEnd)() and [hasError](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamreader.html#hasError)() возвращают истину, а [error](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamreader.html#error)() возвращает ошибку, которая произошла. [errorString](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamreader.html#errorString)(), [lineNumber](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamreader.html#lineNumber)(), [columnNumber](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamreader.html#columnNumber)(), and [characterOffset](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamreader.html#characterOffset)() предназначены для построения соответствующей ошибки или предупреждения. [raiseError](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamreader.html#raiseError)() предназначена для более удобного поднятия частных ошибок. *Есть указание на некоторый пример.* QXmlStream понимает и решает xml пространства имён. Для этого есть некоторые функции. *Далее описаны некоторые аспекты обработки пространства имён при помощи данного класса.* QXmlStreamReader является инкрементным анализатором. Он может быть использован для анализа документа, который предоставляется по кускам. *Далее идёт описание данной возможности, которое я пока также не очень понимаю.* Этот класс является консервативным с точки зрения памяти по своему дизайну, так как он не сохраняет полное дерево xml документа в памяти, но только текущий признак в единицу времени. Также он избегает выделения памяти для множества маленьких строк. Он делает это при помощи сообщения всех строковых данных при помощи класса [QStringRef](http://qt-project.org/doc/qt-5.1/qtcore/qstringref.html) вместо действительных объектов [QString](http://qt-project.org/doc/qt-5.1/qtcore/qstring.html). Это тонкая обёртка вокруг QString подстрок, которая обеспечивает поднабор программного интерфейса QString без выделения памяти и переполнения подсчёта ссылок. можно получить эквивалентный QString объект для каждого из данных объектов. *Следует поэкспериментировать с данным классом более подробно.*

#### QXmlStreamWriter

<http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamwriter.html#details>

QXmlStreamWriter класс обеспечивает xml писателя с простым потоковым программным интерфейсом. Это коллега класса [QXmlStreamReader](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamreader.html) для написания xml. Документ начинается с [writeStartDocument](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamwriter.html#writeStartDocument)() и заканчивается [writeEndDocument](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamwriter.html#writeEndDocument)(). Это неявно закроет все существующие тэги. Тэг элемента открывается при помощи [writeStartElement](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamwriter.html#writeStartElement)(), за которым следует [writeAttribute](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamwriter.html#writeAttribute)() или [writeAttributes](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamwriter.html#writeAttributes)(), а также содержание элементов, а затем [writeEndElement](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamwriter.html#writeEndElement)(). Содержание элементов состоит из символов, ссылок ввода или вложенных элементов. Это делается при помощи [writeCharacters](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamwriter.html#writeCharacters)(), [writeEntityReference](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamwriter.html#writeEntityReference)(), а также последовательного написания [writeStartElement](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamwriter.html#writeStartElement)(). [writeTextElement](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamwriter.html#writeTextElement)() используется для написания конечных элементов, которые не содержат ничего кроме текста. *Далее приводится пример кода.* Данный класс автоматически принимает во внимание префиксы пространств имён. Можно объявлять пространства имён [writeNamespace](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamwriter.html#writeNamespace)() или [writeDefaultNamespace](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamwriter.html#writeDefaultNamespace)(). *Также ещё что-то написано про пространства имён, чего я пока не понял. Далее приводятся некоторые другие функции.* По умолчанию QXmlStreamWriter кодирует xml в кодировке UTF-8. Разные кодировки могут быть поставлены с использованием [setCodec](http://qt-project.org/doc/qt-5.1/qtcore/qxmlstreamwriter.html#setCodec)(). Можно также получить информацию об ошибке.

## JSON ПОДДЕРЖКА В QT

<http://qt-project.org/doc/qt-5.1/qtcore/json.html>

qt обеспечивает поддержку работы с данными в формате JSON. Это формат для кодирования объектный данных, производных от JavaScript, который широко используются как формат данных для обмена в интернете. Поддержка данного формата в qt позволяет легко использовать программный интерфейс qt для анализа, модификации и сохранения json данных. Они также содержат поддержку сохранения этих данных в двоичном формате, который напрямую доступен для отображения и очень быстр для доступа. *Есть ссылка для ознакомления с данным форматом данных. А вот по самой этой ссылке есть ссылка на джавасрипт – также следует освоить в свободное от работы время.* данный файл имеет 4 типа данных: бул, дабл, стринг, массив, объект и нуль. Объект – это коллекция пар «ключ - значение». Объект не может содержать одинаковых ключей, а каждый ключ является строкой. *Приводится пример простого файла данного формата.* Все классы для поддержки данной возможности являются основанными на значении и неявно разделяемыми.

### QJsonArray

<http://qt-project.org/doc/qt-5.1/qtcore/qjsonarray.html#details>

класс QJsonArray инкапсулирует JSON массив.

### QJsonDocument

<http://qt-project.org/doc/qt-5.1/qtcore/qjsondocument.html>

класс QJsonDocument обеспечивает способ чтения и записи JSON-документа. Класс может считывать данный файл из UTF-8 кодировки и из двоичного файла. Кроме других данный класс обеспечивает функции конвертации между объектом данного класса, собственно документом и его двоичным представлением и наоборот.

### QJsonObject

<http://qt-project.org/doc/qt-5.1/qtcore/qjsonobject.html#details>

класс QJsonObject инкапсулирует объект JSON.

### QJsonObject::iterator

<http://qt-project.org/doc/qt-5.1/qtcore/qjsonobject-iterator.html#details>

класс QJsonObject::iterator обеспечивает непостоянный итератор STL-типа для объекта QJsonObject.

### QJsonParseError

<http://qt-project.org/doc/qt-5.1/qtcore/qjsonparseerror.html#details>

класс QJsonParseError используется для сообщения об ошибках во время анализа JSON документа.

### QJsonValue

<http://qt-project.org/doc/qt-5.1/qtcore/qjsonvalue.html#details>

класс QJsonValue инкапсулирует значение JSON. Значение может представлять любой из указанных выше типов. Также у данного класса есть отдельный флаг для обозначения неопределённого значения. Данный класс не обеспечивает в отличие от QVariant неявного преобразования типов значений.

## АЛГОРИТМЫ QT

<http://qt-project.org/doc/qt-5.1/qtcore/qtalgorithms.html#details>

qt обеспечивает ряд глобальных шаблонных функций в <QtAlgorithms>, которые работают с контейнерами для выполнения хорошо известных алгоритмов. Эти алгоритмы можно использовать с контейнерными классами, что обеспечивают итераторы стл. Разные алгоритмы требуют разных типов итераторов. Некоторые алгоритмы накладывают определённые требования на типы данных в контейнере (например, наличие оператора ==() и т.д.).

Типы итераторов:

* Итератор ввода (используется для последовательного чтения данных),
* итератор вывода (используется для записи данных),
* прямой итератор (удовлетворяет требования как для итератора ввода, так и для итератора вывода),
* двунаправленный итератор (по сравнению с прямым итератором также поддерживает операцию --),
* итератор произвольного доступа (наиболее мощный тип итератора).

Каждый из данных типов итераторов поддерживает некоторые операции, *которые также перечислены в данной части.*

## ГЛОБАЛЬНЫЕ ОБЪЯВЛЕНИЯ

[*http://qt-project.org/doc/qt-5.1/qtcore/qtglobal.html#details*](http://qt-project.org/doc/qt-5.1/qtcore/qtglobal.html#details)

Глобальные объявления включают типы, функции и макросы. Содержит типы, которые гарантируют фиксированный размер в битах базовых типов С++. Также в данном классе есть идентификаторы для QMsgType, которые могут быть сгенерированы и отправлены к обработчику сообщений qt. *Эти типы перечислены в данной части.*

Данный файл содержит некоторые функции для сравнения и приведения в порядок значений объектов. Это функции типа qMin(), qAbs() и т.д. также данный файл содержит некоторые функции для генерации некоторых сообщений. *Очень полезные функции.*

Также данный файл содержит ряд макросов, которые определяют, что данное приложение компилируется с использованием определённых платформ. Цель данных макросов, *которые перечислены в данной Главе,* - это добавить специфичный для компилятора или платформы код*.* Также некоторые макросы используются для подключения возможности динамического перевода (во время работы приложения). Также есть макросы для предоставления возможности предупреждений разного уровня утончённости. Также есть макрос для реализации цикла qt foreach.

## ДРУГИЕ КЛАССЫ ЯДРА

### QProcessEnvironment

<http://qt-project.org/doc/qt-5.1/qtcore/qprocessenvironment.html#details>

класс QProcessEnvironment удерживает переменные среды, которые могут быть переданы программе. Среду процесса образует набор значений типа ключ-значение, известных как переменные среды. Данный класс позволяет лёгкую манипуляцию этими переменными. Это означает, что они могут быть использованы вместе с QProcess, чтобы установить среду для дочерних процессов. Он не может быть использован для изменения среды текущего процесса. *В данной части указаны некоторые особенности имён переменных среды в системе юникс.* Данный класс резервирует такие переменные, но не поддерживает манипуляцию переменными, чьи имена или значения не кодируются в текущих локальных настройках.

### QProcess

<http://qt-project.org/doc/qt-5.1/qtcore/qprocess.html#details>

класс QProcess используется для запуска внешних программ и коммуникации с ними. Для начала процесса передайте его имя и аргументы командной строки программы, которую вы желаете запустить, в качестве аргументов функции start(). *также описан другой способ запуска программы.* Данный класс позволяет вам обрабатывать процесс как последовательное устройство ввода/вывода. Вы можете записывать и считывать из процесса также, как осуществлять соединение с использованием [QTcpSocket](http://qt-project.org/doc/qt-5.1/qtnetwork/qtcpsocket.html). Есть функция для получения последнего кода завершённого процесса. Процессы имеют два предопределённых канала вывода:

* стандартный канал вывода (консоль) и
* стандартный канал ошибок (stderr), который обеспечивает вывод ошибок процесса.

Данные каналы представляют два отдельных потока данных. Можно переключаться между ними. Также можно считывать данные из двух каналов.

*Также здесь приведена важная информация о терминологии, связанной с потоками. Очень важно это понимать! (*The terminology for the channels can be misleading. Be aware that the process's output channels correspond to QProcess's *read* channels, whereas the process's input channels correspond to QProcess's *write* channels.*)*

Также для некоторых процессов можно установить переменные среды. Также можно установить директорию работы процесса. Данный класс обеспечивает набор функций, которые позволяют процессу быть использованным без цикла обработки событий при помощи приостановки вызывающего потока, пока определённые сигналы не будут испущены. Вызов данных функций из главного потока приложения может вызвать замерзание вашего интерфейса пользователя. Также есть замечание по поводу использования некоторых команд на виндоус. Некоторые команды не поддерживаются отдельными приложениями, но только собственно командной строкой. В данном случае можно в отдельном процессе вызвать командную строку и выполнить данную команду.

### QSettings

<http://qt-project.org/doc/qt-5.1/qtcore/qsettings.html#details>

класс QSettings обеспечивает подходящие платформенно независимые настройки приложения. Пользователи ожидают, что приложение будет запоминать свои настройки между сессиями. Эта информация часто сохраняется в системных регистрах Windows. На юникс в силу отсутствия стандарта многие приложения используют INI-файлы. Данный класс основан на классе QVariant. Если всё, что вам необходимо – это несовместимая основанная на памяти структура, рассмотрите использование QMap<QString,QVariant>. При создании данного объекта вы должны сообщить ему имя вашей компании и имя вашего приложения. Если данный класс часто используется в приложении, то вы можете определить имена организации и приложения с использованием [QCoreApplication::setOrganizationName](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html#organizationName-prop)() and [QCoreApplication::setApplicationName](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html#applicationName-prop)(), а затем использовать конструктор по умолчанию данного класса. *В данной части есть пример кода.*

Данный класс сохраняет настройки в виде пары значений типа QString, в котором сохраняется имя настройки, и QVariant, в котором сохраняется значение настройки. Есть функция для установки соответствующих настроек. Можно возвращать настройки, проверять, есть ли данный ключ в настройках и т.д. Так как QVariant является классом модуля ядра, то у него нет функций приведения к типам модуля графического интерфейса пользователя. *В данной части показано, как в этом случае следует поступать, чтобы возвращать, например, цвет из настроек.* Также при помощи данного класса можно сохранить частные типы данных, которые зарегистрированы при помощи функций [qRegisterMetaType](http://qt-project.org/doc/qt-5.1/qtcore/qmetatype.html#qRegisterMetaType)() and [qRegisterMetaTypeStreamOperators](http://qt-project.org/doc/qt-5.1/qtcore/qmetatype.html#qRegisterMetaTypeStreamOperators)(). Придерживайтесь при написании имён ключей следующих правил:

1. Всегда обращайтесь к одному и тому же ключу с использованием букв одинакового регистра, так как на некоторых платформах это важно.
2. Избегайте ключей с одинаковыми именами.
3. Не используйте слешы в названиях ключей. Некоторые из них обозначают директорию на некоторых платформах, а слеш «/» используется для определения подключей.

Можно создавать иерархические ключи (есть пример кода). *Также есть пример работы с такими вложенными ключами.* Данный класс поддерживает концепцию группы ключей и массива ключей. *Также в данной части указан замечательный механизм, согласно которому qt ищет файлы настроек в разных директориях. Можно регулировать, откуда ключи могут быть считаны. Это очень удобно, как говорится в данной части. Есть пример кода. Следует разобраться в данной формуле более подробно. Более того данный механизм удобен, так как внутри приложения не надо обращаться к конкретным платформенно специфическим файлам.*

Часто данный класс используется для сохранения состояния приложения с графическим интерфейсом пользователя. *Есть пример хода для сохранения геометрических параметров главного окна приложения. Чтение установок можно осуществить в конструкторе класса главного окна.*

Данный класс является реентерабельным. Можно использовать разные объекты данного класса в различных потоках. Это гарантируется, даже если настройки ссылаются на один и тот же файл. Если один объект настроек изменит настройку, то это будет сразу же видно всем остальным объектам настроек. Также к одному файлу настроек могут обращаться несколько процессов. В данном случае используется консультативная блокировка файлов и алгоритм умного слияния, чтобы гарантировать интегрированность данных. Также есть функция, которая импортирует изменения, совершённые другими процессами.

*В последней части описаны для каждой платформы локации, где сохраняются настройки для приложения. Также в данной части показано, как осуществить прямой доступ к файлам настроек на различных платформах. Также в данной части указаны некоторые платформенно зависимые ограничения на ключи и их значения.*

### QRegExp

<http://qt-project.org/doc/qt-5.1/qtcore/qregexp.html#details>

класс QRegExp обеспечивает совпадение с шаблоном с использованием регулярных выражений. Регулярное выражение – это образец для совпадения подстрок в тексте. Регулярные выражения могут проверять, удовлетворяет ли подстрока некоторым критериям; обеспечивает более мощный образец для совпадения, нежели простое совпадение (чтобы были такие-то слова, но не было бы таких-то слов); также может заменить все подстроки в тексте некоторыми другими подстроками; также данный класс может быть использован для нахождения того, где строка должна быть разделена по частям.

Данный класс моделируют на языке регулярных выражений Perl. Он полностью поддерживает юникод. Также можно выбирать некоторые другие образцы синтаксиса регулярных выражений. *Есть ссылка на хорошую книгу по регулярным выражениям.*

Регулярные выражения строятся из выражений, кванторов и утверждений. Простейшим выражением является символ; также выражение может содержать несколько символов, которые заключаются в квадратные скобки. Также можно использовать укороченную запись выражений. *В данной части есть примеры.* Квантор определяет число появлений выражения, которое должно совпасть. Например **x{1,5}** обозначает совпадение последовательности символов **x**, которая содержит по крайней мере один данный символ, но не более пяти. Также в данной части написано, что невозможно написать регулярное выражение, которое бы помогло распознать вложенные теги, если заранее неизвестна степень вложенности. Если вначале регулярного выражения находится символ ^, то это означает, что выражение должно находится вначале строки; а если в конце регулярного выражения находится символ $, то это означает, что выражение должно находится в конце строки. Если же выражение сопровождают оба данных символа, то выражение целиком должно совпадать со строкой. В регулярных выражениях есть некоторые стандартные символы, значение которых необходимо знать. Знак | обозначает ИЛИ. Круглые скобки выделяют компонент регулярного выражения. \b обозначает границы слова. Граница слова – это любой несловесный символ, например, пробел, новая строка и т.д. *Далее есть таблица символов и аббревиатур для наборов символов.* С++ компилятор преобразует слешы в строках. Чтобы включить \ в регулярное выражение, введите его дважды, то есть \\. Чтобы было совпадение с самим слешом, то его следует вводить четыре раза подряд. В квадратных скобках ^ и – имеют особенное предназначение: первый обозначает любой символ за исключением тех, которые там указаны, а второй, - диапазон символов. Круглые скобки позволяют группировать элементы вместе. Внутри них можно использовать различные логические символы. \b – граница слова. В регулярных выражениях можно ссылаться на круглые скобки (смотри пример). Если вы желаете использовать круглые скобки не для захвата, но для группирования и только, то используйте синтаксис ‘(?:’ … ‘)’. В данном случае мы будем знать о совпадении, но не о том, что конкретно было найдено. Оба вида круглых скобок могут быть вложенными.

Утверждения представляют из себя некоторые утверждения о тексте, но они не являются символами для нахождения совпадений. *Также есть таблица для представления утверждений.* Большинство командных оболочек, таких как cmd.exe, поддерживают подстановку файла, возможность идентифицировать группу файлов при помощи использования шаблона. Wildcard является другим шаблоном, но не шаблоном регулярных выражений, который удобно использовать для поиска файлов. *Есть пример кода и таблица его специальных символов.*

*Также отдельная часть посвящена некоторым различиям данного класса и языка Perl. Также в данной части есть ссылка на книгу по данной теме, а также есть ещё пару замечаний относительно использования некоторых символов в С++. (не нашёл эти замечания при повторном просмотре). Также в конце есть обширная секция примеров кода.* Очень полезной функцией для разделения строк является функция [QString::split](http://qt-project.org/doc/qt-5.1/qtcore/qstring.html#split)(). Данный класс позволяет также проверять регистр.

# МОДУЛЬ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

<http://qt-project.org/doc/qt-5.1/qtgui/qtgui-index.html>

GUI модуль обеспечивает классы

* для интеграции с оконной системой,
* для обработки событий,
* для интеграции OpenGL и OpenGLES,
* для двумерной графики,
* для базовых изображений,
* для шрифтов и текста.

Эти классы используются внутренне технологиями qt интерфейса пользователя и могут также быть использованы напрямую, например, для написания приложений с использованием низкоуровневой OpenGL графики приложения.

*Для разработчиков приложений, пишущих интерфейс пользователя, qt обеспечивает более высокоуровневый программный интерфейс, например Qt Quick, который является более подходящими, чем возможности, найденные в данном модуле.*

Чтобы включить определения классов модуля, используйте следующую директиву: #include <QtGui>

*Наиболее важные классы в данном модуле – это QGuiApplication и QWindow. Qt приложение, которое желает отобразить содержание на экране, будет нуждаться в использовании их. Первый класс содержит Главный цикл обработки событий, где все события из оконной системы и других источников обрабатываются и отправляются. Он также обрабатывает инициализацию и окончание приложения. Второй класс представляет окно в нижележащей оконной системе. Он обеспечивает набор виртуальных функций, чтобы обрабатывать события (QEvent) из оконной системы.*

GUI содержит классы для двумерной графики, изображений, шрифтов и продвинутой типографии. QWindow, созданный вместе с поверхностью типа [QSurface::RasterSurface](http://qt-project.org/doc/qt-5.1/qtgui/qsurface.html#SurfaceType-enum) может быть использован в комбинации с [QBackingStore](http://qt-project.org/doc/qt-5.1/qtgui/qbackingstore.html) and [QPainter](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html), которые являются высоко оптимизированным программным интерфейсом двумерной векторной графики в qt. [QPainter](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html) поддерживает рисование линий, прямоугольников, векторных путей, изображений и текста.

Qt может загружать и сохранять изображения с использованием QImage и QPixmap классов. По умолчанию, qt поддерживает большинство общих форматов изображений JPEG и PNG. Пользователи могут добавить поддержку других форматов через [QImageIOPlugin](http://qt-project.org/doc/qt-5.1/qtgui/qimageioplugin.html) класс. Типография в qt совершается при помощи класса [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html), который использует [QPainter](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html) вместе с классами шрифтов qt.

[QWindow](http://qt-project.org/doc/qt-5.1/qtgui/qwindow.html) поддерживает рисование при помощи настольного OpenGL, OpenGL ES 1.1 b OpenGL ES 2.0 в зависимости от того, что поддерживает платформа. Рисование при помощи данной технологии становится возможным при установке типа поверхности QWindow к [QSurface::OpenGLSurface](http://qt-project.org/doc/qt-5.1/qtgui/qsurface.html#SurfaceType-enum), выбирая атрибуты формата при помощи [QSurfaceFormat](http://qt-project.org/doc/qt-5.1/qtgui/qsurfaceformat.html) и затем создавая контекст [QOpenGLContext](http://qt-project.org/doc/qt-5.1/qtgui/qopenglcontext.html), чтобы управлять нативным контекстом OpenGL. Вдобавок qt имеет класс [QOpenGLPaintDevice](http://qt-project.org/doc/qt-5.1/qtgui/qopenglpaintdevice.html), который даёт возможность использовать OpenGL ускоренное рисование QPainter, также как и удобные классы, которые упрощают написание кода OpenGL и уменьшают сложность обработки расширений и отличий между разными видами OpenGL.

Также данный модуль содержит некоторые математические классы, которые представляют наиболее общие математические операции, связанные с 3D графикой.

До qt5 GUI модуль был монолитным контейнером для всех вещей, связанных с графическим интерфейсом пользователя в qt, и включал набор виджетов qt, элементов изображений, каркас графических приложений и также печать. Начиная с qt5, эти классы были удалены в модуль Qt Widgets. Печать была перемещена в модуль Qt Print Support. Пожалуйста, заметьте, что данные модули могут быть исключены из qt установки. Сейчас GUI содержит только малый набор помощников, которые, в общем, полезны для всех графических приложений.

## СИСТЕМА РИСОВАНИЯ

<http://qt-project.org/doc/qt-5.1/qtgui/paintsystem.html>

система рисования qt предоставляет возможность рисования на экране и на печати с использованием одного и того же интерфейса и основана на классах [QPainter](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html), [QPaintDevice](http://qt-project.org/doc/qt-5.1/qtgui/qpaintdevice.html), и [QPaintEngine](http://qt-project.org/doc/qt-5.1/qtgui/qpaintengine.html). [QPainter](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html) используется для выполнений операций рисования, [QPaintDevice](http://qt-project.org/doc/qt-5.1/qtgui/qpaintdevice.html) является абстракцией двумерных пространств, которые могут быть нарисованы с использованием класса [QPainter](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html). [QPaintEngine](http://qt-project.org/doc/qt-5.1/qtgui/qpaintengine.html) обеспечивает интерфейс, который рисовальщик использует, чтобы рисовать на разных типах устройств. Этот класс используется внутренне остальными классами и спрятан от программистов приложений, если только они не создают свой собственный тип устройства. Главное преимущество этого подхода заключается в том, что всё рисование следует по одному и тому же пути, делая лёгким добавление поддержки новых особенностей и обеспечения реализации по умолчанию для неподдерживаемых особенностей.

### ЧТЕНИЕ И ЗАПИСЬ ФАЙЛОВ ИЗОБРАЖЕНИЙ

<http://qt-project.org/doc/qt-5.1/qtgui/paintsystem-images.html>

наиболее общий способ чтения изображений – это через классы QImage и QPixmap. Вдобавок, qt обеспечивает класс QImageReader, который даёт больший контроль над процессом. В зависимости от ниже лежащей поддержки формата изображений, функции, обеспечиваемые классом, могут сохранять память и ускорять загрузку изображений. Также qt обеспечивает класс QImageWriter, который поддерживает настройку особенных опций формата, таких как гамма уровень, сжатие и качество, перед сохранением изображения. Классы QImageWriter и QImageReader полагаются на QImageIOHandler класс, который является общим интерфейсом для ввода/вывода изображений в qt. Объекты данного класса используются классом QImageReader и QImageWriter для добавления поддержки для разных форматов изображений в qt. Класс QMovie является удобным классом для отображения анимации, используя внутренне класс QImageReader.

Qt поддерживает несколько форматов файлов по умолчанию, и вдобавок несколько новых форматов могут быть добавлены при помощи плагинов. Механизм плагинов в qt также может быть использован для написания частных форматов изображений. Это делается при помощи создания специализированного класса QImageIOHandler и создания QImageIOPlugin объекта, который является фабрикой для создания объектов QImageIOHandler. Когда плагин устанавливается, QImageReader и QImageWriter автоматически загрузят плагин и начнут использование его.

### СИСТЕМА КООРДИНАТ

<http://qt-project.org/doc/qt-5.1/qtgui/coordsys.html>

система координат контролируется классом QPainter. Система координат устройства рисования по умолчанию имеет своё начало в верхнем левом углу. Значения абсциссы увеличиваются направо и значения ординаты увеличиваются вниз. Единицей по умолчанию является пиксел на основанном на пикселах устройстве и точка на принтерах.

Отображение логических координат QPainter на физические координаты QPainDevice обрабатывается матрицей преобразований класса QPainter, окном просмотра и «окном».

Размер графических примитивов всегда соответствует его математической модели, игнорируя ширину пера, которым они рисуются. Во время рисования рисование пикселей контролируется при помощи QPainter::Antialiasing подсказкой рисования. Его значение указывает, что движку следует сглаживать примитивы, если возможно, т.е. сглаживать грани с использованием разных интенсивностей цветов.

При рисовании пером чётного числа пикселов они будут нарисованы симметрично по отношению к математически определённой точке, в то время как рисование пером нечётного количества пикселей, оставшийся пиксел будет нарисован справа и внизу от математической точки, как в случае одного пиксела. *В данной части есть прекрасные иллюстративные примеры, которые демонстрируют как раз использование данного правила.*

*Заметьте, что по историческим причинам возвращаемое значение функций класса прямоугольника право и низ отличаются от настоящих значений данной величины. Но это касается только класса QRect, но не класса QRectF.*

Если установить в рисовальщике подсказку для рисования анти-сглаживания, пикселы будут рисоваться симметрично с обеих сторон математически определённых точек. *Есть очень замечательная иллюстрация данного обстоятельства.*

По умолчанию класс QPainter работает со связанной с устройством собственной системой координат, но он также имеет полную поддержку аффинных преобразований координат. Можно масштабировать систему координат, а также можно вращать их по часовой стрелке и транслировать с использованием определённых функций. *В данной части есть прекрасная иллюстрация данных преобразований. Для себя следует отметить, что важным обстоятельством является тот факт, что преобразования применяются именно к системе координат.*

Если вы желаете использовать одинаковые преобразования снова и снова, вы можете также использовать класс QTransform, объекты и функции [QPainter::worldTransform](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html#worldTransform)() и [QPainter::setWorldTransform](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html#setWorldTransform)(). Вы также можете в любое время сохранить матрицу преобразований рисовальщика с использованием функции [QPainter::save](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html#save)(), которая сохраняет матрицу во внутренний стек. Функция [QPainter::restore](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html#restore)() возвращает его обратно. Частая необходимость в матрице преобразований возникает, когда приходится повторно использовать одинаковый код рисования на многих устройствах рисования.

*Также авторы документации рекомендуют компилировать и запустить данный пример перед чтением остальных. Там же есть блестящий пример кода использования всех выше указанных функций.*

Вьюпорт представляет физические координаты, определённые на произвольном прямоугольнике. Окно описывает такой же прямоугольник в логических координатах. По умолчанию логическая и физическая система координат совпадают, и являются эквивалентными прямоугольнику устройства рисования. С использованием преобразований типа вьюпорт – окно вы можете сделать систему логических координат соответствующей вашим предпочтениям. Также механизм может быть использован для установления независимости от устройства рисования кода рисования. *Там далее есть хороший поясняющий пример использования данного типа преобразований.*

Вьюпорт, окно и матрица преобразований определяют, как логические координаты рисовальщика отображаются на физические координаты устройства рисования. По умолчанию матрица мировых преобразований является единичной матрицей, а окно и вьюпорт совпадают с настройками устройства рисования.

*Также важно, чтобы соотношения сторон прямоугольника для рисования совпадали для окна и вьюпорта, так как при проектировании окна на устройство совмещаются их вершины. Поэтому во избежание деформаций следует сохранять для обоих соотношения сторон.*

### УСТРОЙСТВА РИСОВАНИЯ И БЭКЕНДЫ

<http://qt-project.org/doc/qt-5.1/qtgui/paintsystem-devices.html>

[QPaintDevice](http://qt-project.org/doc/qt-5.1/qtgui/qpaintdevice.html) является базовым классом для объектов, на которых можно рисовать, то есть QPainter может рисовать на любом подклассе данного класса. Возможности рисования данного класса среди прочих применяются классами [QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html), [QImage](http://qt-project.org/doc/qt-5.1/qtgui/qimage.html), [QPixmap](http://qt-project.org/doc/qt-5.1/qtgui/qpixmap.html), [QPicture](http://qt-project.org/doc/qt-5.1/qtgui/qpicture.html), QPrinter, and [QOpenGLPaintDevice](http://qt-project.org/doc/qt-5.1/qtgui/qopenglpaintdevice.html).

[QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html) является базовым классом элементов графического интерфейса пользователя в модуле Qt Widgets. Он получает мышь, клавиатуру и другие события из оконной системы и рисует представление себя на экране.

[QImage](http://qt-project.org/doc/qt-5.1/qtgui/qimage.html) обеспечивает независимое от железа представление изображений, которые разрабатываются и оптимизируются для ввода/вывода и для прямого пиксельного доступа и манипуляции.

[QPixmap](http://qt-project.org/doc/qt-5.1/qtgui/qpixmap.html) является классом, который разработан и оптимизирован для показа изображений на экране. Он не позволяет работать с пикселами. Для оптимизации рисования при помощи данного класса qt предоставляет класс QPixmaxCache, который может быть использован для сохранения временных растровых изображений, которые являются дорогими, чтобы генерировать их без использования большей памяти, чем размер кэша.

Также qt предоставляет класс QBitmap, который гарантирует монохроматические растровые изображения и, в основном, используется для создания частных курсоров и кистей, а также для построения объектов регионов.

Как было упомянуто ранее qt предоставляет классы, которые делают лёгким использование OpenGL в приложениях qt. Например, класс [QOpenGLPaintDevice](http://qt-project.org/doc/qt-5.1/qtgui/qopenglpaintdevice.html) предоставляет возможности программного интерфейса данной библиотеки для рисования с помощью QPainter.

[QPicture](http://qt-project.org/doc/qt-5.1/qtgui/qpicture.html) класс является устройством рисования, который записывает и отображает команды QPainter. Картина сериализует команды рисовальщика в устройство ввода/вывода в независимом от платформы формате. Данный класс может быть отображён на разных устройствах (например, pdf, svg, ps, printer, screen), причём выглядеть он будет одинаково.

Поддержка новых устройств может быть реализована при помощи наследования класса QPaintDevice и переопределения виртуальной функции [QPaintDevice::paintEngine](http://qt-project.org/doc/qt-5.1/qtgui/qpaintdevice.html#paintEngine)(), которая говорит QPainter, какой движок рисования следует использовать для рисования на частном устройстве. Для того чтобы действительно быть способным рисовать, этот движок рисования должен быть частным движком рисования, созданным при помощи наследования от класса QPaintEngine.

### РИСОВАНИЕ И ЗАПОЛНЕНИЕ

<http://qt-project.org/doc/qt-5.1/qtgui/paintsystem-drawing.html>

QPainter обеспечивает высоко оптимизированные функции для производства большинства операций рисования для программ с графическим интерфейсом пользователя. Он может рисовать что угодно от простых графических примитивов до сложных геометрических форм. В qt векторные пути представляются классом [QPainterPath](http://qt-project.org/doc/qt-5.1/qtgui/qpainterpath.html), который обеспечивает контейнер для операций рисования, позволяющих построение и повторное использование графических форм. Путь – это объект, который состоит из линий и кривых. Преимуществом данного класса над простым рисованием является то, что сложные формы нужно создавать только один раз. Они могут быть нарисованы много раз с использованием функции [QPainter::drawPath](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html#drawPath)(). Объект пути может быть использован для заливки, создания контура и усечения. Для генерирования заливающихся контуров для данного пути используйте класс [QPainterPathStroker](http://qt-project.org/doc/qt-5.1/qtgui/qpainterpathstroker.html).

При рисовании рисование пикселов контролируется при помощи [QPainter::Antialiasing](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html#RenderHint-enum). [QPainter::RenderHint](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html#RenderHint-enum) перечисление используется для определения флагов для [QPainter](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html), которые могут или не могут быть восприняты данным движком.

### Классы рисования

классы QLine, QLineF обеспечивают двумерный вектор.

класс QMargins определяет четыре поля прямоугольника. Этот класс описывает границы, которые окружают прямоугольник.

классы QPoint, QPointF определяют точку на плоскости.

классы QRect, QRectF определяют прямоугольник на плоскости.

классы QSize, QSizeF определяют размер двумерных объектов.

класс QSvgGenerator обеспечивает устройство рисования, которое используется для создания SVG рисования (Scalable Vector Graphics).

класс QSvgRenderer используется для рисования содержания файлов SVG на устройства рисования.

класс QSvgWidget обеспечивает виджет, который используется, чтобы отображать содержание SVG файлов.

класс QStylePainter является удобным классом для рисования элементов QStyle внутри виджета.

класс QColormap отображает независимые от устройства QColors на аппаратно-зависимые пиксели.

класс QIcon обеспечивает масштабируемые иконки в разных модах и состояниях.

класс QIconEngine обеспечивает абстрактный базовый касс для рисовальщиков [QIcon](http://qt-project.org/doc/qt-5.1/qtgui/qicon.html).

класс QImage обеспечивает независимое от железа представление, которое обеспечивает прямой доступ к пикселам и может быть использовано как устройство для рисования.

класс QImageReader обеспечивает независимый от формата интерфейс для чтения изображений из файлов и других устройств.

класс QImageWriter обеспечивает независимый от формата интерфейс для записи изображений в файлы или на другие устройства. В данном классе можно рисовать при помощи QPainter.

класс QGenericMatrix является шаблоном для представления матрицы преобразований N\*M с N столбцами и М строками соответственно.

класс QVector2D представляет вектор или узел в двумерном пространстве.

класс QBrush определяет образец заполнения форм, нарисованных при помощи QPainter.

класс QGradient (QLinearGradient, QRadialGradient, QConicalGradient) используется совместно с QBrush, чтобы определить стиль градиента заливки.

класс QColor обеспечивает цвета, основанные на RGB, HSV и CMYK. Данный класс является независимым от платформы и устройства. Класс QColorMap отображает цвет на железо.

класс QPagedPaintDevice является представлением устройства печати, которое поддерживает множественные страницы.

класс QPaintEngine обеспечивает абстрактное определение того, как QPainter рисует на заданном устройстве на заданной платформе. Для использования класса QPainter для рисования на других подсистемах вы должны определить подкласс данного класса и переопределить все его виртуальные функции. Затем данное определение становится доступным при помощи создания подкласса класса QPaintDevice и переопределения его виртуальной функции paintEngine().

класс QPainter выполняет рисование на виджетах и других устройствах рисования. Данный класс обеспечивает высоко оптимизированные функции, которые выполняют большинство операций рисования, которые требуют программы с графическим интерфейсом пользователя. Данный класс рисует в естественной системе координат, но он также может делать отображение и мировые преобразования. Данный класс может работать на любом объекте, который наследует класс QPaintDevice. Общее использование данного класса происходит внутри события рисования виджета: создайте и настройте рисовальщик. Затем рисуйте. Помните, что рисовальщик следует удалить после рисования. Для получения оптимального результата рисования с использованием данного класса вам следует использовать класс QImage как устройство рисования. Использование данного класса гарантирует, что результат имеет одинаковое пиксельное представление на всех платформах. Когда рисуете с помощью данного класса, то обычно вы определяете точки с использованием логических координат, которые затем преобразуются в физические координаты устройства рисования.

Если вы желаете рисовать сложные формы, особенно если вам необходимо делать это периодически, рассмотрите создание QPainterPath и рисование с использование функции drawPath().

класс QPainterPathStroker используется для генерации заполняемых контуров для данного пути рисовальщика.

класс QPdfWriter является классом для генерации PDF, которые могут быть использованы на устройстве рисования.

класс QPen определяет, как QPainter следует рисовать линии и контуры форм.

классы QPolygon, QPolygonF обеспечивает вектор точек.

класс QRegion определяет регион усечения для рисовальщика. Данный класс используется вместе с [QPainter::setClipRegion](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html#setClipRegion)(), чтобы ограничить область рисования той областью, которую следует зарисовать. Данный класс не используется для построения форм.

класс QTransform определяет двумерные преобразования системы координат. Данный класс рекомендуется для осуществления трансформаций в qt.

класс QFont определяет шрифт, используемый при рисовании текста. При создании шрифта вы определяете различные атрибуты, которые вы хотите, чтобы имел шрифт. Qt будет использовать ближайший шрифт к заданному вами. Атрибуты шрифта обычно можно получить из объекта QFontInfo. QFontMetrics используется для получения метрики шрифта.

класс QSupportedWritingSystems используется, когда шрифты регистрируются вместе с внутренней базой данных шрифтов Qt.

класс QBackingStore обеспечивает область рисования для QWindow.

класс QClipboard обеспечивает доступ к буферу обмена оконной системы.

класс QRawFont обеспечивает доступ к одиночному физическому экземпляру шрифта.

класс QRasterPaintEngine позволяет ускорить рисование в qt на встроенной Linux.

класс QPixmapCache обеспечивает распространяющийся на приложение кэш для растровых изображений.

класс QPalette содержит группы цветов для каждого состояния виджета. Если вы создаёте новый виджет, то мы строго рекомендуем, чтобы вы использовали цвета в палитре, нежели жёстко закодированные определённые цвета. Мы также строго рекомендуем, чтобы вы использовали палитру по умолчанию текущего стиля ([QGuiApplication::palette](http://qt-project.org/doc/qt-5.1/qtgui/qguiapplication.html#palette)()) и изменяли её, если необходимо. Это делается виджетами qt, когда они рисуются.

класс QPaintEngineState обеспечивает информацию об активном текущем состоянии движка рисования. Единственный случай прямого использования данного класса – это когда применяется ваш собственный движок.

класс QGlyphRun обеспечивает прямой доступ к внутренним Глифам шрифта.

класс QCursor обеспечивает курсор мыши произвольной формы.

класс QStaticText предоставляет оптимизированное рисование текста, когда текст и его макет обновляются редко. Данный класс может представлять только текст, поэтому в него можно добавлять только тэги HTML для задания форматированного текста.

класс QSurface является абстракцией поверхностей для рисования в qt.

класс QSurfaceFormat представляет формат класса [QSurface](http://qt-project.org/doc/qt-5.1/qtgui/qsurface.html).

#### QLine

<http://qt-project.org/doc/qt-5.1/qtcore/qline.html#details>

класс QLine обеспечивает двумерный вектор с использованием целочисленной точности.

Линия также может транслироваться при помощи специальной функции*.*

#### QLineF

<http://qt-project.org/doc/qt-5.1/qtcore/qlinef.html#details>

класс QLineF обеспечивает двумерный вектор с использованием вещественной точности. Можно изменять длину линии, её угол. Есть функции для проверки того, пересекаются ли линии. Можно получить единичный и нормальный вектора, а также выполнить трансляцию линии.

#### QMargins

<http://qt-project.org/doc/qt-5.1/qtcore/qmargins.html#details>

класс QMargins определяет четыре поля прямоугольника. Этот класс описывает границы, которые окружают прямоугольник.

#### QPoint

<http://qt-project.org/doc/qt-5.1/qtcore/qpoint.html#details>

класс QPoint определяет точку на плоскости с использованием целочисленной точности. Класс может также использоваться как вектор.

#### QPointF

<http://qt-project.org/doc/qt-5.1/qtcore/qpointf.html>

класс QPointF определяет точку на плоскости с использованием вещественной точности. Есть функции для возврата ссылок на координаты для прямой манипуляции ими. Также данный класс может использоваться как вектор.

#### QRect

<http://qt-project.org/doc/qt-5.1/qtcore/qrect.html#details>

класс QRect определяет прямоугольник на плоскости с использованием целочисленной точности. Прямоугольник обычно задаётся при помощи верхней левой вершины и размера (длина и ширина). Есть функции для перемещения прямоугольника. Есть функции для определения того, находится ли данная точка внутри прямоугольника. Также есть функция, которая определяет, пересекаются ли данные прямоугольники. Также есть функции, которые возвращают пресечение прямоугольников и их объединение. *В данной части есть поясняющие рисунки. Также есть секция, которая показывает, как будут прорисовываться границы с различными параметрами для прямоугольника. Также есть секция о том, какие именно координаты возвращают различные функции.* Также важно помнить, что размеры данного прямоугольника ограничиваются максимальным размером целочисленного значения.

#### QRectF

<http://qt-project.org/doc/qt-5.1/qtcore/qrectf.html#details>

класс QRectF определяет прямоугольник на плоскости с использованием вещественной точности. *Описание данного класса полностью соответствует описанию класса целочисленного прямоугольника.*

#### QSize

[*http://qt-project.org/doc/qt-5.1/qtcore/qsize.html#details*](http://qt-project.org/doc/qt-5.1/qtcore/qsize.html#details)

класс QSize определяет размер двумерных объектов с использованием целочисленной точности. Есть различные функции, которые обеспечивают прямые ссылки на ширину и высоту, также есть функции, которые позволяют, сравнивая два размера, ограничивать один из них или расширять. *Данный класс следует проверить экпериментально.*

#### QSizeF

<http://qt-project.org/doc/qt-5.1/qtcore/qsizef.html#details>

класс QSizeF определяет размер двумерного объекта с использованием вещественной точности. Кроме прочих есть функция для изменения местами ширины и высоты. также можно приводить один тип размера к другому. *Всё остальное подобно классу* QSize.

#### QSvgGenerator

<http://qt-project.org/doc/qt-5.1/qtsvg/qsvggenerator.html#details>

класс QSvgGenerator обеспечивает устройство рисования, которое используется для создания SVG рисования (Scalable Vector Graphics). Это, как и QPainter, устройство только для записи, которое генерирует выход в специальном формате. Для записи данного файла для начала следует настроить выход при помощи установки имени файла и свойств устройства вывода. *Есть пример кода.* Также говорится, что лучше применять функции начала и конца, а также отмечается, что в данной документации есть пример с использованием данного класса.

#### QSvgRenderer

<http://qt-project.org/doc/qt-5.1/qtsvg/qsvgrenderer.html#details>

класс QSvgRenderer используется для рисования содержания файлов SVG на устройства рисования. С использованием данного класса векторное изображение может быть нарисовано на любом подклассе QPaintDevice, включая QWidget и QImage, а также QGLWidget. Данный класс поддерживает как базовые особенности векторных изображений, так и позволяет более интерактивные особенности, например, анимацию. Данные могут предоставляться напрямую в формате XML или ненапрямую, с использованием имени файла. Данный класс обеспечивает слот для рисования текущего документа или текущего анимированного документа с использованием данного рисовальщика. Также можно получать информацию о количестве пространства, которое будет занимать изображение. Есть несколько функций для анимации, а также сигнал, который высылается каждый раз, когда рисование документа нуждается в обновлении.

#### QSvgWidget

<http://qt-project.org/doc/qt-5.1/qtsvg/qsvgwidget.html#details>

класс QSvgWidget обеспечивает виджет, который используется, чтобы отображать содержание SVG файлов. Этот класс позволяет изображать такую графику отдельно от стандартных виджетов и используется во многом похожим образом, как и QLabel. Так как данный класс является виджетом, то рисование происходит с использованием свойств экрана. Для большего контроля можно попробовать процесс рисования вместе с классом QSvgRenderer. Файл изображения можно загрузить двумя способами: при помощи имени файла или с использованием QByteArray, содержащем сериализованное XML-представление SVG-файла.

#### QStylePainter

<http://qt-project.org/doc/qt-5.1/qtwidgets/qstylepainter.html#details>

класс QStylePainter является удобным классом для рисования элементов QStyle внутри виджета. Данный класс расширяет QPainter при помощи набора высокоуровневых функций draw(), реализованных над программным интерфейсом QStyle. Если объект QStyle должен быть способен рисовать на любом виджете с использованием любого рисовальщика (так как приложение обычно имеет один данный объект, разделяемый всеми виджетами), то данный класс инициализируется вместе с виджетом, исключая необходимость определять виджет, рисовальщик и стиль для каждого вызова функции. *Есть пример кода. Не совсем я понял преимущества данного класса.*

#### QColormap

<http://qt-project.org/doc/qt-5.1/qtwidgets/qcolormap.html#details>

класс QColormap отображает независимые от устройства QColors на аппаратно-зависимые пиксели. *Следует лучше понять данный класс.*

#### QBitmap

<http://qt-project.org/doc/qt-5.1/qtgui/qbitmap.html#details>

класс QBitmap обеспечивает монохроматическое (Глубиной в 1 бит) растровое изображение. Данный класс используется обычно для создания частных курсоров и кистей, построения объектов регионов и для установки масок для растровых изображений и виджетов. *Также следует понять, что имеется в виду относительно какой-то маски. Возможно, это будет важно для меня.*

#### QIcon

<http://qt-project.org/doc/qt-5.1/qtgui/qicon.html#details>

класс QIcon обеспечивает масштабируемые иконки в разных модах и состояниях. Класс позволяет преобразовать некоторое текущее изображение в другие типы изображений, которые соответствуют некоторым действиям пользователей. Наиболее простой способ использования данного класса – это использование файла [QPixmap](http://qt-project.org/doc/qt-5.1/qtgui/qpixmap.html). *Есть пример кода.* Также можно создавать функцию для частных виджетов, которая рисует иконку для них. *В данной части есть пример кода.*

#### QIconEngine

<http://qt-project.org/doc/qt-5.1/qtgui/qiconengine.html#details>

класс QIconEngine обеспечивает абстрактный базовый касс для рисовальщиков [QIcon](http://qt-project.org/doc/qt-5.1/qtgui/qicon.html). Класс обеспечивает функции рисования для иконки. Каждая иконка имеет соответствующий движок иконки, который несёт ответственность за рисование иконки с заданным размером, модой и состоянием. *В данной части показано, как создавать подклассы данного движка.*

#### QImage

<http://qt-project.org/doc/qt-5.1/qtgui/qimage.html#details>

класс QImage обеспечивает независимое от железа представление, которое обеспечивает прямой доступ к пикселам и может быть использовано как устройство для рисования. Qt предоставляет четыре класса для обработки растровых изображений: QImage, [QPixmap](http://qt-project.org/doc/qt-5.1/qtgui/qpixmap.html), [QBitmap](http://qt-project.org/doc/qt-5.1/qtgui/qbitmap.html) and [QPicture](http://qt-project.org/doc/qt-5.1/qtgui/qpicture.html). Данный класс разработан и оптимизирован под ввод/вывод и для прямого доступа и манипулирования, тогда как [QPixmap](http://qt-project.org/doc/qt-5.1/qtgui/qpixmap.html) разработан и оптимизирован для показа изображений на экране. QBitmap является просто удобным классом, который гарантирует глубину в 1 бит. QPicture – это устройство рисования, которое записывает и воспроизводит команды QPainter.

Данный класс является подклассом класса QImageDevice. Поэтому QPainter может быть использован для прямого рисования на данных изображениях. Также при использовании данного класса рисование может производиться в другом потоке, нежели текущий поток графического интерфейса пользователя. Данный класс содержит набор функций для получения информации об изображениях. Также есть несколько функций, которые обеспечивают трансформацию изображений. Данные объекты могут передаваться в потоковом режиме и сравниваться. *Если вы желаете загрузить объекты данного класса при статическом построении qt, то читайте часть про плагины.*

*Также сказано, что нельзя изображать изображения в каком-то там формате. Далее перечислены различные способы загрузки изображений: из ресурсов, из файлов, из данных. А также перечислены различные полезные функции данного класса.*

Функции для манипуляции над пикселами изображения зависят от его формата.

*Далее очень информативно и замечательно показано, как манипулировать пикселами для различных типов изображений. Далее описаны различные форматы изображений, а также* сказано, что есть функции для определения различных форматов. *Далее перечислены функции для трансформации изображения, например, для получения зеркального изображения. Также в конце рассмотрен некоторый юридический вопрос по использованию одного из алгоритмов в данном классе. Там есть по нему лицензионное соглашение.*

#### QImageReader

<http://qt-project.org/doc/qt-5.1/qtgui/qimagereader.html#details>

класс QImageReader обеспечивает независимый от формата интерфейс для чтения изображений из файлов и других устройств. Наиболее общим способом чтения изображений является использование классов QImage и QPixmap. Данный класс определён для большего контроля при чтении изображений. Например, вы можете считать изображение в определённый размер, а также установить урезанный прямоугольник для эффективной загрузки только частей изображений. *Далее сказано, как считывать изображение, а также как получать информацию об ошибке, если она случилась при считывании.* Данный класс обнаруживает формат файла по умолчанию, а вы можете разрешить или запретить данную особенность.

#### QImageWriter

<http://qt-project.org/doc/qt-5.1/qtgui/qimagewriter.html#details>

класс QImageWriter обеспечивает независимый от формата интерфейс для записи изображений в файлы или на другие устройства. *В остальном описание данного класса подобно на описание класса QImageReader.*

#### QPixmap

<http://qt-project.org/doc/qt-5.1/qtgui/qpixmap.html#details>

класс QPixmap является неэкранным представлением изображения, которое может быть использовано как устройство рисования. Данный класс может быть легко изображён на экране с использованием метки или одного из классов кнопок. Метка имеет свойство растрового изображения, а кнопка свойство иконки. Данный класс может передаваться по значению, а также выводится в потоковом режиме, так как он использует неявное разделение. Пиксели в данном классе являются внутренними, и к ним нет доступа. В данном классе можно рисовать при помощи QPainter. Есть функции для конвертации из одного типа изображения в qt к другому типу изображения в qt. *Далее показан способ чтения и записи растрового изображения. Перечислены функции для получения информации о растровом изображении, о функции преобразования между различными типами данных в qt, описывающими изображения, а также функции преобразования самих изображений.*

#### QGenericMatrix

<http://qt-project.org/doc/qt-5.1/qtgui/qgenericmatrix.html#details>

класс QGenericMatrix является шаблоном для представления матрицы преобразований N\*M с N столбцами и М строками соответственно.

#### QVector2D

<http://qt-project.org/doc/qt-5.1/qtgui/qvector2d.html#details>

класс QVector2D представляет вектор или узел в двумерном пространстве.

#### QBrush

<http://qt-project.org/doc/qt-5.1/qtgui/qbrush.html#details>

класс QBrush определяет образец заполнения форм, нарисованных при помощи QPainter. Кисть имеет стиль, цвет, градиент или текстуру. *Далее описано с примерами кода и иллюстрациями, как задавать и использовать данные свойства. Также далее ведётся речь о том, как задавать прозрачность кисти, и какая кисть считается непрозрачной.*

#### QGradient

<http://qt-project.org/doc/qt-5.1/qtgui/qgradient.html#details>

класс QGradient используется совместно с QBrush, чтобы определить стиль градиента заливки. В настоящее время qt поддерживает три типа заливки градиента: линейный, простой радиальный, расширенный радиальный и конический. *Далее показано с примерами кода, как создавать градиентную заливку, совместно с точками опорными точками.* Также можно повторять или отражать градиент за пределами его области при помощи специальных функций. Данные функции работают только для линейного и сферического градиентов, так как конический градиент является сам по себе замкнутым. Также можно установить координатную моду градиента: по отношению к координатам устройства, по отношению к координатам ограничивающего объект ящика.

#### QLinearGradient

<http://qt-project.org/doc/qt-5.1/qtgui/qlineargradient.html#details>

класс QLinearGradient используется для определения линейного градиента кисти. *Далее в данной части описывается его использование.*

#### QRadialGradient

<http://qt-project.org/doc/qt-5.1/qtgui/qradialgradient.html#details>

класс QRadialGradient используется в комбинации с кистью для определения радиального градиента кисти. *Далее в данной части описывается использование данного класса.*

#### QConicalGradient

<http://qt-project.org/doc/qt-5.1/qtgui/qconicalgradient.html#details>

класс QConicalGradient используется для определения конического градиента кисти. *Далее в данной части описывается использование данного класса.*

#### QColor

<http://qt-project.org/doc/qt-5.1/qtgui/qcolor.html#details>

класс QColor обеспечивает цвета, основанные на RGB, HSV и CMYK. Цвет обычно определяется на основании компонентов RGB, но также возможно его определить на основании (оттенок, насыщенность и яркость - HSV) и (голубой, пурпурный, жёлтый и чёрный – CMYK). Также цвет может быть определён с использованием имени цвета (SVG 1.0). Также можно делать цвета темнее и светлее. Можно получать компоненты цвета отдельно. *Также рассказывается про тип QRgb и функции qRed(), qGray() и другие.* Данный класс является независимым от платформы и устройства. Класс QColorMap отображает цвет на железо. Данный класс предоставляет вещественную точность и для многих функций имеет дубликаты, возвращающие значения с вещественной точностью.??? Данный класс поддерживает рисование с использованием альфа-смешивания. Альфа канал цвета определяет эффект прозрачности. *Есть пример кода. Также в специальной таблице помещены доступные предопределённые цвета.* Также есть дополнительные цвета [Qt::color0](http://qt-project.org/doc/qt-5.1/qtcore/qt.html#GlobalColor-enum), [Qt::color1](http://qt-project.org/doc/qt-5.1/qtcore/qt.html#GlobalColor-enum) and [Qt::transparent](http://qt-project.org/doc/qt-5.1/qtcore/qt.html#GlobalColor-enum), предназначенные для специальных целей. *Их описание дано в данной части.*

RGB модель является железо-ориентированной. Её представление близко к тому, что показывают большинство мониторов. В отличие от этого, HSV представляет цвет способом, более подходящим для человеческого восприятия цвета. *Далее в данной части приводится прекрасное описание HSV модели цвета. А также говорится про то, как qt реагирует на выход за пределы диапазона изменения компонентов данной модели.*

HSL похожа на HSV. Данная модель имеет такой параметр, как яркость. Яркость проходит от чёрного к цвету и от цвета к белому.

CMYK используется для процесса печати: прессы и некоторых других твёрдых копий. *Также описывается, почему в данной цветовой модели используется именно четыре цвета*

#### QPagedPaintDevice

<http://qt-project.org/doc/qt-5.1/qtgui/qpagedpaintdevice.html#details>

класс QPagedPaintDevice является представлением устройства печати, которое поддерживает множественные страницы. Устройства страничного формата используются для генерации вывода для печати или для форматов, например для PDF. QPdfWriter и QPrinter наследуют данный класс.

#### QPaintDevice

<http://qt-project.org/doc/qt-5.1/qtgui/qpaintdevice.html>

класс QPaintDevice является базовым классом для объектов, которые могут быть нарисованы с помощью QPainter. Устройство рисования – это абстракция двумерного пространства, которое может быть нарисовано с использованием класса QPainter. Его система координат по умолчанию имеет нуль в левом верхнем углу. Возможности рисования данного класса в настоящий момент использованы классами [QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html), [QImage](http://qt-project.org/doc/qt-5.1/qtgui/qimage.html), [QPixmap](http://qt-project.org/doc/qt-5.1/qtgui/qpixmap.html), [QGLPixelBuffer](http://qt-project.org/doc/qt-5.1/qtopengl/qglpixelbuffer.html), [QPicture](http://qt-project.org/doc/qt-5.1/qtgui/qpicture.html), and QPrinter. Чтобы реализовать поддержку новой подсистемы, вам следует наследовать данный класс и переопределить виртуальную функцию paintEngine(), которая говорит QPainter, какой движок рисования должен быть использован на устройстве, т.е. производный от [QPaintEngine](http://qt-project.org/doc/qt-5.1/qtgui/qpaintengine.html) с некоторыми переопределёнными виртуальными функциями.

Qt требует, чтобы существовал любой объект QGuiApplication перед созданием любого устройства рисования. Устройства рисования имеют доступ к системе ресурсов окон. Эти ресурсы не инициализируются перед тем, как объект приложения создаётся. Также данный класс содержит множество функций для извлечения информации об устройстве.

#### QPaintEngine

<http://qt-project.org/doc/qt-5.1/qtgui/qpaintengine.html#details>

класс QPaintEngine обеспечивает абстрактное определение того, как QPainter рисует на заданном устройстве на заданной платформе. Qt по умолчанию предоставляет движок для растрового рисования и движок для OpenGL (доступен через QGLWidget), а также движок для печати. Для использования класса QPainter для рисования на других подсистемах вы должны определить подкласс данного класса и переопределить все его виртуальные функции. Затем данное определение становится доступным при помощи создания подкласса класса QPaintDevice и переопределения его виртуальной функции paintEngine(). Объект данного класса создаётся и подчиняется объекту класса QPaintDevice, который его создал.

#### QPainter

<http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html#details>

класс QPainter выполняет рисование на виджетах и других устройствах рисования. Данный класс обеспечивает высоко оптимизированные функции, которые выполняют большинство операций рисования, которые требуют программы с графическим интерфейсом пользователя. Данный класс рисует в естественной системе координат, но он также может делать отображение и мировые преобразования. Данный класс может работать на любом объекте, который наследует класс QPaintDevice. Общее использование данного класса происходит внутри события рисования виджета: создайте и настройте рисовальщик. Затем рисуйте. Помните, что рисовальщик следует удалить после рисования.

Центр функциональности данного класса – это рисование, но данный класс также обеспечивает несколько функций, которые позволяют вам настроить настройки рисовальщика, его качество рисования и другие настройки, которые делают возможным усечение. Также можно контролировать то, как разные формы сливаются вместе при помощи задания моды композиции рисовальщика.

Для начала рисования нужно активировать рисовальщик, а после завершения рисования следует деактивировать рисовальщик. Если устройством рисования является виджет, то данный класс может быть вызван только из функции paintEvent() или из функции, которая вызывается из данной функции. *Далее перечислены настройки, которые вы можете настраивать, чтобы сделать ваш рисовальщик таким, какой вы предпочитаете.* Также можно заметить, что некоторые из настроек данного класса совпадают с настройками некоторых типов устройств рисования. Следует копировать настройки из устройств в данный класс. В любое время вы можете сохранить состояние рисовальщика при помощи вызова функции save(), которая сохраняет все доступные настройки во внутреннем стеке. Также данный класс обеспечивает многочисленные функции рисования различных фигур. Все данные функции имеют как целочисленные, так и вещественные версии.

Если вы желаете рисовать сложные формы, особенно если вам необходимо делать это периодически, рассмотрите создание QPainterPath и рисование с использование функции drawPath(). Также есть функции для заливки сложных фигур и заливки вокруг них. *В данной части есть множество ссылок на полезные примеры.* Есть функции для рисования растровых изображений, текста и картин.

Для получения оптимального результата рисования с использованием данного класса вам следует использовать класс QImage как устройство рисования. Использование данного класса гарантирует, что результат имеет одинаковое пиксельное представление на всех платформах.

Также можно контролировать качество изображения при помощи различных функций качества изображения, точности рисования. Данный класс обеспечивает хорошую поддержку трансформации системы координат. Обычно данный класс работает на системе координат устройства (обычно, в пикселах). Наиболее часто используемые трансформации – это масштабирование, вращение, трансляция и сдвиг. *Также значительная часть посвящена обеспечению различных преобразований, но я воспользуюсь данной частью затем более подробно.*

Когда рисуете с помощью данного класса, то обычно вы определяете точки с использованием логических координат, которые затем преобразуются в физические координаты устройства рисования. Отображение логических координат в физические координаты обрабатывается специальными функциями.

Данный класс может усекать любую операцию рисования в прямоугольник, регион или векторный путь. *В общем я не совсем чувствую данную тему, а потому в дальнейшем следует с ней поупражняться.* Данный класс обеспечивает моду композиции, которая определяет правила Портера-Дафа для композиции цифровых изображений; она описывает модель комбинирования пикселей на изображении источника с пикселами на другом изображении (назначении).

Для данного класса есть некоторые ограничения. Если вы используете координаты с основанным на растре движком рисования, то важно заметить, что пока координаты более +(-)215 can be used, любое другое рисование, выполненное с координатами вне этого диапазона не гарантируется быть показанным; рисование может быть усечено. Это происходит из-за применения короткого целочисленного типа в определении. *Не понял данного ограничения.*

*В конце говорится про то, что невозможно угадать, какие операции рисования будут эффективно выполняться на устройствах рисования. Поэтому в данной части перечислены особенности каждого из используемых движков, а также перечислены операции, которые гарантированно будут выполняться быстро. По всем другим операциям даётся совет протестировать их перед использованием на месте, чтобы определить, являются ли они критическими для производительности.*

#### QPainterPath

<http://qt-project.org/doc/qt-5.1/qtgui/qpainterpath.html#details>

класс QPainterPath обеспечивает контейнер для рисования, дающий возможность построения и повторного использования графических форм. Это объект, который состоит из набора графических блоков построения, таких как прямоугольники, эллипсы, линии и кривые. Построение блоков может быть соединено в близкие подпути, например как прямоугольник или эллипс. Закрытый путь имеет совпадающие начальную и конечную точки. Или они могут существовать как независимые пути, такие как линии или кривые. Данный объект может быть использован для заполнения, проведения контура или для усечения. Преимуществом данного класса является то, что сложные формы необходимо создавать только однажды, а затем они могут быть нарисованы много раз с использованием вызовов функции drawPath(). Данный класс содержит набор функций, которые могут быть использованы для получения информации о пути и его элементах. Вдобавок возможно реверсировать порядок использования элементов.

*Далее написано, как создавать объект данного класса.* Текущая позиция класса всегда является конечной позицией последнего подпути, который был добавлен. Есть функция для перемещения текущей позиции без добавления компонента. Также есть функции для явного закрытия и создания новых подпутей. Также данный класс позволяет добавлять закрытые подпути к данному пути. Также путь может быть добавлен к текущему пути.

*Далее в данной части показан пример кода, как использовать данный класс.* Также можно выбирать один из двух методов заливки путей. Также данный класс содержит набор функций, которые возвращают информацию о текущем данном пути. *Также есть функции для преобразования путей в прямоугольники, но я не понял, для чего нужна данная функция, и, Главное, для чего она предназначена.*

#### QPainterPathStroker

<http://qt-project.org/doc/qt-5.1/qtgui/qpainterpathstroker.html#details>

класс QPainterPathStroker используется для генерации заполняемых контуров для данного пути рисовальщика. При помощи вызова функции createStroke(), передавая данный QPainterPath ей как аргумент, новый путь, представляющий контур данного пути создаётся. Затем новый созданный путь может быть заполнен, чтобы нарисовать контур оригинального пути.

#### QPdfWriter

<http://qt-project.org/doc/qt-5.1/qtgui/qpdfwriter.html#details>

класс QPdfWriter является классом для генерации PDF, которые могут быть использованы на устройстве рисования. Он генерирует PDF из серии команд рисования с использованием QPainter. Также можно создавать несколько страниц.

#### QPen

<http://qt-project.org/doc/qt-5.1/qtgui/qpen.html#details>

класс QPen определяет, как QPainter следует рисовать линии и контуры форм. Данный класс имеет различные стили: цвет пера, стиль пера, разные типы окончания, разные типы соединений. *Обо всём этом подробнее написано в данном классе. Я этого не читал сейчас, но всё это я уже изучал ранее*.

#### QPolygon

<http://qt-project.org/doc/qt-5.1/qtgui/qpolygon.html#details>

класс QPolygon обеспечивает вектор точек с использованием целочисленной точности представления. Данный класс является неявно разделяемым. В данном классе можно добавлять точки к полигону, а также выполнять его трансляцию и получать ограничивающий прямоугольник. А также можно использовать класс QMatrix для более общих трансформаций полигона. *В данной части есть пример кода, а также полезная ссылка на использование класса матрицы.*

#### QPolygonF

<http://qt-project.org/doc/qt-5.1/qtgui/qpolygonf.html#details>

класс QPolygonF обеспечивает вектор точек с использованием вещественной точности представления. Его можно приводить к целочисленному полигону. В остальном описание данного класса подобно описанию класса целочисленного полигона.

#### QRegion

<http://qt-project.org/doc/qt-5.1/qtgui/qregion.html#details>

класс QRegion определяет регион усечения для рисовальщика. Данный класс используется вместе с [QPainter::setClipRegion](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html#setClipRegion)(), чтобы ограничить область рисования той областью, которую следует зарисовать. Также есть функция [QWidget::repaint](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#repaint)(), которая получает в качестве параметра регион. Данный класс является лучшим инструментом, который минимизирует количество области экрана, которая обновляется при перерисовке. Данный класс не используется для построения форм. Это неявно разделяемый класс.

Регион можно создать из прямоугольника, эллипса, полигона или растра. Сложные регионы могут быть созданы при помощи комбинирования простых регионов с использованием специальных функций. Также можно передвигать регион при помощи специальной операции. *Есть пример использования данного класса. Также есть некоторое прилагаемое к документации данного класса лицензионное соглашение. Его в дальнейшем обязательно следует изучить.*

#### QTransform

<http://qt-project.org/doc/qt-5.1/qtgui/qtransform.html#details>

класс QTransform определяет двумерные преобразования системы координат. Трансформация определяет, как транслировать, масштабировать, сдвигать, вращать или проектировать систему координат и обычно используется, когда рисуется графика. Этот класс отличается от класса QMatrix, который устарел, тем, что он является действительно матрицей три на три, позволяющей перспективные трансформации. Метод toAffine() позволяет привести данный класс к классу матрицы. Если трансформация перспективы была определена на матрице, тогда преобразование вызовет потерю данных. Данный класс рекомендуется для осуществления трансформаций в qt.

*Далее говорится о разных методах создания объектов данного класса.* Данный класс поддерживает отображение графических примитивов: данная точка, линия, полигон, регион или путь могут быть отображены в координатной системе, определённой данной матрицей с использованием функции map(). Есть функции на проверку единичности и сингулярности матрицы. Также есть функции для обращения матриц, для возвращения присоединённой матрицы. Также есть функция для вычисления определителя матрицы. Также класс поддерживает перемножение матриц, сложение и разность. Объекты данного класса могут поточно передаваться и сравниваться.

Класс QPainter имеет функции для некоторого преобразования системы координат без использования данного класса. *Есть пример кода.* Но если вы желаете выполнить более одной операции преобразования, то более эффективным будет построение класса QTransform и вызов функции QPainter::setTransformation(). *Есть пример кода в этом.*

Объект данного класса содержит матрицу 3\*3. Каждый элемент данной матрицы отвечает за некоторое преобразование. *Это всё перечислено в данной части. Есть пример кода, как данный класс преобразует координаты.* Устанавливать и получать элементы матрицы можно при помощи специальных функций. *Есть пример кода использования данного класса.*

#### QFont

<http://qt-project.org/doc/qt-5.1/qtgui/qfont.html#details>

класс QFont определяет шрифт, используемый при рисовании текста. При создании шрифта вы определяете различные атрибуты, которые вы хотите, чтобы имел шрифт. Qt будет использовать ближайший шрифт к заданному вами. Атрибуты шрифта обычно можно получить из объекта QFontInfo. QFontMetrics используется для получения метрики шрифта.

Перед созданием шрифта должен существовать экземпляр приложения QGuiApplication. Если в данном шрифте нет знаков, то qt ищет знаки в ближайших похожих шрифтах. *Есть пример кода создания шрифта. Также в данной части написаны различные функции.*

Каждый шрифт имеет ключ, который может быть использован, например, как ключ в кэше или словаре. Если вы желаете сохранить предпочтения пользователя насчёт шрифтов, вы можете использовать QSettings, записывая информацию о шрифте с помощью функции toString() и считывая её назад с помощью fromString().

Есть как зависимые от устройства, так и независимые от устройства функции для установки размеров шрифта. *Также в данной части описан алгоритм совпадения шрифтов.* Но действительный алгоритм совпадения изменяется от платформы к платформе. *Есть примеры определения атрибутов шрифта, действительно используемого в оконной системе, а также пример получения информации о метрике шрифта.*

*При использовании шрифта следует детально изучить алгоритм поиска подходящих шрифтов подробнее. Также в данной части есть полезные ссылки на интернет источники по шрифтам.*

#### QFontMetrics

<http://qt-project.org/doc/qt-5.1/qtgui/qfontmetrics.html#details>

класс QFontMetrics обеспечивает информацию о метрике шрифта. Данный класс рассчитывает размер символов и строк для данного шрифта. Создать данный объект можно тремя способами: через конструктор, при помощи функций [QWidget::fontMetrics](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#fontMetrics)() и [QPainter::fontMetrics](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html#fontMetrics)(), которые возвращают информацию о шрифте виджета и рисовальщика соответственно. Данный класс обеспечивает набор функций для возврата информации о шрифте, которые являются быстрыми. Но некоторые функции отмечены, как медленные, и рекомендуется их избегать. Для каждого символа вы можете получить его параметры и найти его шрифт. Также символ можно обработать как строку. Также можно получать параметры строки. *Есть пример того, как использовать данный класс. Очень информативный пример, надо сказать.*

#### QFontMetricsF

[http://qt-project.org/doc/qt-5.1/qtgui/QGenericMatrix.html#details](http://qt-project.org/doc/qt-5.1/qtgui/qfontmetricsf.html#details)

класс QFontMetricsF обеспечивает информацию о метрике шрифта. Если шрифт будет далее изменён, то информаиця в данном объекте не изменяется. В остальном данный класс подобен классу QFontMetrics.

#### QSupportedWritingSystems

<http://qt-project.org/doc/qt-5.1/qtgui/qsupportedwritingsystems.html#details>

класс QSupportedWritingSystems используется, когда шрифты регистрируются вместе с внутренней базой данных шрифтов Qt. Данный класс использует для обеспечения лёгкого интерфейса, который указывает на то, какую записывающую систему поддерживает определённый шрифт. *Не очень я понял назначение данного класса. Я закончил изучать систему рисования qt, а теперь возвращаюсь на страницу модули Gui, чтобы прочитать следующий блок.*

#### QPainterPath::Element

[*http://qt-project.org/doc/qt-5.1/qtgui/qpainterpath-element.html#details*](http://qt-project.org/doc/qt-5.1/qtgui/qpainterpath-element.html#details)

класс QPainterPath::Element определяет позицию и тип подпути. Также можно задавать точку, в которую будет присоединяться кривая. *Как это сделать, показано в данной части.*

#### QPixmapCache::Key

<http://qt-project.org/doc/qt-5.1/qtgui/qpixmapcache-key.html#details>

класс QPixmapCache::Key может быть использован для эффективного доступа к классу [QPixmapCache](http://qt-project.org/doc/qt-5.1/qtgui/qpixmapcache.html). Используйте [QPixmapCache::insert](http://qt-project.org/doc/qt-5.1/qtgui/qpixmapcache.html#insert)() для получения экземпляра ключа, сгенерированного кэшем растрового изображения. Вы можете сохранить ключ в вашем собственном объекте для очень эффективного отображения объекта и растрового изображения.

#### QPainter::PixmapFragment

<http://qt-project.org/doc/qt-5.1/qtgui/qpainter-pixmapfragment.html#details>

QPainter::PixmapFragment класс используется в сочетании с [QPainter::drawPixmapFragments](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html#drawPixmapFragments)() функцией, чтобы определить, как растровое изображение или его некоторый прямоугольник рисуется. *Я так понимаю, что данный класс определяет размеры области, куда будет помещаться растровое изображение.*

#### QTextLayout::FormatRange

<http://qt-project.org/doc/qt-5.1/qtgui/qtextlayout-formatrange.html#details>

QTextLayout::FormatRange структура используется для применения экстраформатирования для определённой области содержания макета текста.

#### QBackingStore

<http://qt-project.org/doc/qt-5.1/qtgui/qbackingstore.html#details>

класс QBackingStore обеспечивает область рисования для QWindow. Данный класс даёт возможность использовать класс QPainter для рисования на QWindow с типом растровой поверхности. Другой способ рисования на QWindow - это рисование через OpenGL с использованием класса QOpenGLContext. данный класс содержит буферизованное представление содержания окна и таким образом поддерживает частичное обновление с использованием QP ainter, чтобы обновить только подрегион содержания окна. Данный класс может быть использован приложением, которое хочет использовать рисовальщик без OpenGL ускорения и без экстра переполнения с использованием стеков QWidget или QGraphicsView. *Есть пример использования данного класса.*

#### QClipboard

<http://qt-project.org/doc/qt-5.1/qtgui/qclipboard.html#details>

класс QClipboard обеспечивает доступ к буферу обмена оконной системы. Буфер обмена предоставляет простой механизм копирования и вставки данных между приложениями. Он поддерживает те же типы данных, что и QDrag и использует похожие механизмы. Для продвинутого использования буфера обмена прочитайте секцию Drag and Drop. В приложении может быть единственный объект данного класса. Он доступен с использованием функции QApplication::clipboard(). *Есть пример кода.* Данный класс обеспечивает удобные функции для доступа к общим типам данных, таким как текст в кодировке Unicode или растровые изображения. Также есть очень удобная функция для обеспечения высокой гибкости: вы можете добавлять собственные Mime данные в буфер обмена. *Есть пример кода использования данного класса. Очень наглядный пример.* Windows не поддерживает Глобальное выделение мышью, то есть операционная система только добавляет текст в буфер обмена, когда выполняется явное копирование или вырезание.

#### QCursor

<http://qt-project.org/doc/qt-5.1/qtgui/qcursor.html#details>

класс QCursor обеспечивает курсор мыши произвольной формы. Этот класс используется для создания курсоров мыши, которые связаны с некоторыми виджетами и получают или устанавливают положение курсора мыши. Вы можете также создавать частные формы курсоров с использованием QBitmap, маски или горячей точки. В данной части указаны функции для установки курсора для некоторого виджета, для всех виджетов приложения. Есть функции для установки формы курсора. Также есть функции для установки либо получения позиции курсора. Попытка использования курсора, который был создан до приложения, приведёт к краху.

#### QFontInfo

<http://qt-project.org/doc/qt-5.1/qtgui/qfontinfo.html#details>

класс QFontInfo обеспечивает общую информацию о шрифтах. Данный класс обеспечивает те же функции доступа, что и класс QFont, но данный класс возвращает не те значения, которые были установлены, но значения, которые используются в данный момент для написания.

#### QGlyphRun

<http://qt-project.org/doc/qt-5.1/qtgui/qglyphrun.html#details>

класс QGlyphRun обеспечивает прямой доступ к внутренним Глифам шрифта. Когда qt отображает строку текста в кодировке юникод, qt сначала преобразует юникод точки в список индексов Глифов и список позиций, основанных на одном или более шрифтах. Представление юникод текста и QFont объект будут в данном случае служить как удобные абстракции, которые прячут детали того, что действительно имеет место, когда отображается текст на экране. Например, к тому времени как текст действительно достигнет экрана, он может быть представлен набором шрифтов вдобавок к одному определённому пользователем, например в случае, если первоначальный шрифт не поддерживает все системы письма, содержащиеся в тексте. При некоторых обстоятельствах это может быть полезным, когда разработчик приложения хочет иметь более низкоуровневый контроль над тем, какие Глифы определённого шрифта отображаются на экране. Данный класс обеспечивает интерфейс к прямым данным, необходимым для получения текста на экране. Он содержит набор индексов Глифов, позицию для каждого Глифа и шрифта.

#### QPaintEngineState

<http://qt-project.org/doc/qt-5.1/qtgui/qpaintenginestate.html#details>

класс QPaintEngineState обеспечивает информацию об активном текущем состоянии движка рисования. Данный класс записывает, какие свойства были изменены со времени последнего обновления движка, также как и их текущие значения. Экземпляр данного класса, представляющий текущее состояние активного движка рисования, передаётся как аргумент в функцию [QPaintEngine::updateState](http://qt-project.org/doc/qt-5.1/qtgui/qpaintengine.html#updateState)(). Единственный случай прямого использования данного класса – это когда применяется ваш собственный движок.

#### QPalette

<http://qt-project.org/doc/qt-5.1/qtgui/qpalette.html#details>

класс QPalette содержит группы цветов для каждого состояния виджета. Палитра состоит из трёх групп цветов: активный, недоступный и неактивный. Все виджеты qt содержат палитру и используют её, чтобы рисовать себя. Это делает интерфейс пользователя легко настраиваемым и совместимым. Если вы создаёте новый виджет, то мы строго рекомендуем, чтобы вы использовали цвета в палитре, нежели жёстко закодированные определённые цвета.

В большинстве стилей активные и неактивные виджеты выглядят одинаково. Цвета и кисти могут быть установлены для частных ролей в любой группе цветов палитры при помощи специальных функций. Мы также строго рекомендуем, чтобы вы использовали палитру по умолчанию текущего стиля ([QGuiApplication::palette](http://qt-project.org/doc/qt-5.1/qtgui/qguiapplication.html#palette)()) и изменяли её, если необходимо. Это делается виджетами qt, когда они рисуются. Вы можете также использовать для фона некоторое растровое изображение. Палитру можно скопировать и проверить на совпадение с другой палитрой. Данный класс использует неявное разделение, что делает эффективным его использование в качестве аргумента. Некоторые стили не используют палитру для рисования, например, если они используют нативные движки. Это случай XP, Vista, Mac OS X.

#### QPixmapCache

<http://qt-project.org/doc/qt-5.1/qtgui/qpixmapcache.html#details>

класс QPixmapCache обеспечивает распространяющийся на приложение кэш для растровых изображений. Это инструмент оптимизированного рисования с использованием класса [QPixmap](http://qt-project.org/doc/qt-5.1/qtgui/qpixmap.html). Вы можете использовать данный класс для сохранения временных растров, которые являются дорогими для генерирования, без использования большего места, чем [cacheLimit](http://qt-project.org/doc/qt-5.1/qtgui/qpixmapcache.html#cacheLimit)(). Данный класс создаёт внутренний QCache объект для кэширования растров. Кэш связывает растр с обеспечиваемым пользователем строковым ключом или при помощи [QPixmapCache::Key](http://qt-project.org/doc/qt-5.1/qtgui/qpixmapcache-key.html), который генерирует кэш. Последний вариант работает быстрее. Строковый интерфейс очень полезен для сложных ключей, но интерфейс ключей будет очень эффективен и удобен для отображения объект-растр, - в этом случае вы можете сохранять ключи как члены объекта. Если два растра сохраняются в кэш с одинаковыми ключами, то последний растр вытеснит предыдущий растр. Размер кэша по умолчанию составляет 10 МБ, но его можно увеличивать при помощи специальной функции.

#### QRasterPaintEngine

<http://qt-project.org/doc/qt-5.1/qtgui/qrasterpaintengine.html#details>

класс QRasterPaintEngine позволяет ускорить рисование в qt на встроенной Linux. Это касается, как я понимаю, мобильных приложений, и поэтому пока не представляет для меня интерес.

#### QRawFont

<http://qt-project.org/doc/qt-5.1/qtgui/qrawfont.html#details>

класс QRawFont обеспечивает доступ к одиночному физическому экземпляру шрифта. Это низкоуровневый класс. Для большинства задач подходит класс QFont. При представлении текста в графическом интерфейсе пользователя обычно до конца неизвестно, какой именно физический шрифт используется. Поэтому класс QFont в действительности представляет запрос для шрифтов. Данный класс представляет из себя некоторый единичный физический шрифт. *В данной части перечислены те шрифты, для которых данный класс может оказаться полезным, но подчёркивается, что для других шрифтов он оказывается бесполезным.* Данный класс может использоваться совместно с QGlyphRun. Данный класс рассматривается локальным по отношению к потоку, который его создал. В конце отмечается, что лучшим выбором является класс QStaticText.

#### QStaticText

<http://qt-project.org/doc/qt-5.1/qtgui/qstatictext.html#details>

класс QStaticText предоставляет оптимизированное рисование текста, когда текст и его макет обновляются редко. Данный класс обеспечивает способ кэширования макета данных для блока текста, так что он может быть нарисован более эффективно, чем при использовании [QPainter::drawText](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html#drawText)(), в котором информация о макете пересчитывается при каждом вызове. Класс напрямую обеспечивает оптимизацию для случаев, где текст, его шрифт и преобразования на рисовальщике являются статическими во время нескольких событий рисования. Если текст или его макет изменяются при каждой итерации, то [QPainter::drawText](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html#drawText)() является более эффективным. Изменение любых частей шрифта или преобразований рисовальщика вызовут пересчёт макета текста. Их поэтому следует избегать. Также данным классом поддерживаются только аффинные преобразования.

Данный текст можно помещать в некоторый прямоугольник, либо помещать в некоторую линию. *В данной части показано, как это регулировать.* Можно устанавливать формат отображаемого текста: простой или форматированный. Данный класс может представлять только текст, поэтому в него можно добавлять только тэги HTML для задания форматированного текста.

#### QSurface

<http://qt-project.org/doc/qt-5.1/qtgui/qsurface.html#details>

класс QSurface является абстракцией поверхностей для рисования в qt. Есть функции для получения размера поверхности, а также для получения атрибутов формата.

#### QSurfaceFormat

<http://qt-project.org/doc/qt-5.1/qtgui/qsurfaceformat.html#details>

класс QSurfaceFormat представляет формат класса [QSurface](http://qt-project.org/doc/qt-5.1/qtgui/qsurface.html). Формат включает размер буфера цветов, красный, зелёный и синий; размер альфа буфера; размер буфера Глубины и трафарета; число сэмплов на пиксел для сультисэмплинга. Также данный класс содержит настройки поверхности, такие как профиль OpenGL и версия для рисования.

## НЕКОТОРЫЕ ДРУГИЕ КЛАССЫ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

### класс QDesktopServices обеспечивает методы для доступа к общим службам десктопа. Данная система делает лёгким использование системы справки.

### класс QDoubleValidator обеспечивает выбор диапазона вещественных чисел.

класс QGuiApplication управляет потоком управления приложений с графическим интерфейсом пользователя и Главными настройками. Данный класс содержит Главный цикл обработки событий, где все события от оконной системы и других источников обрабатываются и отображаются. Также он управляет инициализацией и завершением приложения, и обеспечивает управление сессией. Вдобавок, данный класс обрабатывает большинство настроек системы и приложения. Для приложений с графическим интерфейсом пользователя может быть только один экземпляр данного класса. Если графический интерфейс не нужен, то следует использовать объект класса QCoreApllication вместо этого. Для приложений, основанных на QWidget, следует использовать QAplication вместо этого, так как он обеспечивает некоторую функциональность, необходимую для создания экземпляров класса QWidget.

класс QIntValidator обеспечивает валидатор, который гарантирует, что строки содержат действительной целое число в определённом диапазоне.

класс QPlatformSystemTrayIcon абстрагирует системные tray иконки и взаимодействие.

класс QStyleHints содержит специфические для платформы подсказки и настройки. Объект данного класса, полученный из [QGuiApplication](http://qt-project.org/doc/qt-5.1/qtgui/qguiapplication.html), обеспечивает доступ к определённому Глобальному интерфейсу пользователя текущей платформы. Доступ к данным параметрам полезен, когда создаются частные компоненты интерфейса пользователя. При этом они позволяют компонентам проявлять одинаковое поведение и чувствовать себя как другие компоненты.

класс QRegExpValidator используется для проверки совпадения строки с некоторым регулярным выражением.

класс QScreen используется для запроса свойств экрана.

класс QValidator обеспечивает действительность вводимого текста. Валидатор обычно используется вместе с [QLineEdit](http://qt-project.org/doc/qt-5.1/qtwidgets/qlineedit.html), [QSpinBox](http://qt-project.org/doc/qt-5.1/qtwidgets/qspinbox.html) and [QComboBox](http://qt-project.org/doc/qt-5.1/qtwidgets/qcombobox.html).

класс QWindow представляет окно в нижележащей оконной системе. Приложение обычно использует для графического интерфейса [QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html) или [QQuickView](http://qt-project.org/doc/qt-5.1/qtquick/qquickview.html), но не данный класс. Но, всё ещё можно рисовать прямо на окне с использованием [QBackingStore](http://qt-project.org/doc/qt-5.1/qtgui/qbackingstore.html) или [QOpenGLContext](http://qt-project.org/doc/qt-5.1/qtgui/qopenglcontext.html).

### QDesktopServices

<http://qt-project.org/doc/qt-5.1/qtgui/qdesktopservices.html#details>

класс QDesktopServices обеспечивает методы для доступа к общим службам десктопа. Многие среды десктопов обеспечивают службы, которые могут быть использованы приложениями для выполнения общих задач, таких как открытие веб страниц, способом, который как является подходящим, так и принимает во внимание предпочтения приложения. Данный класс содержит функции, которые обеспечивают простой интерфейс к данным службам, которые показывают, выполнились ли они успешно или потерпели крах. Некоторые переменные среды настроены так, чтобы предостеречь пользователей от выполнения файлов, полученных не из локальных url, или запрашивают разрешение пользователя перед выполнением этого. Данная система делает лёгким использование системы справки. Справка может быть обеспечена с использованием меток или текстовых браузеров. *Есть пример кода того, как зарегистрировать справочную информацию внутри приложения.* Для каждой схемы url можно создать только одного обработчика.

### QDoubleValidator

<http://qt-project.org/doc/qt-5.1/qtgui/qdoublevalidator.html#details>

класс QDoubleValidator обеспечивает выбор диапазона вещественных чисел. Данный класс обеспечивает верхний предел, нижний предел и число знаков после запятой. Для интерпретации чисел данный класс использует локацию.

### QGuiApplication

<http://qt-project.org/doc/qt-5.1/qtgui/qguiapplication.html#details>

класс QGuiApplication управляет потоком управления приложений с графическим интерфейсом пользователя и Главными настройками. Данный класс содержит Главный цикл обработки событий, где все события от оконной системы и других источников обрабатываются и отображаются. Также он управляет инициализацией и завершением приложения, и обеспечивает управление сессией. Вдобавок, данный класс обрабатывает большинство настроек системы и приложения. Для приложений с графическим интерфейсом пользователя может быть только один экземпляр данного класса. Если графический интерфейс не нужен, то следует использовать объект класса QCoreApllication вместо этого. Для приложений, основанных на QWidget, следует использовать QAplication вместо этого, так как он обеспечивает некоторую функциональность, необходимую для создания экземпляров класса QWidget.

Есть функция для возврата Глобального указателя приложения. Главными областями ответственности данного класса являются:

* Он инициализирует приложение с настройками десктопа пользователя, а также следит за данными настройками, если пользователь изменяет их Глобально.
* Он выполняет обработку событий, что означает, что он получает события от нижележащей оконной системы и отправляет их к соответствующим виджетам. Вы можете выслать ваши собственные события окнам с использованием [sendEvent](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html#sendEvent)() and [postEvent](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html#postEvent)().
* Он анализирует аргументы командной строки и устанавливает их внутреннее состояние соответственно.
* Он обеспечивает локализацию строк, которые видимы пользователям через [translate](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html#translate)().
* Он обеспечивает некоторые магические объекты, такие как буфер обмена.
* Он знает об окнах приложения.
* Он управляет обработкой курсора мыши приложения.
* Он обеспечивает поддержку для изысканного управления сессиями. Это делает возможным приложениям завершаться изящно, когда пользователь выходит, чтобы отменить процесс выключения, если завершение невозможно и даже сохранить полное состояние приложения для будущей сессии.

*Здесь есть пару ссылок, на которые затем было бы интересно взглянуть.* Так как данный объект делает столько много инициализации, он должен быть создан перед тем, как любые другие объекты, связанные с графическим интерфейсом пользователя будут созданы. Хорошей идеей всегда является создание аргументов командной строки перед любой их интерпретацией или модификацией в приложении.

### QIntValidator

<http://qt-project.org/doc/qt-5.1/qtgui/qintvalidator.html#details>

класс QIntValidator обеспечивает валидатор, который гарантирует, что строки содержат действительной целое число в определённом диапазоне. *Далее есть пример кода использования данного класса. Также есть пример кода использования данного класса для отображения чисел.*

### QKeySequence

<http://qt-project.org/doc/qt-5.1/qtgui/qkeysequence.html#details>

класс QKeySequence инкапсулирует последовательности клавиш, которые используются как горячие клавиши. В наиболее общей форме последовательность клавиш описывает комбинацию клавиш, которые должны быть использованы вместе, чтобы выполнить некоторое действие. *В данной части перечислены три способа создания горячих клавиш. Есть пример кода создания последовательности клавиш каждым из данных способов.*

Предпочтительнее использовать стандартные горячие клавиши, где это возможно. При создании нестандартных последовательностей клавиш, вам следует использовать строки, а не жёстко закодированные целочисленные значения. *Далее в данной части приводится таблица стандартных горячих клавиш. Далее приводится большая часть, которая говорит о том, что в разных локациях есть разные клавиатуры, где, например, для нажатия клавиши + нужен ещё и Shift, а на других платформах для нажатия данной клавиши Shift не нужен. Только использование стандартных горячих клавиш может обеспечить нормальную работу данных горячих клавиш на различных макетах клавиатуры.* Также есть специальная часть, посвящённая созданию последовательностей клавиш в стиле GNU Emacs. Объект данного класса должен создаваться после создания объекта QApplication.

### QMovie

<http://qt-project.org/doc/qt-5.1/qtgui/qmovie.html#details>

класс QMovie является удобным классом для проигрывания фильмов при помощи [QImageReader](http://qt-project.org/doc/qt-5.1/qtgui/qimagereader.html). Данный класс используется для показа простых анимаций без звука. Если вы желаете отобразить видео и медиа содержание, то используйте вместо этого Phonon мультимедиа.

Для создания анимации следует создать объект данного класс и передать ему или имя файла или указатель на QIODevice, содержащий формат изображения анимации. Для запуска фильма в вашем приложении вам следует использовать функцию [QLabel::setMovie](http://qt-project.org/doc/qt-5.1/qtwidgets/qlabel.html#setMovie)(). Также класс обеспечивает ряд функций и сигналов для слежения за процессом проигрывания фильма. *Есть пример кода использования данного класса.* Также можно контролировать скорость фильма. Также можно остановить фильм или поставить его на паузу. Некоторые форматы анимации могут позволить вам установить цвет фона, что вы также можете сделать при помощи специальной функции. Также можно получать номер текущего кадра, полное число кадров в анимации. А также вы можете задать количество циклов, через которые следует запускать анимацию. Также данный класс может быть проинструктирован кэшировать фреймы анимации при помощи вызова специальной функции. *А вот как создавать данные фильмы? Было бы очень полезно узнать. Есть ссылка на пример анимации.*

### QPlatformSystemTrayIcon

<http://qt-project.org/doc/qt-5.1/qtgui/qplatformsystemtrayicon.html#details>

класс QPlatformSystemTrayIcon абстрагирует системные tray иконки и взаимодействие.

### QStyleHints

<http://qt-project.org/doc/qt-5.1/qtgui/qstylehints.html#details>

класс QStyleHints содержит специфические для платформы подсказки и настройки. Объект данного класса, полученный из [QGuiApplication](http://qt-project.org/doc/qt-5.1/qtgui/qguiapplication.html), обеспечивает доступ к определённому Глобальному интерфейсу пользователя текущей платформы. Доступ является только для чтения; обычно платформа обеспечивает пользователя способом настройки данных параметров. Доступ к данным параметрам полезен, когда создаются частные компоненты интерфейса пользователя. При этом они позволяют компонентам проявлять одинаковое поведение и чувствовать себя как другие компоненты.

### QRegExpValidator

<http://qt-project.org/doc/qt-5.1/qtgui/qregexpvalidator.html#details>

класс QRegExpValidator используется для проверки совпадения строки с некоторым регулярным выражением. *Есть примеры кода.* Данный класс позволяет устанавливать валидаторы при помощи регулярных выражений, обеспечиваемых классом QRegExp. *Наиполезнейшая вещь, хочу я сказать!*

### QScreen

<http://qt-project.org/doc/qt-5.1/qtgui/qscreen.html#details>

класс QScreen используется для запроса свойств экрана. Физическое количество точек на дюйм основано на действительном физическом размере пикселов и полезно для предварительного просмотра печати и других случаев, где желательны реальные измерения экрана. Логические точки на дюйм используются для преобразования шрифта и элементов пользовательского интерфейса от размера точек к размеру пикселов и могут отличаться от физических точек на дюйм.

### QDrag

<http://qt-project.org/doc/qt-5.1/qtgui/qdrag.html#details>

класс QDrag обеспечивает поддержку переноса данных по методу перетянул и отпустил, основанном на MIME. Перетаскивание и отпускание является интуитивным способом для пользователей копировать или перемещать данные между приложениями. Поддержка данного механизма в qt обеспечивает средствами данного класса. Данные, которые следует перенести при помощи данного способа, содержатся в объекте QMimeData. Он определяется при поммощи функции [setMimeData](http://qt-project.org/doc/qt-5.1/qtgui/qdrag.html#setMimeData)() способом, *который показан в примере кода в данной части.*

Механизм использования данной операции очень прост. Создаётся объект данного класса, которым должен владеть объект QObject. Затем создаются необходимые данные в объекте класса QMimeData. Данный объект передаётся драгу. Необходимо, чтобы драгом владел некоторый другой объект по той простой причине, чтобы после совершения операции перетаскивания qt удалял ненужный более драг объект.

Пока перетаскивание в процессе, растровое изображение может быть использовано для представления данных. Это изображение будет двигаться вместе с курсором к цели. Это изображение обычно показывает иконку, которая представляет MIME тип данных, которые переносятся, но может быть установлен любой растр. Можно устанавливать, на какую точку растрового изображения показывает горячая точка курсора. *Есть пример кода.*

### QTouchDevice

<http://qt-project.org/doc/qt-5.1/qtgui/qtouchdevice.html#details>

класс QTouchDevice описывает устройство, от которого происходят события прикосновений. *Пока что мне данный класс не нужен.*

### QValidator

<http://qt-project.org/doc/qt-5.1/qtgui/qvalidator.html#details>

класс QValidator обеспечивает действительность вводимого текста. Валидатор обычно используется вместе с [QLineEdit](http://qt-project.org/doc/qt-5.1/qtwidgets/qlineedit.html), [QSpinBox](http://qt-project.org/doc/qt-5.1/qtwidgets/qspinbox.html) and [QComboBox](http://qt-project.org/doc/qt-5.1/qtwidgets/qcombobox.html). Класс является абстрактным. Если встроенных валидаторов недостаточно, то вы можете использовать собственный валидатор, создав подкласс данного класса. *В данной части показано, как создать необходимый подкласс.* Валидатор определяет три состояния. Недействительная строка является чисто недействительной. Промежуточное обозначает, что строка недействительна, но она ещё редактируется и может стать действительной. Допустимое означает, что строка подходит в качестве окончательного результата. *Есть хорошие примеры демонстрации данных состояний.* Валидатор имеет локацию.

### QWindow

<http://qt-project.org/doc/qt-5.1/qtgui/qwindow.html#details>

класс QWindow представляет окно в нижележащей оконной системе. Приложение обычно использует для графического интерфейса [QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html) или [QQuickView](http://qt-project.org/doc/qt-5.1/qtquick/qquickview.html), но не данный класс. Но, всё ещё можно рисовать прямо на окне с использованием [QBackingStore](http://qt-project.org/doc/qt-5.1/qtgui/qbackingstore.html) или [QOpenGLContext](http://qt-project.org/doc/qt-5.1/qtgui/qopenglcontext.html). *Есть названия примеров рисования на окне.* Данный класс характеризуется размерами и Глубиной, а также различными буферами. Можно получать расположение окна на экране. Вращать окно полезно при изменении ориентации анимации. *В данной части есть информации касательно отображения окна на экране. Также рассказано, как предоставить некоторую часть работы по размещению окон на экране нижележащей системе.*

## УПРАВЛЕНИЕ СЕССИЯМИ

<http://qt-project.org/doc/qt-5.1/qtdoc/session.html>

сессия – это группа работающих приложений, каждое из которых имеет определённое частное состояние. Сессия управляется при помощи службы, называемой менеджером сессий. Приложения, участвующие в сессии, называются клиентами сессии. Менеджер сессий выдаёт команды для своих клиентов от имени пользователя. Эти команды могут вызвать то, что клиенты завершатся с не сохранёнными изменениями. Он может позволить сохранить их состояние для будущих сессий, или завершить их изящно. Набор данных операций называется управлением сессией.

Обычно сессии выключаются менеджерами управления сессий, обычно от имени пользователей, когда они хотят выйти из системы. Также система может выполнить автоматическое выключение в чрезвычайной ситуации, например, если мощность скоро закончится. Ясно, что есть существенное отличие между этими типами выключений. Во время первого, пользователь может захотеть взаимодействовать с приложением, определяя точно, какие файлы следует сохранить и какие можно отбросить. Во втором случае времени на взаимодействие нет. Рядом с машиной может даже не быть пользователя в данный момент.

При работе с управлением сессиями на qt начните с переопределения QApplication::commitData(), чтобы дать возможность вашему приложению принять участие в процессе изящного выхода из системы. Если вы работаете с Windows, то это всё, что вы должны обеспечить. В идеале, ваше приложение следует обеспечить диалог выключения, подобный на тот, что *изображён в данной части. Есть ссылка на пример кода создания такого диалога. Также далее приводится информация о том, что ещё необходимо делать для управления сессиями, но отмечается, что это характерно только для системы X11.*

Управление сессиями, поддерживаемое в Windows и Mac OS X достаточно ограниченно из-за недостатка этой функциональности в самой операционной системе. На Unix вы можете или использовать среду десктопа, которая поддерживает стандартное управление сессиями X11R6, или (рекомендуемый метод), использовать реализацию ссылки менеджера сессии, обеспечиваемую консорциумом X. Этот менеджер образцов называется xsm и является частью стандартной установки X11R6. *Далее в данной части описывается способ того, как можно тестировать управление сессиями в системе X11.*

## КЛАССЫ МОДЕЛИ/ПРЕДСТАВЛЕНИЯ

### QStandardItem

<http://qt-project.org/doc/qt-5.1/qtgui/qstandarditem.html#details>

класс QStandardItem обеспечивает элемент для использования вместе с [QStandardItemModel](http://qt-project.org/doc/qt-5.1/qtgui/qstandarditemmodel.html) классом. Элементы обычно содержат текст, иконки или флажки. Каждый элемент может иметь свой цвет фона, для установки которого есть специальная функция. Текстовая метка каждого элемента может быть нарисована с собственным шрифтом и цветом фона. Для каждого элемента есть набор флагов, *о которых указано в данной части.* Каждый элемент может иметь дочернюю двумерную таблицу дочерних элементов. При помощи этого можно строить иерархии элементов. Типичным примером иерархии является дерево, когда дочерним объектом является таблица с одной колонкой. Можно устанавливать размеры дочерних таблиц. При специализации данного класса можно обеспечить частные элементы. Можно определить их новые типы. Есть функция, которая возвращает тип элемента. *Как её пользоваться, написано в данной части.* Также следует переопределить функции data(), setData(), clone(), - последнюю для создания элемента по требованию. Также можно переопределить read() и write(), чтобы управлять тем, как элементы будут представлены в сериализованном виде. Также можно переопределить [operator<](http://qt-project.org/doc/qt-5.1/qtgui/qstandarditem.html#operator-lt)(), если вы желаете контролировать семантикой сравнения элементов между собой. Это полезно, например, для сортировки.

### QStandardItemModel

<http://qt-project.org/doc/qt-5.1/qtgui/qstandarditemmodel.html#details>

класс QStandardItemModel обеспечивает общую модель для сохранения частных данных. Данный класс может быть использован как репозиоторий для стандартных qt типов данных. Это один из классов Model/View и часть а qt model/view. *Информация о данной модели будет в разделе про виджеты.*

Данный класс обеспечивает классический основанный на элементах подход к работе с моделью. Данная модель применяет интерфейс [QAbstractItemModel](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html), что означает, что модель может быть использована для обеспечения данными в любом представлении, которое использует этот интерфейс. Для производительности и гибкости вы можете захотеть специализировать [QAbstractItemModel](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html), чтобы обеспечить поддержку для разных видов репозиториев данных. Например, [QDirModel](http://qt-project.org/doc/qt-5.1/qtwidgets/qdirmodel.html) (УСТАРЕЛ) обеспечивает интерфейс модели для нижележащей файловой системы.

*В данной части написано, как создавать таблицы, деревья, списки, а также как осуществлять поиск элементов в таблице, а также как сортировать элементы в таблице. Есть примеры кода создания таблицы и дерева. Также показано, как при помощи механизма сигналов и слотов получать индексы элементов и элементы, на которые указывает данный индекс. Есть пример кода.*

## КЛАССЫ ТЕКСТОВОГО ДОКУМЕНТА

класс QAbstractTextDocumentLayout::PaintContext является удобным классом для определения параметров, используемых во время рисования макета документа.

класс QAbstractTextDocumentLayout является базовым абстрактным классом, который используется для применения частных макетов для QTextDocuments.

класс QSyntaxHighlighter позволяет вам определить правила синтаксического выделения и также вы можете использовать класс для запроса текущего форматирования документа или данных пользователя. Это базовый класс для реализации синтаксического выделения [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html).

класс QTextBlock обеспечивает контейнер для фрагментов текста в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Данный класс обеспечивает доступ только для чтения к структуре блоков и параграфов текстового документа. он в основном используется, если вы желаете применить ваш собственный формат для визуализации представления [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html), или если вы желаете произвести обход документа и написать его содержание в вашем собственном формате.

класс QTextBlockFormat обеспечивает информацию форматирования для блоков текста в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Документ состоит из набора блоков, представленных объектами [QTextBlock](http://qt-project.org/doc/qt-5.1/qtgui/qtextblock.html). Каждый блок может содержать таблицу, изображение, простой текст и т.д. каждый блок имеет связанный с ним формат, который определяет его характеристики.

класс QTextBlockGroup обеспечивает контейнер для блоков текста внутри [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html).

класс QTextBlockUserData используется для связи частных данных с блоками текста. Класс обеспечивает абстрактный интерфейс для контейнерных классов, которые используются, чтобы связать специфические для приложения данные пользователя с текстовыми блоками в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html).

класс QTextCharFormat обеспечивает информацию форматирования для символов в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html).

класс QTextCursor предоставляет программный интерфейс для доступа и модификации QTextDocuments.

класс QTextDocument удерживает форматируемый текст. Это контейнер для структурированных документов форматированного текста, обеспечивающий поддержку для стилизованного текста и различных типов элементов документов, таких как списки, таблицы, фреймы и изображения.

класс QTextDocumentFragment представляет часть форматируемого текста из [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html).

класс QTextFormat обеспечивает информацию форматирования для [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Это общий класс для описания формата частей документа. Производные классы [QTextCharFormat](http://qt-project.org/doc/qt-5.1/qtgui/qtextcharformat.html), [QTextBlockFormat](http://qt-project.org/doc/qt-5.1/qtgui/qtextblockformat.html), [QTextListFormat](http://qt-project.org/doc/qt-5.1/qtgui/qtextlistformat.html), and [QTextTableFormat](http://qt-project.org/doc/qt-5.1/qtgui/qtexttableformat.html) обычно более полезные и описывают форматирования, которое можно применить к определённым частям документа.

класс QTextFragment удерживает кусок текста в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html) с одинаковым [QTextCharFormat](http://qt-project.org/doc/qt-5.1/qtgui/qtextcharformat.html).

класс QTextFrame представляет фрейм в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Данный класс обеспечивает структуру текста в документе.

класс QTextFrameFormat обеспечивает информацию форматирования для фреймов в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Фрейм объединяет один или более текстовых блоков, обеспечивая структурный слой, больший, чем параграф.

класс QTextImageFormat обеспечивает информацию форматирования для изображений в документе.

класс QTextInlineObject представляет встроенный объект в [QTextLayout](http://qt-project.org/doc/qt-5.1/qtgui/qtextlayout.html). Этот класс используется, только если текстовый макет используется для определения положения элементов [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html).

класс QTextItem обеспечивает всю информацию, требуемую для рисования текста в частном движке рисования.

класс QTextLayout используется для размещения и рисования текста. Данный класс имеет множество удобных средств для редактирования текста. Данный класс может использоваться для создания набора экземпляров [QTextLine](http://qt-project.org/doc/qt-5.1/qtgui/qtextline.html) заданной ширины и размещать их независимо на экране. Как только макет создан, данные линии могут быть нарисованы на устройстве рисования.

класс QTextLength инкапсулирует разные типы длин, используемых в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). В общем, данный класс используется для определения ширины таблицы.

класс QTextList обеспечивает декорированный список элементов в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Список содержит последовательность текстовых блоков, каждый из которых помечен при помощи булева значения или другим символом.

класс QTextListFormat обеспечивает информацию форматирования для списков в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html).

класс QTextObject является базовым классом для разных видов объектов, которые могут группировать части [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html) вместе. Общие группирующие текстовые объекты – это списки, фреймы и таблицы. Текстовый объект имеет связанный с ним формат и документ. Есть два существенных типа текстовых объектов: те, которые используются вместе с блоками и те, которые используются вместе с символами. Первый тип является производным от [QTextBlockGroup](http://qt-project.org/doc/qt-5.1/qtgui/qtextblockgroup.html), а второй – от [QTextFrame](http://qt-project.org/doc/qt-5.1/qtgui/qtextframe.html). Вам редко необходимо использовать данный класс. Только если вы создаёте частный текстовый объект.

класс QTextObjectInterface позволяет рисование частных текстовых объектов в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html)s. Текстовый объект описывает структуру одного или более элементов в текстовом документе. Текстовый объект знает, как размещать и рисовать его элементы, когда документ будет нарисован.

класс QTextOption обеспечивает описание общих свойств форматируемого текста.

класс QTextTable представляет таблицу в текстовом документе.

класс QTextTableCell представляет свойства ячейки в [QTextTable](http://qt-project.org/doc/qt-5.1/qtgui/qtexttable.html).

класс QTextTableCellFormat обеспечивает информацию форматирования для ячеек таблицы а документе.

класс QTextTableFormat обеспечивает информацию форматирования для таблиц в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html).

класс QAbstractTextDocumentLayout::Selection является удобным классом для определения параметров выделения. класс QTextBlock::iterator обеспечивает итератор для чтения содержимого [QTextBlock](http://qt-project.org/doc/qt-5.1/qtgui/qtextblock.html).

### QAbstractTextDocumentLayout::PaintContext

<http://qt-project.org/doc/qt-5.1/qtgui/qabstracttextdocumentlayout-paintcontext.html#details>

класс QAbstractTextDocumentLayout::PaintContext является удобным классом для определения параметров, используемых во время рисования макета документа. Контекст рисования используется, когда вы рисуете частные макеты для QTextDocuments вместе с функцией [QAbstractTextDocumentLayout::draw](http://qt-project.org/doc/qt-5.1/qtgui/qabstracttextdocumentlayout.html#draw)(). Он определяется позицией курсора, цветом текста по умолчанию, прямоугольником усечения и набором выделений.

### QAbstractTextDocumentLayout

<http://qt-project.org/doc/qt-5.1/qtgui/qabstracttextdocumentlayout.html#details>

класс QAbstractTextDocumentLayout является базовым абстрактным классом, который используется для применения частных макетов для QTextDocuments. Стандартный макет, обеспечиваемый qt, может управлять простой обработкой слов, включая встроенные изображения, списки и таблицы. Если ваше приложение требует больших возможностей, то вам следует создать подкласс QAbstractTextDocumentLayout, чтобы обеспечить поведение частного макета для ваших текстовых документов. Экземпляр данного подкласса может быть установлен на [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html) объект при помощи функции [setDocumentLayout()](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html#setDocumentLayout). Вы также можете вставлять частные объекты в документ, для чего смотрите класс [QTextObjectInterfaceInterface](http://qt-project.org/doc/qt-5.1/qtgui/qtextobjectinterface.html).

### QSyntaxHighlighter

<http://qt-project.org/doc/qt-5.1/qtgui/qsyntaxhighlighter.html#details>

класс QSyntaxHighlighter позволяет вам определить правила синтаксического выделения и также вы можете использовать класс для запроса текущего форматирования документа или данных пользователя. Это базовый класс для реализации синтаксического выделения [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Выделитель часто используется, когда пользователь вводит текст в определённом формате и помогает пользователю читать текст и идентифицировать синтаксические ошибки. Для обеспечения вашего собственного синтаксического выделителя вам следует специализировать данный класс и переопределить функцию [highlightBlock](http://qt-project.org/doc/qt-5.1/qtgui/qsyntaxhighlighter.html#highlightBlock)(). При создании объекта выделителя ему необходимо передать документ, к которому он будет применяться. *Есть демонстрация этого. Также есть пример кода осуществления выделения определённым образом. Также есть пример кода, как делать многострочное редактирование.*

### QTextBlock

<http://qt-project.org/doc/qt-5.1/qtgui/qtextblock.html#details>

класс QTextBlock обеспечивает контейнер для фрагментов текста в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Данный блок инкапсулирует блок или параграф текста в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Данный класс обеспечивает доступ только для чтения к структуре блоков и параграфов текстового документа. он в основном используется, если вы желаете применить ваш собственный формат для визуализации представления [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html), или если вы желаете произвести обход документа и написать его содержание в вашем собственном формате. Текстоввые блоки создаются их родительскими документами. Если вы желаете создать новый текстовый блок или изменить содержания документа во время проверки его содержания, используйте основанный на курсоре интерфейс, обеспечиваемый классом [QTextCursor](http://qt-project.org/doc/qt-5.1/qtgui/qtextcursor.html). Класс предоставляет множество функций для работы с текстовыми блоками.

### QTextBlockFormat

<http://qt-project.org/doc/qt-5.1/qtgui/qtextblockformat.html#details>

класс QTextBlockFormat обеспечивает информацию форматирования для блоков текста в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Документ состоит из набора блоков, представленных объектами [QTextBlock](http://qt-project.org/doc/qt-5.1/qtgui/qtextblock.html). Каждый блок может содержать таблицу, изображение, простой текст и т.д. каждый блок имеет связанный с ним формат, который определяет его характеристики. данный класс обеспечивает много функций для форматирования текста в текстовом блоке.

### QTextBlockGroup

<http://qt-project.org/doc/qt-5.1/qtgui/qtextblockgroup.html#details>

класс QTextBlockGroup обеспечивает контейнер для блоков текста внутри [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Группы блоков могут быть использованы для организации блоков текста внутри документа. текстовые блоки могут быть вставлены в блок и удалены из него. Блоки в списке не обязательно являются ближайшими соседями в документе, например, более высокоуровневые элементы в многоуровневом списке будут отделены от элементов на более низких уровнях.

### QTextBlockUserDate

<http://qt-project.org/doc/qt-5.1/qtgui/qtextblockuserdata.html#details>

класс QTextBlockUserData используется для связи частных данных с блоками текста. Класс обеспечивает абстрактный интерфейс для контейнерных классов, которые используются, чтобы связать специфические для приложения данные пользователя с текстовыми блоками в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html).

В общем подклассы данного класса обеспечивают функции, которые позволяют сохранять данные и получать их, а экземпляры присоединяются к блокам текста с использованием [QTextBlock::setUserData](http://qt-project.org/doc/qt-5.1/qtgui/qtextblock.html#setUserData)(). Это позволяет сохранять дополнительные данные помимо текстовых блоков таким образом, что они могут быть безопасно получены при помощи приложения.

### QTextCharFormat

<http://qt-project.org/doc/qt-5.1/qtgui/qtextcharformat.html#details>

класс QTextCharFormat обеспечивает информацию форматирования для символов в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Формат символов текста в документе определяет визуальные свойства текста, также как и информацию о его роли в гипертекстовом документе. Есть функции для установки шрифта, цвета, а также несколько функций для обработки гиперссылок.

### QTextCursor

<http://qt-project.org/doc/qt-5.1/qtgui/qtextcursor.html#details>

класс QTextCursor предоставляет программный интерфейс для доступа и модификации QTextDocuments. Текстовые курсоры – это объекты, которые используются для доступа и изменения содержания и структуры текстовых документов через программный интерфейс, который имитирует поведение курсора в редакторе текста. Курсор текста содержит информацию о позиции и о выделении, которое он сделал. Для курсора документ может пониматься как единственная строка текста. Данная строка может содержать просто строки, а может содержать и текстовые блоки. Также курсор имеет позицию якорь: текст, находящийся между якорем и текущей позицией, является выделенным. При определённых условиях, *о которых сказано в данной части*, можно получать информацию о формате блоков. Класс обеспечивает функции для редактирования текста. Также данный класс обеспечивает разнообразные функции для вставки фрагментов в текст различной природы.

### QTextDocument

<http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html#details>

класс QTextDocument удерживает форматируемый текст. Это контейнер для структурированных документов форматированного текста, обеспечивающий поддержку для стилизованного текста и различных типов элементов документов, таких как списки, таблицы, фреймы и изображения. Они могут быть созданы для использования в [QTextEdit](http://qt-project.org/doc/qt-5.1/qtwidgets/qtextedit.html) или использованы независимо. Каждый элемент документа описывается при помощи специального формата объекта. документ может редактироваться программно при помощи [QTextCursor](http://qt-project.org/doc/qt-5.1/qtgui/qtextcursor.html). Полная структура документов сохраняется как иерархия элементов документа ниже корневого фрейма. Вы можете создать свой собственный подкласс класса [QAbstractTextDocumentLayout](http://qt-project.org/doc/qt-5.1/qtgui/qabstracttextdocumentlayout.html) и установить его с использованием [setDocumentLayout](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html#setDocumentLayout)(). Также можно получить метаинформацию о документе. Также есть функции, которые позволяют получить содержимое документа как простой текст и как документ HTML. *Также в данном документе описано, как можно обрабатывать и создавать операции undo/redo.*

### QTextDocumentFragment

<http://qt-project.org/doc/qt-5.1/qtgui/qtextdocumentfragment.html#details>

класс QTextDocumentFragment представляет часть форматируемого текста из [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Это фрагмент форматируемого текста, который может быть вставлен в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Фрагмент документа может быть создан из документа, из выделения курсора или из другого фрагмента документа.

### QTextFormat

<http://qt-project.org/doc/qt-5.1/qtgui/qtextformat.html#details>

класс QTextFormat обеспечивает информацию форматирования для [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Это общий класс для описания формата частей документа. Производные классы [QTextCharFormat](http://qt-project.org/doc/qt-5.1/qtgui/qtextcharformat.html), [QTextBlockFormat](http://qt-project.org/doc/qt-5.1/qtgui/qtextblockformat.html), [QTextListFormat](http://qt-project.org/doc/qt-5.1/qtgui/qtextlistformat.html), and [QTextTableFormat](http://qt-project.org/doc/qt-5.1/qtgui/qtexttableformat.html) обычно более полезные и описывают форматирования, которое можно применить к определённым частям документа.

### QTextFragment

<http://qt-project.org/doc/qt-5.1/qtgui/qtextfragment.html#details>

класс QTextFragment удерживает кусок текста в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html) с одинаковым [QTextCharFormat](http://qt-project.org/doc/qt-5.1/qtgui/qtextcharformat.html). Текст, в котором формат символов изменяется, может быть представлен при помощи последовательности фрагментов текста с разным форматом. Если пользователь отредактирует текст в фрагменте и введёт другой формат символов, то текст фрагмента будет разбит в каждой точке, где происходит изменение формата. Новые фрагменты будут созданы. Также данный класс обеспечивает разнообразные функции для получения информации о фрагментах текста.

### QTextFrame

<http://qt-project.org/doc/qt-5.1/qtgui/qtextframe.html#details>

класс QTextFrame представляет фрейм в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Данный класс обеспечивает структуру текста в документе. Они используются как общие контейнеры для других элементов документа. Фреймы обычно создаются при помощи [QTextCursor::insertFrame](http://qt-project.org/doc/qt-5.1/qtgui/qtextcursor.html#insertFrame)(). Они могут использоваться для создания иерархической структуры в документах форматируемого текста. Каждый документ имеет корневой фрейм и каждый фрейм ниже корневого имеет родительский фрейм и список (возможно и пустой) дочерних фреймов. Можно осуществлять обход внутри фреймов. *Как данный класс связан с классами текстовых блоков и групп текстовых блоков?*

### QTextFrameFormat

<http://qt-project.org/doc/qt-5.1/qtgui/qtextframeformat.html#details>

класс QTextFrameFormat обеспечивает информацию форматирования для фреймов в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Фрейм объединяет один или более текстовых блоков, обеспечивая структурный слой, больший, чем параграф. Формат фрейма определяет, как он рисуется и размещается на экране. Он лишь обеспечивает ограничения на макет его наследников. Он определяет ширину и высоту фрейма, также каждый фрейм может иметь визуальную границу и т. д.

### QTextImageFormat

<http://qt-project.org/doc/qt-5.1/qtgui/qtextimageformat.html>

класс QTextImageFormat обеспечивает информацию форматирования для изображений в документе. Встроенные изображения представляются при помощи символа замещения объекта (0xFFFC in Unicode), который имеет связанный с ним QTextImageFormat. Может быть включён любой формат, поддерживаемый qt.

### QTextInlineObject

<http://qt-project.org/doc/qt-5.1/qtgui/qtextinlineobject.html#details>

класс QTextInlineObject представляет встроенный объект в [QTextLayout](http://qt-project.org/doc/qt-5.1/qtgui/qtextlayout.html). Этот класс используется, только если текстовый макет используется для определения положения элементов [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). *Назначение данного класса пока мне не совсем понятно.*

### QTextItem

<http://qt-project.org/doc/qt-5.1/qtgui/qtextitem.html#details>

класс QTextItem обеспечивает всю информацию, требуемую для рисования текста в частном движке рисования. Если вы создаёте некоторый частный движок, то вам следует переопределить данный класс.

### QTextLayout

<http://qt-project.org/doc/qt-5.1/qtgui/qtextlayout.html#details>

класс QTextLayout используется для размещения и рисования текста. Данный класс имеет множество удобных средств для редактирования текста. Данный класс может использоваться для создания набора экземпляров [QTextLine](http://qt-project.org/doc/qt-5.1/qtgui/qtextline.html) заданной ширины и размещать их независимо на экране. Как только макет создан, данные линии могут быть нарисованы на устройстве рисования. *Есть пример кода использования данного класса.*

### QTextLength

<http://qt-project.org/doc/qt-5.1/qtgui/qtextlength.html#details>

класс QTextLength инкапсулирует разные типы длин, используемых в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). В общем, данный класс используется для определения ширины таблицы. Она может быть определена или как фиксированное количество пикселов, или как процентное значение, или при помощи переменной длины, которая позволяет занимать столько места, сколько требуется.

### QTextLine

<http://qt-project.org/doc/qt-5.1/qtgui/qtextline.html#details>

класс QTextLine представляет линию текста внутри [QTextLayout](http://qt-project.org/doc/qt-5.1/qtgui/qtextlayout.html). Данная функция позволяет получать и устанавливать различные параметры текстовых линий.

### QTextList

<http://qt-project.org/doc/qt-5.1/qtgui/qtextlist.html#details>

класс QTextList обеспечивает декорированный список элементов в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Список содержит последовательность текстовых блоков, каждый из которых помечен при помощи булева значения или другим символом. Списки создаются курсором. *Есть пример вставки списка.*

### QTextListFormat

<http://qt-project.org/doc/qt-5.1/qtgui/qtextlistformat.html#details>

класс QTextListFormat обеспечивает информацию форматирования для списков в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Список заполняется одним или более элементом, представляющим из себя блок текста. Формат определяет внешность элементов в списке. *В данной части есть пример кода.*

### QTextObject

<http://qt-project.org/doc/qt-5.1/qtgui/qtextobject.html#details>

класс QTextObject является базовым классом для разных видов объектов, которые могут группировать части [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html) вместе. Общие группирующие текстовые объекты – это списки, фреймы и таблицы. Текстовый объект имеет связанный с ним формат и документ. Есть два существенных типа текстовых объектов: те, которые используются вместе с блоками и те, которые используются вместе с символами. Первый тип является производным от [QTextBlockGroup](http://qt-project.org/doc/qt-5.1/qtgui/qtextblockgroup.html), а второй – от [QTextFrame](http://qt-project.org/doc/qt-5.1/qtgui/qtextframe.html). Вам редко необходимо использовать данный класс. Только если вы создаёте частный текстовый объект. При этом также следует переопределить [QTextDocument::createObject](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html#createObject)() функцию.

### QTextObjectInterface

<http://qt-project.org/doc/qt-5.1/qtgui/qtextobjectinterface.html#details>

класс QTextObjectInterface позволяет рисование частных текстовых объектов в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html)s. Текстовый объект описывает структуру одного или более элементов в текстовом документе. Текстовый объект знает, как размещать и рисовать его элементы, когда документ будет нарисован. Также qt позволяет вставлять частные текстовые объекты в документ при помощи регистрации частных типов объектов при помощи [QTextCharFormat](http://qt-project.org/doc/qt-5.1/qtgui/qtextcharformat.html). Данный класс также должен быть переопределён для данного типа и должен быть зарегистрирован при помощи [QAbstractTextDocumentLayout](http://qt-project.org/doc/qt-5.1/qtgui/qabstracttextdocumentlayout.html) документа. *далее в данной части описаны этапы вставки частного объекта в документ.* Класс, который применяет текстовый объект, нуждается в наследовании как [QObject](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html), так и QTextObjectInterface. [QObject](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html) должен всегда стоять на первом месте. Операции копирования и вставки игнорируют частные объекты.

### QTextOption

<http://qt-project.org/doc/qt-5.1/qtgui/qtextoption.html#details>

класс QTextOption обеспечивает описание общих свойств форматируемого текста. Данный класс используется для инкапсуляции общих свойств форматируемого текста в одиночном объекте. Он содержит стандартные свойства, связанные с рисованием текста и с макетами.

### QTextTable

<http://qt-project.org/doc/qt-5.1/qtgui/qtexttable.html#details>

класс QTextTable представляет таблицу в текстовом документе. Каждая ячейка таблицы содержит блок и окружена фреймом. Таблицы создаются и вставляются в документ при помощи курсоров. *Есть пример кода.* Также следует определять формат таблицы. Также есть функция курсора, которая позволяет получить редактируемую в настоящий момент таблицу. есть функции для редактирования таблицы. Строки и столбцы в таблице могут быть объединены и разделены. *Есть пример кода, как это работает.*

### QTextTableCell

<http://qt-project.org/doc/qt-5.1/qtgui/qtexttablecell.html#details>

класс QTextTableCell представляет свойства ячейки в [QTextTable](http://qt-project.org/doc/qt-5.1/qtgui/qtexttable.html). Это кусочки структуры документа, которые принадлежат таблице. Также данный класс содержит функции для получения и установки определённых параметров ячейки таблицы в текстовом документе.

### QTextTableCellFormat

<http://qt-project.org/doc/qt-5.1/qtgui/qtexttablecellformat.html#details>

класс QTextTableCellFormat обеспечивает информацию форматирования для ячеек таблицы а документе. Данный формат определяет визуальные свойства ячейки таблицы.

### QTextTableFormat

<http://qt-project.org/doc/qt-5.1/qtgui/qtexttableformat.html#details>

класс QTextTableFormat обеспечивает информацию форматирования для таблиц в [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Ячейки в таблице разделены разделителями ячеек. Их можно устанавливать. Также можно устанавливать расстояние от края ячейки до её текстового содержимого при помощи функции [setCellPadding](http://qt-project.org/doc/qt-5.1/qtgui/qtexttableformat.html#setCellPadding)(). Можно задавать фоновый цвет таблицы, но цвет фона каждой ячейки можно устанавливать независимо. Также можно ограничивать размеры структурных элементов таблицы.

### QAbstractTextDocumentLayout::Selection

<http://qt-project.org/doc/qt-5.1/qtgui/qabstracttextdocumentlayout-selection.html#details>

класс QAbstractTextDocumentLayout::Selection является удобным классом для определения параметров выделения.

### QTextBlock::iterator

<http://qt-project.org/doc/qt-5.1/qtgui/qtextblock-iterator.html#details>

класс QTextBlock::iterator обеспечивает итератор для чтения содержимого [QTextBlock](http://qt-project.org/doc/qt-5.1/qtgui/qtextblock.html). Блок состоит из последовательности фрагментов текста. Данный класс обеспечивает способ обхода по ним и чтения их содержания. Он не обеспечивает способ изменения внутренней структуры содержимого блока.

#### QPlainTextDocumentLayout

<http://qt-project.org/doc/qt-5.1/qtwidgets/qplaintextdocumentlayout.html#details>

QPlainTextDocumentLayout класс реализует макет простого текста для класса [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html). Данный класс требуется для текстовых документов, которые могут быть отображены или отредактированы в [QPlainTextEdit](http://qt-project.org/doc/qt-5.1/qtwidgets/qplaintextedit.html). Он использует программный интерфейс [QAbstractTextDocumentLayout](http://qt-project.org/doc/qt-5.1/qtgui/qabstracttextdocumentlayout.html), но переопределяет его для поддержки простого текста.

## РИСОВАНИЕ В 3D

[*http://qt-project.org/doc/qt-5.1/qtgui/painting-3d.html*](http://qt-project.org/doc/qt-5.1/qtgui/painting-3d.html)

### QGenericMatrix

<http://qt-project.org/doc/qt-5.1/qtgui/qgenericmatrix.html#details>

класс QGenericMatrix является шаблонным классом, который представляет матрицу преобразований N\*M.

### QMatrix4x4

<http://qt-project.org/doc/qt-5.1/qtgui/qmatrix4x4.html#details>

класс QMatrix4x4 представляет матрицу преобразований 4\*4 в трёхмерном пространстве. Класс обрабатывается как матрица строк данных. Но внутренне данные сохраняются как столбцы данных, так как это оптимально для передачи в OpenGL, который ожидает данные в виде столбцов. При использовании данных функций будьте уверены, что они возвращают данные в формате столбцов.

### QQuaternion

<http://qt-project.org/doc/qt-5.1/qtgui/qquaternion.html#details>

класс QQuaternion представляет кватернион, состоящий из вектора и скаляра. Они используются для представления вращения в трёхмерном пространстве и состоят из оси вращения, определяемой координатами и скаляра, представляющего угол вращения.

### QVector2D

<http://qt-project.org/doc/qt-5.1/qtgui/qvector2d.html#details>

класс QVector2D представляет вектор или вершину в двумерном пространстве.

### QVector3D

<http://qt-project.org/doc/qt-5.1/qtgui/qvector3d.html#details>

Класс QVector3D представляет вектор или вершину в трёхмерном пространстве. Векторы являются одними из основных блоков построения и рисования в трёхмерном пространстве. Они состоят из трёх координат.

### QVector4D

<http://qt-project.org/doc/qt-5.1/qtgui/qvector4d.html#details>

класс QVector4D представляет вектор или вершину в четырёхмерном пространстве.

### QOpenGLBuffer

<http://qt-project.org/doc/qt-5.1/qtgui/qopenglbuffer.html#details>

класс QOpenGLBuffer обеспечивает функции для создания и управления объектами буфера OpenGL. Они создаются на сервере данной библиотеки, так что клиент приложения может избежать загрузки вершин, индексов, данных изображения текстуры и т.д. каждый раз, когда они необходимы. Буферы могут присваиваться. Но при этом выполняется мелкое копирование. Поэтому оригинальный объект будет подвержен влиянию при изменении копии.

### QOpenGLDebugLogger

<http://qt-project.org/doc/qt-5.1/qtgui/qopengldebuglogger.html#details>

класс QOpenGLDebugLogger даёт возможность создания журнала отладочных сообщений OpenGL. Программирование OpenGL является очень подверженным ошибкам. В большинстве случаев простой сбой данной библиотеки может вызвать полное прекращение работы приложения без рисования на экране. Единственный способ быть уверенным, что никакие ошибки не были возвращены из данной библиотеки – это проверка с помощью функции gjGetError после каждого вызова. Более того ошибки данной библиотеки накапливаются в стеке, поэтому данная функция может всегда быть использована в цикле наподобие того, *что показано в примере кода в данной части документации.* Также есть и другая интересная информация, которую важно получать в тех или иных случаях.

Данный класс преследует своей целью обеспечение доступа к журналу отладки данной библиотеки. Для того чтобы получать отладочную информацию от OpenGL следует установить контекст отладчика в OpenGL. *как это делается, показано в данной части.* Любой класс OpenGL в qt следует инициализировать перед использованием с помощью функции initialize(). *Как это делается, показано в данной части.* Реализация OpenGL содержит внутренний журнал сообщений отладки. Сообщения, сохранённые в данном журнале, могут быть получены при помощи использования функции [loggedMessages](http://qt-project.org/doc/qt-5.1/qtgui/qopengldebuglogger.html#loggedMessages)(). *Есть пример кода её использования.*

Внутренний журнал имеет ограниченный размер. Когда он переполняется, старые сообщения будут отброшены, чтобы освободить место для новых приходящих сообщений. Также внутренний журнал очищается при вызове функции [loggedMessages](http://qt-project.org/doc/qt-5.1/qtgui/qopengldebuglogger.html#loggedMessages)(). Если вы желаете быть уверенным, что нет потери каких-либо сообщений отладки, вы должны использовать журнал реального времени вместо вызова данной функции. Однако отладочные сообщения могут всё ещё генерироваться в интервале времени между созданием контекста и активацией журналирования в реальном времени.

*Как включить протоколирование во время работы, показано в данной части.* Данное протоколирование может быть как синхронным, так и асинхронным. В асинхронной моде OpenGL может генерировать сообщения в любое время и/или в порядке, который отличается от порядка следования тех команд OpenGL, которые его вызвали. Также сообщения могут быть сгенерированы из потока, который отличается от потока, в котором существует контекст. Это связано с тем, что OpenGL обычно в своей реализации сильно распараллелен и асинхронен. Синхронное протоколирование характеризуется обратными свойствами, но оно может вызывать гораздо большее переполнение, нежели асинхронное. Но при использовании синхронной моды вы будете способны использовать точки останова, чтобы остановить выполнение программы именно после той команды, которая вас интересует.

Также отмечается, что при использовании протоколирования во время работы сообщения более не будут помещаться во внутренний журнал OpenGL. после начала протоколирования всегда важно проверять, содержатся ли во внутреннем журнале какие-либо сообщения.

Также возможно для приложений и библиотек вставлять частные сообщения в журнал отладки, например, для создания групп связанных команд OpenGL, чтобы затем было возможно идентифицировать возможные сообщения, приходящие от них. *Есть пример кода использования данных сообщений.* Также следует иметь в виду, что в зависимости от производителя данной библиотеки ограничен размер строк, которые могут быть вставлены в журнал ошибок. Есть функция для получения данного значения. Также данный класс может создавать фильтры для сообщений, тем самым ограничивая количество сообщений в журнале. Это делается при помощи специальных функций, но при этом следует быть внимательным по отношению к порядку следования данных функций, так как это может повлиять на то, какие сообщения будут фильтроваться.

### QOpenGLDebugMessage

<http://qt-project.org/doc/qt-5.1/qtgui/qopengldebugmessage.html#details>

класс QOpenGLDebugMessage обёртывает класс сообщения OpenGL. Сообщения отладки обычно создаются сервером OpenGL и считываются обычно его клиентами. Сообщение отладки имеет текстовое представление, а также зависящий от производителя численный идентификатор, источник, тип и степень тяжести. Для третьих приложений или библиотек возможно создавать и вставлять сообщения в журнал отладки.

### QOpenGLFramebufferObjectFormat

<http://qt-project.org/doc/qt-5.1/qtgui/qopenglframebufferobjectformat.html#details>

класс QOpenGLFramebufferObjectFormat определяет формат OpenGL объекта фреймбуфера. Данный объект имеет несколько характеристик:

* Число образцов на пиксел
* Глубина или трафарет вложения.
* Цель текстуры.
* Внутренний формат текстуры.

Имейте в виду, что желаемые вложения или число образцов на пиксел могут не поддерживаться железом. *Пока что назначение данного класса мне остаётся непонятным.*

### QOpenGLFramebufferObject

<http://qt-project.org/doc/qt-5.1/qtgui/qopenglframebufferobject.html#details>

класс QOpenGLFramebufferObject инкапсулирует OpenGL объект фреймбуфера. Данный класс инкапсулирует OpenGL объект фреймбуфера, определённый при помощи GL\_EXT\_framebuffer\_object. Он обеспечивает поверхность, на которой можно рисовать с помощью класса QPainter c помощью [QOpenGLPaintDevice](http://qt-project.org/doc/qt-5.1/qtgui/qopenglpaintdevice.html), или нарисованного с использованием родных OpenGL вызовов. Эта поверхность может быть ограничена и использована как регулярная текстура в вашем собственном коде рисования OpenGL. Также важно иметь текущий контекст OpenGL, когда создаётся объект данного класса, иначе инициализация будет прекращена. Цвет изображения или текстура будут иметь определённый внутренний формат, и будут ограничены вложением GL\_COLOR\_ATTACHMENT0 в объектк фреймбуфера. Также возможно рисовать в данном объекте с использованием класса QPainter и QOpenGLPaintDevice в отдельном потоке. *Для понимания данного класса следует более детально изучить OpenGL.*

### QOpenGLFunctions

<http://qt-project.org/doc/qt-5.1/qtgui/qopenglfunctions.html#details>

класс QOpenGLFunctions обеспечивает кроссплатформенный доступ к программному интерфейсу OpenGL 2.0. данная библиотека определяет набор определений, которые являются общими среди многих платформ и встроенных реализаций OpenGL. однако сложно использовать функции из этого набора, так как их следует переопределить вручную на целевой системе. Данный класс обеспечивает гарантированный программный интерфейс, который доступен на всех OpenGL системах и следит за разрешением функций на системах, которые нуждаются в нём. *Далее есть большой пример кода. Далее есть объяснение того, что данный класс позволяет использовать функции OpenGL без явного задания разрешения, а также говорится о том, что в данном классе есть обёртки для большинства функций данной библиотеки, кроме трёх, которые также перечислены в данной части. Но я пока плохо понимаю данную тему, так как я существенно не знаком с данной библиотекой.*

### QOpenGLPaintDevice

<http://qt-project.org/doc/qt-5.1/qtgui/qopenglpaintdevice.html#details>

класс QOpenGLPaintDevice предоставляет возможность контексту OpenGL использовать класс QPainter. Данный класс использует текущий OpenGL контекст, чтобы рисовать команды рисовальщика. Он требует поддержку версии 2.0 или выше. Данный класс всегда быстрее, но он также и более чувствителен к изменениям состояния, и поэтому требует, чтобы команды рисования были внимательно упорядочены, чтобы достичь максимальной производительности. Сглаживание требует для OpenGL значительно более ресурсов, но и результат получается лучше.

Когда рисование на данном типе устройства осуществляется средствами класса QPainter, то состояние текущего OpenGL контекста будет изменено движком рисования, чтобы отразить его нужды. Приложениям тогда не следует полагаться на состояние OpenGL, которое сбрасывается в начальное состояние, в частности текущая программа затенения, OpenGL вьюпорт, единицы текстуры и моды рисования. При перемешивании возможностей OpenGL и рисовальщика важно уведомить последнего, что OpenGL состояние может быть приведено в беспорядок, так что оно может пересохранить своё внутреннее состояние. *Там же есть пример функций для осуществления данного действия.*

### QOpenGLShaderProgram

[*http://qt-project.org/doc/qt-5.1/qtgui/qopenglshaderprogram.html#details*](http://qt-project.org/doc/qt-5.1/qtgui/qopenglshaderprogram.html#details)

класс QOpenGLShaderProgram позволяет OpenGL шейдерам быть связанными и использованными. Этот класс поддерживает программы, написанные на языке GLSL и GLSL/ES. Данный класс и [QOpenGLShader](http://qt-project.org/doc/qt-5.1/qtgui/qopenglshader.html) скрывают программиста от деталей компиляции и сборки узловых и фрагментных шейдеров. *Есть пример кода.*

<http://qt-project.org/doc/qt-5.1/qtgui/qopenglshader.html#details>

*то же самое.*

[*http://qt-project.org/doc/qt-5.1/qtgui/qopengltimerquery.html#details*](http://qt-project.org/doc/qt-5.1/qtgui/qopengltimerquery.html#details)

*то же самое.*

[*http://qt-project.org/doc/qt-5.1/qtgui/qopengltimemonitor.html#details*](http://qt-project.org/doc/qt-5.1/qtgui/qopengltimemonitor.html#details)

*то же самое.*

[*http://qt-project.org/doc/qt-5.1/qtgui/qopenglvertexarrayobject.html#details*](http://qt-project.org/doc/qt-5.1/qtgui/qopenglvertexarrayobject.html#details)

*то же самое.*

[*http://qt-project.org/doc/qt-5.1/qtgui/qopenglvertexarrayobject-binder.html#details*](http://qt-project.org/doc/qt-5.1/qtgui/qopenglvertexarrayobject-binder.html#details)

*то же самое.*

### QOffscreenSurface

<http://qt-project.org/doc/qt-5.1/qtgui/qoffscreensurface.html#details>

класс QOffscreenSurface представляет поверхность за кадром в нижележащей платформе. Данный класс предполагается использовать совместно с [QOpenGLContext](http://qt-project.org/doc/qt-5.1/qtgui/qopenglcontext.html), чтобы разрешить рисование при помощи OpenGL в произвольном потоке без необходимости создания [QWindow](http://qt-project.org/doc/qt-5.1/qtgui/qwindow.html). Данный класс может использоваться только для создания OpenGL ресурсов, таких как текстуры. Приложение использует обычно данный класс для выполнения некоторых затратных по времени задач в отдельном потоке, чтобы избежать застоя Главного потока рисования. Ресурсы, созданные при помощи данного класса, могут быть разделены вместе с Главным контекстом OpenGL. как реализуется данный класс, зависит от нижележащей платформы, но он обычно использует буфер пикселов.

<http://qt-project.org/doc/qt-5.1/qtgui/qtgui-module.html>

*здесь есть также очень много классов, содержащих обёртки функций библиотеки OpenGL различных версий. Всего 23 штуки. Их следует просмотреть после изучения данной библиотеки.*

# QML

[*http://qt-project.org/doc/qt-5.1/qtqml/qtqml-index.html*](http://qt-project.org/doc/qt-5.1/qtqml/qtqml-index.html)

qml модуль qt обеспечивает разработку приложения и библиотек при помощи языка qml. Он объявляет и реализует язык и движок и обеспечивает программный интерфейс, чтобы предоставить возможность разработчикам приложения расширить язык qml при помощи частных типов и интегрировать qml код с java scriptом и С++. Заметьте, что если qml модуль обеспечивает язык и инфраструктуру для qml приложения, то QtQuick модуль обеспечивает много визуальных компонентов, поддержку модель-представление и каркас для анимации, а также многое другое для построения интерфейса пользователя. *Далее приводятся некоторые ссылки и типы qml. Теперь переходим к подразделам данного модуля для изучения qml.*

## ОБЗОР

<http://qt-project.org/doc/qt-5.1/qtqml/qmlreference.html>

qml – это декларативный язык для создания высоко динамических приложений. При помощи qml строительные блоки приложения, такие как gui компоненты, объявляются и в них устанавливаются различные свойства, которые определяют поведение приложения. При объединении с JavaScript поведение приложения становится скриптовым. Вдобавок qml сильно использует qt, который позволяет использовать различные особенности qt из qml приложений.

## ОСНОВЫ СИНТАКСИСА

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-syntax-basics.html>

qml – это декларативный язык, который позволяет определить объекты на основании их свойств и того, как они соотносятся и откликаются на изменения в других объектах. В отличие от императивного кода, где изменения в атрибутах и поведении выражаются через набор утверждений, которые обрабатываются шаг за шагом, декларативный синтаксис qml интегрирует атрибуты и изменения в поведении в объявлении индивидуальных объектов. (*разнавіднасці моў – дэкларатыўная і імператыўная*)

Qml исходный код в общем загружается при помощи движка через qml документы, которые являются отдельными документами qml кода. Qml документ может иметь несколько импортов сверху файла. Это могут быть версия пространства имён, в которую типы были зарегистрированы; относительная директория, которая содержит определения типов как qml документы; файлы java script. *Далее есть отдельная информация про импорт java script. Есть примеры импорта.*

Синтаксически блок qml кода определяет дерево qml объектов, которые следует создать. Каждый объект может также содержать объявления дочерних объектов. *Есть пример объявления объекта.* Маленькие объекты можно писать в одну строку. *Есть пример кода*. *Также есть пример кода создания дочерних объектов.* Также отмечается, что иерархически связанные объекты не обязательно являются таковыми на экране. В то время как родительское свойство может быть изменено в визуальном представлении, этого нельзя сделать по отношению к дереву qml. Также описано, какие должны быть в данном языке комментарии. *Есть пример кода.*

## АТРИБУТЫ ОБЪЕКТОВ QML

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-syntax-objectattributes.html>

каждый qml тип объекта имеет определённый набор свойств. Каждый экземпляр типа объекта создаётся с набором свойств, которые были определены для этих объектов. Есть несколько разных видов атрибутов, которые могут быть определены. Типы атрибутов qml:

* идентификатор,
* свойство,
* сигналы,
* обработчики сигналов,
* методы,
* присоединённые свойства
* присоединённые обработчики сигналов.

Каждый qml объект имеет ровно один идентификатор атрибут. Он обеспечивается самим языком. Идентификатор присваивается объекту, чтобы он мог быть распознан другими объектами. *Есть правила написания идентификаторов. Есть интересный пример кода установки идентификатора.* Свойство – это атрибут объекта, которому может быть присвоено статическое значение или связано с динамическим выражением. Свойства обычно открыты, но их можно делать и закрытыми. Свойства могут определятся для типов С++, в которых определено Q\_PROPERTY. *Также есть пример обычного синтаксиса объявления свойства.* Свойства должны начинаться со строчных букв и содержать только буквы, числа и нижние подчеркивания. *Это связано с тем, что в данном языке автоматически создаются обработчики сигналов для обработки таких свойств.* Объявление частного свойства неявно создаёт также сигнал для данного свойства и связанный с ним обработчик сигнала. *Есть примеры их названий. Есть примеры кода.* Все базовые типы qml кроме перечисления могут быть использованы как типы частных свойств. *Есть пример кода.* Базовый тип var – это тип, который может содержать любое значение, включая списки и объекты. *Далее показано присваивание значения свойства при инициализации. Далее показано императивное присваивание значения свойства. Далее есть таблица описания, что является статическим значением, а что является связывающим выражением. Есть примеры задания свойств как первого, так и второго типа.* Свойства являются безопасными по типу: свойству может быть присвоено только такое значение, которое совпадает с типом свойства. *Далее показано, как создавать списки свойств.* Если вы желаете объявить свойство для сохранения списка значений, которые не являются qml типами, вам следует объявить вместо них свойство var. *Далее показано создание групп свойств: при помощи соглашения точки и соглашения группы.* Псевдонимы свойств – это свойства, которые удерживают ссылку на другое свойство. Псевдоним соединяет вновь созданное свойство с уже существующим свойством. *Есть пример кода их использования.* Псевдонимы активируются только после полной инициализации компонента. Ошибка генерируется, когда ссылаются на неинициализированный псевдоним. Также нельзя создавать псевдоним псевдонима. Если псевдоним имеет такое же имя, как некоторое свойство объекта, то он его переписывает. *Есть пример кода.* Объявление объекта может иметь одно свойство по умолчанию. Это свойство, которому присваивается значение, если объект объявляется внутри другого объекта без объявления значения для конкретного свойства. *Есть специальное ключевое слово для создания свойств только для чтения. Данное свойство не может быть свойством по умолчанию или псевдонимом свойства. Есть пример кода. Далее показано, как объявлять модификатор свойства.*

Сигнал – это уведомления от объекта, что некоторое событие произошло. Объект может быть уведомлён через обработчик сигналов. В блоке обработчика должен содержаться код java script, который следует выполнить. *Есть пример кода. Есть описание синтаксиса объявления сигналов и их обработчиков.* Сигнал может быть определён для типа С++ при помощи регистрирования Q\_SIGNAL класса, который затем регистрируется при помощи системы типов qml. *Также есть пример объявления частного сигнала. Д*ля испускания сигнала его следует запросить как метод. Qml типы также обеспечивают встроенные сигналы изменения свойства, которые испускаются всякий раз, как свойство изменяется. Добавление сигнала в определение объекта в qml автоматически добавит связанный обработчик сигналов к определению объекта, который, по умолчанию, будет пуст. *Есть пример создания сигналов и их обработчиков, которые могут находиться в различных qml файлах.*

Методы – это некоторые функции, которые могут быть вызваны. Они могут соединяться с сигналами. *В данной части показано, как определить метод для типа С++ или частный метод. Есть пример использования метода.*

Присоединённые свойства или присоединённые обработчики сигналов – это механизмы, которые позволяют объектам быть аннотированными экстра свойствами или обработчиками сигналов, которые иначе будут недоступными для объектов. В частности, они позволяют объектам получить доступ к свойствам или сигналам, которые являются уместными для индивидуальных экземпляров. Экземпляры таких свойств могут затем быть созданы и добавлены к объекту во время выполнения. Общей ошибкой является полагать, что присоединённые свойства и обработчики сигналов являются доступными напрямую из наследников объекта, к которым данные атрибуты были присоединены. Это не тот случай. Экземпляр присоединённого типа является только присоединённым к определённому объекту, но не к объекту и ко всем его наследникам. *Также ниже есть пример кода того, чем отличаются присоединённые свойства и обработчики сигналов от обычных.*

## СВЯЗЫВАНИЕ СВОЙСТВ

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-syntax-propertybinding.html>

Для того чтобы использовать qml полностью и его встроенную поддержку для применения динамических изменений поведения объектов, большинство qml-объектов будет использовать связывание свойств. Это центральная особенность qml, которая позволяет объектам автоматически обновлять свои свойства в ответ на изменения атрибутов других объектов или на появление некоторых внешних событий. Когда свойству объекта присваивается значение, ему может быть присвоено или статическое значение или оно может быть связано с java script выражением. В первом случае, свойство не изменится, пока ему не будет присвоено новое значение. В последнем случае связывание свойства создаётся и значение свойства автоматически обновляется при помощи qml движка всякий раз, когда значение оцениваемого выражения изменяется. *В данной части есть пример кода связывания свойства. Будьте внимательны, так как если в будущем свойству присвоится статическое значение, то это удалит связывание.*

Если вы желаете одно связывание заменить другим связыванием, то используйте функцию Qt.binding(), которая обеспечивает желаемый результат. *В данной части есть пример кода использования данной функции. Также в данной части показаны некоторые особенности связывания свойств, когда в иерархии некоторые свойства имеют одинаковые названия. Тогда следует применять, например, ключевое слово this.*

## СИСТЕМА СИГНАЛОВ И ОБРАБОТЧИКОВ СИГНАЛОВ

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-syntax-signals.html>

приложение и компоненты интерфейса пользователя нуждаются в общении друг с другом. *Есть пример связывания сигнала с обработчиком сигнала.* Обработчики сигналов изменения свойств обеспечиваются неявно и их не найти в документации соответствующих классов. Модуль Qt Quick обеспечивает тип Connections для присоединения сигналов к произвольным объектам. Не обязательно внутри объекта, который испускает данный сигнал. *Есть пример кода использования соединений.* Присоединённый обработчик сигнала получает сигнал от присоединённого типа, а не от объекта, внутри которого обработчик объявлен. *Далее показано, как создавать и использовать частные сигналы. Далее показано, как присоединять сигналы к методам и другим сигналам. А есть также метод для удаления соединения сигналов.*

## ИНТЕГРИРОВАНИЕ QML И JAVA SCRIPT

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-javascript-topic.html>

qml использует JSON-like синтаксис и позволяет определять различные выражения и методы как функции java script. Также можно импортировать и использовать функциональность java script файлов.

Qml движок обеспечивает java script среду, которая имеет несколько отличий от java script среды, обеспечиваемой веб браузером. *Данные ограничения перечислены в данной части.*

### JAVA SCRIPT ВЫРАЖЕНИЕ В QML ДОКУМЕНТАХ

[*http://qt-project.org/doc/qt-5.1/qtqml/qtqml-javascript-expressions.html*](http://qt-project.org/doc/qt-5.1/qtqml/qtqml-javascript-expressions.html)

java script хост среда, предоставляемая qml, может выполнять доступные стандартные java script конструкты, такие как условные операторы, массивы, различные настройки и циклы. Ещё вдобавок к стандартным свойствам java script qml Глобальный объект включает несколько вспомогательных методов, которые упрощают построение графического интерфейса и взаимодействие с qml средой. Java script среда в qml строже, чем среда в веб браузере. Например, в qml вы не можете добавить или изменить члены Глобального объекта java script. Все локальные переменные поэтому должны явно объявляться. Qml типы объектов могут использовать java script выражения, которые выполняют логику программы. Есть четыре способа для этого:

* связывание свойств,
* обработчики сигналов,
* частные методы,
* импортированные java script файлы.

*Есть пример кода связывания свойства.* Любое java script выражение может быть связано со свойством, если его возвращаемое значение может быть присвоено соответствующему свойству. Есть два способа связывания свойств:

* при инициализации и
* с использованием Qt.binding().

*Есть пример использования обработчиков сигналов.* Java script логика может быть определена в частных функциях, создаваемых пользователем. *Есть пример кода частного метода с логикой java script.* Важно, что частные методы, определённые встроенными в qml документе, показываются другим объектам и поэтому встроенные функции в корневом объекте в qml компоненте могут быть запрошены вне компонента. Если это нежелательно, то метод может быть добавлен в некорневой объект или желательно написан во внешнем java script файле. *Есть пример кода импортирования java script файлов. Также есть пример, как соединять сигналы с java script функциями.*

Иногда необходимо запустить некоторый императивный код при запуске приложения. Заманчиво просто включить начальный скрипт как Глобальный код во внешний скриптовый файл. Это может иметь тяжёлые ограничения, так как qml среда может не быть полностью установлена. Каждый qml объект имеет добавленное Component свойство, которое ссылается на компонент, внутри которого объект был инстанцирован. Каждый компонент будет эмитировать сигнал о завершении и, таким образом, каждый объект может определить определение для сигнала onCompleted(), который может быть использован для вызова выполнения скриптового кода при запуске после того, как среда qml будет полностью установлена. Если существует более одного обработчика onCompleted(), то они будут вызываться последовательно в неизвестном порядке. Также есть присоединённый обработчик сигнала для выполнения некоторого кода при разрушении компонента.

### ОПРЕДЕЛЕНИЕ JAVA SCRIPT РЕСУРСОВ В QML

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-javascript-resources.html>

программная логика для qml приложения может быть определена в java script. Данный код может быть определён или в документе qml или отдельно в java script файлах (известных в qml как джс ресурсы). Есть два вида java script ресурсов, поддерживаемых qml:

* файлы с выделенным кодом реализации и
* разделяемые файлы (библиотеки).

Для того чтобы код java script вёл себя корректно в qml документе, необходимо создавать в каждом документе копию java script объекта. В общем, простая логика должна быть определена в qml файле, а более сложная логика должна быть отделена в отдельных ресурсах выполнения. Некоторые java script файлы действуют больше как библиотеки – они обеспечивают набор вспомогательных функций, которые имеют некоторую вводную информацию и рассчитывают что-то на выходе, но никогда не манипулируют qml компонентами напрямую.

*Заметьте, что множественные qml документы могут импортировать библиотеку java script и вызывать одну и ту же функцию. состояние java script импорта в данном случае разделяется среди qml документов, которые импортируют его, и поэтому возвращаемое значение функции может быть ненулевым, когда она вызывается внутри qml документа, который никогда не вызывал функции. Есть пример кода, демонстрирующий данное обстоятельство.*

*Java script разделяемая библиотека одна для многих qml компонентов, тогда как ресурс реализлации некоторого кода связывается с каждым экземпляром компонента.*

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-javascript-hostenvironment.html> (недействительная ссылка)

java script ресурсы могут импортироваться qml документами и другими java script ресурсами. Java script ресурсы могут импортироваться через относительные или абсолютные url. В случае относительных url локация решается относительно локации qml документа, содержащего импорт. *Есть некоторая информация относительно загрузки файла ресурсов из интернета. Далее показано, как импортировать java script ресурс в qml документ. Квалификаторы java script ресурса должны быть всегда уникальны.*

В Qt Quick 2.0 была добавлена поддержка того, чтобы java script ресурсы могли импортировать другие java script ресурсы и также пространства имён qml с использованием вариаций стандартного qml синтаксиса импорта. *Затем перечислены некоторые дополнительные правила по импортированию ресурсов в данном случае, а также приведены примеры их использования.*

Иногда желательно в одном java script файле обращаться к функциям в другом java script файла без использования квалификаторов. В этом случае можно вызвать функцию Qt.include(), которая игнорирует все прагмы и импорты из некоторого файла, но которая просто копирует функции из данного в файла в другой java script файл. *Есть пример кода.*

### JAVA SCRIPT ХОСТ СРЕДА

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-javascript-hostenvironment.html>

qml обеспечивает java script хост среду с учётом написания qml приложений. Эта среда отличается от хост среды браузера или среды java script со стороны сервера. Например, qml не обеспечивает объект окна или DOM программный интерфейс, как обычно происходит в среде браузера. Qml применяет тот же стандарт [ECMAScript Language Specification](http://www.ecma-international.org/publications/standards/Ecma-262.htm), что и браузер или серверная среда. Стандартные встроенные возможности данного стандарта не показаны явно в документации qml. *Но есть ссылка на ресурс в пдф формате*. Qml java script среда выполняет определённый набор хостовых объектов и функций. Они всегда доступны, несмотря на то, были ли импортированы модули. *Также в данной части показаны добавления, которые характерны для qml по отношению к стандартным объектам и функциям java script.*

qml следует некоторым ограничениям кода java script:

* Java script код не может модифицировать Глобальный объект. *Есть описание и пример кода.*
* Глобальный код выполняется в уменьшенной области видимости. *Есть пример кода.*
* В большинстве контекстов значение this не определено в qml. This обеспечивается только при связывании свойств из java script. В других случаях значения этого ключевого слова не определено в qml. Для ссылки на конкретный объект используйте идентификатор.
* Утверждение with считается устаревшим.
* Java script связанные выражения выполняются не в строгой моде. Но в qt 5.2 планируется это изменить.

### ДИНАМИЧЕСКОЕ СОЗДАНИЕ QML ОБЪЕКТОВ ИЗ JAVA SCRIPT

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-javascript-dynamicobjectcreation.html>

qml поддерживает динамическое создание объектов из java script. Это полезно для задержки инстанцирования объектов, пока это необходимо, тем самым улучшая время загрузки приложения. Это также позволяет динамически создавать и добавлять визуальные объекты на экран в ответ на реакцию ввода пользователя или других событий. *Есть ссылка на пример.*

Есть два способа создания динамически объектов из java script.

* [Qt.createComponent()](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-qt.html#createComponent-method) динамически создаёт Component объект.
* [Qt.createQmlObject()](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-qt.html#createQmlObject-method) динамически создаёт объект из строки qml.

Для динамической загрузки компонента, определённого в qml файле, вызовите [Qt.createComponent()](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-qt.html#createComponent-method) в [Qt object](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-qt.html#qmlglobalqtobject). Однажды получив компонент, вы можете вызвать [createObject()](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-component.html#createObject-method) для создания экземпляра этого компонента. *При создании объекта вы можете определить его родителя, а также начальные свойства. Есть пример кода динамического создания объекта. Есть пример второго способа динамического создания объекта, который мне также понятен. Далее следует обсуждение того, что контекст, в котором создаётся объект, должен быть действителен. Описано, что является контекстом для каждого способа создания объекта.*

Во многих интерфейсах пользователя достаточно установить непрозрачность визуального объекта в нуль или убрать визуальный объект с экрана вместо его удаления. Но если объектов много, то удаление объектов может улучшить производительность. Заметьте, что вам не следует никогда вручную удалять объекты, которые были динамически созданы при помощи удобных qml фабрик объектов. Также вам следует избегать удаления объектов, которые вы не создаёте динамически.

*Есть замечательный пример кода, в котором демонстрируется удаление объектов, которые были созданы динамически.* Могут быть уничтожены только объекты, которые были созданы динамически.

### ИНТЕГРИРОВАНИЕ JAVA SCRIPT В QML

<http://qt-project.org/doc/qt-5.1/qtdoc/qtquick-usecase-integratingjs.html>

интегрирование в qml java script. Java script код может быть легко интегрирован в qml, чтобы обеспечить логику графического интерфейса, императивный контроль или другие преимущества. *далее приводятся примеры использования java script выражений, функций и файлов. Я уже всё это читал ранее.*

## СИСТЕМА ТИПОВ QML

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-typesystem-topic.html>

типы, которые могут быть использованы в определении иерархии объектов в документе qml, могут прийти из различных источников. Это могут быть:

* родные для языка qml типы;
* зарегистрированные через С++ при помощи модулей qml;
* обеспечиваемые как qml документы при помощи qml модулей.

Есть базовые типы qml. Обеспечены java script типы. Каждый стандартный java script тип может быть создан и сохранён с использованием общего var типа. *Есть пример кода создания стандартных типов java script.* Qml объектный тип является типом, из которого qml объект может быть инстанцирован. Qml объектные типы являются производными от QtObject и обеспечиваются qml модулями. Приложения могут импортировать данные модули для использования объектов, которые они обеспечивают. QtQuick модуль обеспечивает наиболее общие типы объектов, необходимые для создания интерфейсов пользователя в qml. Наконец, каждый qml документ неявно определяет qml объектный тип, который может быть повторно использован в других qml документах.

### БАЗОВЫЕ ТИПЫ

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-typesystem-basictypes.html>

базовый тип – это тип, который относится к простому значению, такому как инт или стринг. Данный тип является другим в отличие от объектных типов qml, которые могут иметь свойства, сигналы, методы и т.д. базовый тип не может быть использован для объявления qml объектов. Данные типы используются для обозначения простых значений или значений, которые содержат простой набор пар свойство-значение. Некоторые базовые типы поддерживаются движком по умолчанию, а некоторые могут быть использованы только с импортом соответствующего модуля. *В данной части перечислены некоторые ограничения на использование базовых типов. Далее есть таблица поддерживаемых базовых типов qml. Далее есть базовые типы, обеспечиваемые qml модулями.*

Глобальный объект qt предоставляет полезные функции для манипулирования базовыми типами. Для использования некоторых базовых типов некоторого qml модуля, его необходимо импортировать в qml документы. У некоторых базовых типов есть некоторые свойства. Но у них нет сигналов изменения данных свойств. Вместо этого есть сигнал просто для изменения базового типа. В объектных типах для каждого свойства есть сигнал, который испускается только тогда, когда данное свойство реально изменяется.

### ОБЪЕКТНЫЕ ТИПЫ QML

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-typesystem-objecttypes.html>

qml тип объекта – это тип, из которого может быть инстанцирован qml объект. Создатели плагинов и разработчики приложений могут обеспечить типы, определённые как qml документы. Qml документ, когда обнаруживается системой импорта qml, определяет тип, идентифицируемый именем файла минус расширение файла. Ещё одним способом создания объектов из qml является использование типа Component. Это позволяет определение типа внутри документа qml вместо создания нового документа для данного типа. *Есть пример кода.*

Пространство имён компонента отличается от пространства имён документа, поэтому имена можно дублировать. С++ разработчики плагина и приложения могут зарегистрировать типы, определённые в С++ через программный интерфейс, обеспечиваемый qml модулем. Qml система типов полагается на импорты, плагины и расширения, установленные в известном пути импорта. Плагины могут быть обеспечены сторонними разработчиками и повторно использованы разработчиками приложения.

### ОПРЕДЕЛЕНИЕ ОБЪЕКТНЫХ ТИПОВ ПРИ ПОМОЩИ QML ДОКУМЕНТОВ

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-documents-definetypes.html>

одной из центральных особенностей qml является то, что он позволяет легко определять объекты при помощи документов в соответствии с потребностями отдельных qml приложений. Стандартный модуль Qt Quick обеспечивает различные типы для построения qml приложений. Но вы можете легко создавать и свои собственные объекты. Для создания объектного типа при помощи qml документа следует создать текстовый файл с именем <TypeName>.qml, который должен начинаться с прописной буквы. Этот документ распознаётся движком автоматически, как тип объекта. Также он становится доступным остальным qml файлам в той же директории. *Рекомендуется, чтобы имя файла в точности совпадало с имененм qml объекта. Есть пример кода создания qml документа и его дальнейшего использования.* Для того, чтобы пользоваться объектами, которые определены в файлах qml в других директориях, следует их импортировать. Корневой объект в qml файле определяет атрибуты, которые становятся доступны для qml типа. *Есть пример кода установки различных атрибутов объектов qml.* Следует отметить, что идентификаторы доступны только в области видимости того компонента, в котором они были созданы. Нельзя обратиться к ним из другого компонента.

### ОПРЕДЕЛЕНИЕ ТИПОВ QML ИЗ С++

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-cppintegration-definetypes.html>

при расширении qml при помощи С++ кода класс С++ может быть зарегистрирован при помощи qml системы типов, чтобы позволить использовать класс как тип данных в qml коде. Для использования данного типа в qml его следует зарегистрировать в системе типов qml. В дополнение qml модуль обеспечивает механизмы для применения в С++ специфических для qml механизмов, таких как присоединённые свойства или свойства по умолчанию. QObject наследник может быть зарегистрирован в системе типов qml. Можно регистрировать как инстанцируемые, так и неинстанцируемые классы. Каждый класс С++, который наследует QObject, может быть зарегистрирован как определение типа объекта qml. Когда зарегистрирован, класс может использоваться как любой другой qml объект. *Есть замечательный пример кода регистрации и использования С++ класса в qml приложении.* С++ классы, которые могут быть неинстацируемыми, полезны в следующих случаях:

* как интерфейсы,
* как базовые классы, которые не нужно отображать в qml;
* которые объявляют некоторые перечисления, к которым должен быть доступ из qml, но экземпляры класса не нужны;
* как тип, который должен быть одиночкой.

*Далее перечислены 4 функции для регистрации различных вариантов классов С++.* Объекты одиночки не связаны с QQmlContext и разделяются различными контекстами. Они создаются QQmlEngine и им же уничтожаются.

Q\_PROPERTY и Q\_INVOKABLE могут использоваться из синглетона. Они могут быть использованы вместо .pragma библиотеки, чтобы сохранять Глобальное состояние или обеспечить Глобальную функциональность. *Есть пример кода использования такого синглетона. Отмечается, что он замечательно подходит для создания различных стилей.* Многие типы функций регистрации требуют задания версий. Они позволяют существовать в новых версиях новым методам и свойствам, вместе с тем обеспечивая совместимость со старыми версиями. В qml есть такая особенность, как присоединяемые методы и свойства. Это означает, что к объекту можно присоединить некоторые свойства и методы, которые определяются в специальном присоединяемом типе, вместо того, чтобы определить их в самом типе объекта. *есть блестящий пример кода, который демонстрирует данное свойство. Далее показано, как сделать объект С++ таким присоединённым типом. А далее есть пример, который прекрасно демонстрирует суть использования присоединённых свойств и методов в С++ с использованием средств qml.*

Модификатор свойства является особенным видом объектного типа qml. Экземпляр данного объекта влияет на свойства, к которому он применяется. Есть два вида модификаторов свойств:

* перехватчики записи свойства и
* источники свойства.

Первый может быть использован для фильтрации и изменения значений свойства при записи. Источник может быть использован для автоматического обновления свойства во времени. Пока клиенты могут создавать только собственные источники свойства. *Есть пример кода создания модификатора свойства.* Источники свойства – это qml типы, которые автоматически обновляют свойство во времени. Например, это различные типы анимации свойств модуля Qt Quick. *Есть пример использования источника свойств.*

Каждый QObject класс, который зарегистрирован как инстанцируемый объект qml, может быть определён как свойство по умолчанию для данного типа. Свойство по умолчанию – это свойство, к которому наследники объекта автоматически присваиваются, если им не присвоены некоторые определённые значения. *Есть очень понятный и простой пример кода использования свойства по умолчанию. В данной части есть ссылка на некоторое руководство, которое позволяет потренироваться в написании qml расширений при помощи С++. Также написано, что необходимо сделать для того, чтобы включить в Qt Quick приложение визуальный класс, написанный на С++.*

Для некоторых объектов qml предпочтительно задерживать инициализацию некоторых данных, пока объект не будет создан и все его свойства не будут установлены. Qml модуль обеспечивает QQmlParserStatus класс, который можно специализировать для данных целей. Он оперделяет набор виртуальных функций, которые вызываются на различных стадиях во время инстанцирования компонентов. *Есть пример кода того, как использовать данный класс в некотором С++ классе.*

## QML МОДУЛИ

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-modules-topic.html>

qml модуль обеспечивает типы с версиями и java script ресурсы в пространстве имён типов, которые могут быть использованы клиентами, кто импортирует модуль. Типы, которые обеспечивает модуль, могут быть определены при помощи С++ внутри плагина, или при помощи qml документов. Модули используют систему версий qml, которая позволяет независимо обновлять модули. Определение qml модуля позволяет:

Разделение общих qml типов внутри проекта;

* распространение qml библиотек;
* модульность некоторых особенностей, так что приложения только загружают библиотеки, необходимые для их индивидуальных нужд;
* создание версий типов и ресурсов, так что модуль может быть обновлён безопасно без разрешения клиентского кода.

Модуль определяется при помощи [module definition .qmldir file](http://qt-project.org/doc/qt-5.1/qtqml/qtqml-modules-qmldir.html). Каждый модуль имеет связанное с ним пространство имён. Модуль обеспечивает qml объектные типы (определённые через qml документы или через С++ плагины), и java script ресурсы, а также могут быть импортированы клиентами. Для создания модуля все файлы, входящие в него должны собраться в одной директории, после чего в ней же создаётся файл модуля, а затем директория устанавливается в qml импорт путь в качестве модуля. В qml есть два типа модулей:

* идентифицируемые модули и
* некоторые другие, которые уже устарели.

Первые явно определяют их идентификатор и устанавливаются в qml импорт пути. Приложение, которое имеет много логики, реализованной в виде С++, или которое определяет типы в С++ и показывает их qml, может желать реализовывать qml плагин. Разработчик расширения модуля qml может желать реализовать несколько типов в С++ плагине, чтобы достичь лучшей производительности или большей гибкости. Каждый С++ плагин для qml имеет функцию инициализации, которая вызывается qml движком, когда он загружает плагин. Эта функция должна регистрировать все типы, которые обеспечивает плагин, но больше не должна ничего делать.

### ФАЙЛЫ ОПРЕДЕЛЕНИЯ МОДУЛЕЙ QML

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-modules-qmldir.html>

существует два различных типа qmldir файлов:

* файлы, содержащие директорию документа,
* а также файлы определения qml модуля.

Файл определения модуля является простым текстовым файлом, который состоит из следующих команд. *Далее идёт понятная таблица команд.* Каждая команда должна быть на отдельной строке. Типы, которые экспортируются для некоторой версии, всё ещё остаются доступными, если более поздняя версия импортируется. Если модуль обеспечивает кнопку в версии 1.0 и тип окна в версии 1.1, клиенты, которые импортируют версию 1.1 модуля будут способны использовать кнопку и окно. Однако, обратное не справедливо. Версия не может быть импортирована, если никакие типы не были явно экспортированы для данной версии. Если модуль обеспечивает кнопку в версии 1.0 и окно в версии 1.1, вы не можете импортировать версию 1.2 или версию 2.0 данного модуля. *Не понимаю этого. Теперь понимаю. Если в списке файлов нет ни одного файла с данной*

*версией библиотеки, то данная версия не может быть экспортирована.* Тип может быть определён разными файлами в разных версиях. В данном случае будет использована наиболее близкая версия. *Есть понятный пример кода. Далее есть пример кода использования модуля и файла модуля. Очень понятный мне код.*

Qml модули могут ссылаться на один или более файлов информации о типах. Они обычно имеют расширение .qmltypes. их единственное предназначение – это обеспечить сторонним службам проверку завершённости кода, а также проверку ошибок и функциональности модуля. Если модуль использует плагины, то данный файл должен существовать. .qmltypes считываются внешними инструментами, чтобы получить информацию о типах, определённых в плагинах. qmlplugindump обеспечивается qt для автоматической генерации таких файлов. *Было бы интересно найти пример создания плагина для qml.*

### ИДЕНТИФИЦИРУЕМЫЕ МОДУЛИ

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-modules-identifiedmodules.html>

идентифицируемые модули – это модули, которые устанавливаются и являются идентифицируемыми qml движком при помощи url в форме помеченного точкой строки идентификатора, которую следует определять при помощи qmldir файла. Это позволяет импортировать такие модули с уникальным идентификатором, который остаётся одинаковым независимо от того, где модуль находится в локальной файловой системе. При импортировании модуля используется его идентификатор и обязательная версия. Каждый qml файл должен импортировать данную директорию, чтобы использовать её функциональность. *Также описывается, в каких путях можно хранить модули. Я это изучу подробнее позже. Пока понял слабо. Далее есть понятное объяснение с примерами того, где следует хранить модуль qml.*

Qml движок гарантирует, что другие модули не способны изменить или переписать типы в пространстве имён данного модуля; не способны зарегистрировать новые типы в пространстве имён модуля; использование имён типов клиентами решит детерминированно определённый тип определения в зависимости от версии, определённой в порядке импорта. Qml модуль имеет ограничения:

* должен быть установлен в qml путь импорта; идентификатор модуля должен совпадать с путём установки модуля;
* модуль должен зарегистрировать его типы внутри пространства имён идентификаторов типов модуля;
* клиенты должны определить версию при импортировании модуля.

*Есть примеры того, как формируется идентификатор модуля исходя из того, где данный модуль расположен. Эти примеры мне очень понятны.*

## СОЗДАНИЕ С++ ПЛАГИНОВ ДЛЯ QML

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-modules-cppplugins.html>

qml движок загружает С++ плагины для qml. Такие плагины обычно обеспечиваются в модулях расширения qml, и могут обеспечить типы и функциональность для использования клиентами в qml документах, которые импортируют модуль. [QQmlExtensionPlugin](http://qt-project.org/doc/qt-5.1/qtqml/qqmlextensionplugin.html) является интерфейсом плагина, который позволяет создавать расширения qml, которые могут быть динамически загружены в qml приложения. Эти расширения позволяют делать доступными qml движку частных типов. *Далее в данной части описаны этапы создания плагина. Есть пример кода создания плагина. В целом, он понятен.*

## QML ДОКУМЕНТЫ

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-documents-topic.html>

qml документ – это строка, которая соответствует qml синтаксису документа. Документ определяет объектный тип qml. Документ обычно загружается из .qml файла или локально или удалённо, но может быть построен и вручную в коде. Экземпляр объекта, определённого документом, может быть создан с использованием Component в qml коде или с использованием QQmlComponent в С++. Способность определять повторно используемые qml типы объектов в документах является важным помощником для создания модульного, хорошо читаемого и поддерживаемого кода. Документ состоит из двух секций:

* секция импорта и
* секция объявления объекта.

Секция объявления должна придерживаться подходящего синтаксиса qml. Как было описано ранее, документ неявно определяет qml тип объекта. Одним из ключевых принципов qml является способность определить и затем повторно использовать типы объектов. Это улучшает поддерживаемость qml кода, увеличивает читаемость объявлений объектной иерархии и позволяет отделить графический интерфейс и логику реализации. Выражения в документах обычно вовлекают объекты или их свойства и поэтому множественные объекты могут быть определены и поэтому разные объекты могут иметь свойства с одинаковыми именами.

### СТРУКТУРА QML ДОКУМЕНТА

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-documents-structure.html>

qml документ содержит в себе qml код, который состоит из двух частей:

* утверждения импорта и
* объявление единственного корневого объекта.

по соглашению импорт отделяется от определения объектов одной строкой. Qml документы всегда закодированы в формате UTF-8.

Пространство имён определяется модулем. Любому qml файлу доступны объекты из той же директории, из которой он сам. Для использования объектов из других модулей необходимо объявить соответствующее пространство имён. Qml документ описывает иерархию объектов, которые можно инстанцировать. Каждое определение объекта имеет определённую структуру: оно имеет тип, оно может иметь идентификатор и имя объекта, оно может иметь свойства, оно может иметь методы, оно может иметь сигналы и может иметь обработчики сигналов.

Qml документ должен только содержать определение единственного корневого объекта. *есть пример кода.* Это связано с тем, что qml документ инкапсулирует только определение одного qml объекта.

### РАЗВЁРТЫВАНИЕ QML ДОКУМЕНТОВ

[*http://qt-project.org/doc/qt-5.1/qtdoc/qtquick-deployment.html*](http://qt-project.org/doc/qt-5.1/qtdoc/qtquick-deployment.html)

qml документы загружаются и выполняются qml во время выполнения. Это включает декларативный движок графического интерфейса вместе с встроенными типами qml и модулями плагинов, а также это обеспечивает доступ к сторонним qml типам и модулям. Приложения, которые используют qml, должны запрашивать qml во время выполнения, чтобы выполнять qml документы. Это может быть сделано при помощи создания [QQuickView](http://qt-project.org/doc/qt-5.1/qtquick/qquickview.html) или [QQmlEngine](http://qt-project.org/doc/qt-5.1/qtqml/qqmlengine.html). Также декларативный пользовательский интерфейс включает qmlscene, который загружает .qml файлы. Этот инструмент полезен для разработки и тестирования qml кода без необходимости написания С++ приложения для загрузки qml во время выполнения. Для развёртывания приложения, которое использует qml, qml должен быть запрошен во время выполнения приложением. Это делается при помощи написания qt С++ приложения, которое загружает qml движок одним из следующих способов:

* загрузка qml файла через экземпляр [QQuickView](http://qt-project.org/doc/qt-5.1/qtquick/qquickview.html).
* создание экземпляра [QQmlEngine](http://qt-project.org/doc/qt-5.1/qtqml/qqmlengine.html) и загрузки qml файлов вместе с [QQmlComponent](http://qt-project.org/doc/qt-5.1/qtqml/qqmlcomponent.html).

*Далее идут примеры кода, очень понятные и продуктивные, использования qml в С++ приложении. Также есть супер полезная ссылка преобразования данных между qml и С++.* При написании приложения С++/qml полезно загружать qml файлы через систему ресурсов, которая встраивает их в исполняемый файл. Это позволяет обращаться к ним через систему ресурсов, а не через абсолютные или относительные пути. Интересно было бы поэкспериментировать с этим. Для использования данной возможности следует создать .qrc файл коллекции ресурсов, который перечисляет файлы ресурсов в xml формате. Из С++ необходимо загрузить Главный qml файл как ресурс с использованием префикса :/ или как url при помощи qrc схемы. При этом сами файлы qml ничего не знают о системе ресурсов и могут использовать всё те же относительные и абсолютные пути. *Есть далее блестящий пример кода использования данной возможности.*

## ЗАГРУЗКА РЕСУРСОВ И СЕТЕВАЯ ПРОЗРАЧНОСТЬ

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-documents-networktransparency.html>

qml поддерживает сетевую прозрачность при помощи использования url для всех ссылок из qml документа на другой контент. Поэтому qml может обрабатывать также удалённые ресурсы вместо локальных. *Есть примеры кода. Есть информация о встройке ресурсов. Но я пока её не понял.* Также вам не следует загружать в qml ресурсы, которым вы не доверяете. Это то же самое, как загрузить данные ресурсы из любого другого приложения. *Есть пример кода.*

## РАЗРЕШЕНИЕ ОБЛАСТИ ВИДИМОСТИ И ИМЁН

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-documents-scope.html>

область видимости управляет тем, к каким переменным выражение имеет доступ, и какие переменные имеют приоритет при конфликте имён. Qml обеспечивает java script встроенный механизм областей видимости. Qml расширение области видимости не вмешивается в java script область видимости.

Каждая java script функция, выражение или файл имеют собственный уникальный объект-переменную. Локальные переменные, объявленные в одном, никогда не конфликтуют с локальными переменными, объявленными в другом. Можно использовать функции из импортируемых java script файлов, если они туда встроены. *Есть пример кода.*

Связывание свойств является наиболее общим способом использования java script в qml. Связывание свойств связывает результат некоторого java script выражения с некоторым свойством объекта. Объект, которому принадлежит связанное свойство, известен как объект области связывания.

Присоединённые свойства существуют во всех объектах, даже если они необходимы только для их части. Следовательно, присоединённые свойства без квалификатора интерпретируются как присоединённые свойства к объекту области видимости, что не всегда является тем, что задумал программист.

Каждый qml компонент в qml документе определяет логическую область видимости. Каждый документ имеет, по крайней мере, один корневой компонент, но также может иметь другие встроенные под-компоненты. Область видимости компонента – это объединение идентификаторов объектов внутри компонента и свойства корневого объекта компонента. Идентификаторы всегда определяются явно qml программистом, так что они всегда имеют преимущество перед другими именами свойств. В qml экземпляры компонентов соединяют вместе их области видимости, образуя иерархию областей видимости. Экземпляры компонента могут напрямую получить доступ к областям видимости их предков. Например, если в некотором компоненте определён другой компонент, то внутри последнего можно использовать названия свойств предыдущего, и они будут соответствовать свойствам родителя. Если же во вложенном компоненте есть свойство с идентичным именем, то оно перекроет свойство родителя. Qml является языком динамического связывания. В зависимости от того, где используется данный файл, его содержимое может интерпретироваться по-разному.

Динамическое определение областей видимости является очень мощным механизмом, но оно должно использоваться осторожно, чтобы предостеречь поведение qml кода от сложности предсказания. В общем, его следует использовать только в том случае, когда два компонента уже тесно связаны другим способом. При построении повторно используемых компонентов, желательно использовать интерфейсы свойств. *Есть пример кода.*

Qml разрешает переписывать имена свойств, определённых в одном объявлении объекта именами свойств, объявленными в другом объявлении объекта, который расширяет первый. *Есть пример кода, который демонстрирует данное обстоятельство.* Qml не разрешает типы, имена свойств и идентификаторов, которые конфликтуют со свойствами Глобального объекта, чтобы предотвратить любую путаницу. Программисты должны быть уверены, что математические функции всегда работают, как ожидается.

## СОГЛАШЕНИЯ КОДИРОВАНИЯ QML

[*http://qt-project.org/doc/qt-5.1/qtdoc/qml-codingconventions.html*](http://qt-project.org/doc/qt-5.1/qtdoc/qml-codingconventions.html)

в данном документе содержатся соглашения использования qml, которые рекомендуется использовать всем остальным пользователям данной библиотеки.

1. Порядок следования атрибутов объекта: идентификатор, объявления свойств, сигналы, java script функции, свойства объекта, дочерние объекты, состояния, переходы.
2. Для лучшей читабельности данные части разделяются при помощи пустой линии. *Есть пример кода.*
3. Если вы используете много свойств из некоторой группы свойств, то рассмотрите использование предлагаемого в данной части интерфейса. *Есть пример кода.*
4. Если список содержит только один элемент, то квадратные скобки опускаются. *Есть пример кода.*
5. Если скрипт – это одно выражение, то его рекомендуется писать в линию. Если скрипт – это пара строк, то обычно используется блок. Если скрипт содержит более одной пары строк или может быть использован разными объектами, то мы рекомендуем создание функции и вызова её. *На все данные случаи есть примеры кода.* Для больших функций мы будем помещать их в отдельный java script файл и импортировать. *Есть пример кода.*

## СЛОВАРЬ ТЕРМИНОВ QML

<http://qt-project.org/doc/qt-5.1/qtdoc/qml-glossary.html>

Qml – это язык, на котором пишутся qml приложения. Архитектура языка и движок реализуются при помощи модуля qml. Qt Quick – стандартная библиотека типов и функциональности для qml языка, которая обеспечивается модулем Qt Quick и может быть доступна при помощи импортирования. Тип – или базовый тип или объектный тип. Qml язык обеспечивает много встроенных базовых типов, а Qt Quick модуль обеспечивает различные типы для построения qml приложений. Типы также могут быть обеспечены сторонними разработчиками через модули или разработчиками приложения через qml документы. Базовый тип – может быть инстанцирован движком qml. Может быть определён или при помощи документа в qml файле или при помощи свойств класса QObject С++. Объект – это экземпляр объектного типа qml. Они создаются движком, когда он выполняет объявление объекта. Также есть некоторые функции для динамического создания объекта во время выполнения. Компонент – шаблон, из которого qml объект или дерево объектов создаётся. Он создаётся, когда документ загружается при помощи qml движка. После его загрузки он может быть использован для инстанцирования объекта или дерева объектов, которые он представляет. Также это специальный тип, который можно использовать для встроенного создания компонентов внутри документа. Qml документ является самодостаточным кусочком исходного кода qml, который начинается с одного или более утверждений импорта и содержит одно объявление объекта высокого уровня. Документ может существовать внутри qml файла или внутри текстовой строки. Название qml файла должно начинаться с большой буквы, чтобы его распознал движок. Свойство – это атрибут объекта, который имеет имя и связанное значение, которое может быть считано внешне. Связывание – это java script выражение, которое привязано к свойству. Значение свойства в любой момент времени – это значение, которое возвращает данное выражение. Сигнал – уведомление от qml объекта. Большинство свойств имеют встроенные сигналы и обработчики сигналов. Обработчик сигналов – выражение, которое инициируется сигналом. Медленное инстанцирование – экземпляры объектов могут быть инстанцированы медленно во время выполнения, чтобы избежать выполнения ненужной работы, пока она не требуется. QtQuick обеспечивает тип Loader для проведения медленного инстанцирования более удобно.

## ВИЗУАЛЬНЫЕ ЭЛЕМЕНТЫ В QML

<http://qt-project.org/doc/qt-5.1/qtdoc/qtquick-usecase-visual.html>

Все визуальные типы Qt Quick основаны на типе Item, который обеспечивает общий набор атрибутов для визуальных элементов, включая прозрачность и атрибуты преобразований. Для большинства базовых визуальных элементов Qt Quick обеспечивает тип прямоугольника, чтобы рисовать прямоугольники. Для рисования частных форм за прямоугольником смотрите тип Канва, а для отображения изображений смотрите класс имэдж. *Есть пример кода создания и использования прямоугольника.* Тип изображения может быть использован для отображения изображений. Изображение имеет ресурс. *Есть пример кода использования данного элемента.* Прозрачность можно анимировать. *Есть пример кода использования.* Преобразования также можно анимировать. Для более сложных трансформаций следует смотреть свойство [Item::transform](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-item.html#transform-prop).

### ПОДДЕРЖИВАЕМЫЕ ТИПЫ ВВОДА ПОЛЬЗОВАТЕЛЯ

<http://qt-project.org/doc/qt-5.1/qtdoc/qtquick-usecase-userinput.html>

Отклик на ввод пользователя в qml. Qt Quick обеспечивает поддержку для большинства общих типов ввода пользователя, включая мышь и события прикосновений, текстовый ввод и события нажатия клавиш. Другие модели обеспечивают поддержку для других типов ввода пользователя (Qt Sensors обеспечивают поддержку сотрясений). Тип MouseArea позволяет обрабатывать события мыши и прикосновений. Данный тип можно объединить вместе с изображением, прямоугольником или текстом для получения простой кнопки. *Есть пример кода.* Есть типы для обработки событий от касаний. Присоединённое свойство Keys позволяет обрабатывать события клавиатуры и кнопки. Данное свойство доступно для всех типов, происходящих от Item. Отклик на события осуществляется при помощи свойства focus. *Есть пример кода.* Также Qt Quick обеспечивает несколько типов для ввода текста.

### АНИМАЦИИ В QML

<http://qt-project.org/doc/qt-5.1/qtdoc/qtquick-usecase-animations.html>

Qt Quick позволяет анимировать свойства. Qml был разработан для способствования созданию плавных интерфейсов пользователей. Это такие интерфейсы, в которых компоненты анимируются вместо резкого появления, исчезновения и перемещения. Qt Quick позволяет вам объявлять различные состояния графического интерфейса пользователя в объектах State. Эти состояния включены в изменения свойств и могут быть полезны для организации вашей логики графического интерфейса пользователя. Переходы – это объекты, которые вы можете связывать с элементом, чтобы определить, как его свойства будут анимироваться, когда они изменяются благодаря изменению состояний. Состояния и переходы для элемента могут быть объявлены при помощи свойств [Item::states](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-item.html#states-prop) and [Item::transitions](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-item.html#transitions-prop). *Есть пример использования состояний и переходов.* Поведения используются для задания анимации во время изменения свойства. *Есть пример кода. Нужно будет его исследовать.* Вы также можете создавать более общие анимации, не привязанные к свойству или к состоянию. *Есть пример кода, который в будущем следует изучить.*

### ОТОБРАЖЕНИЕ ТЕКСТА В QML

<http://qt-project.org/doc/qt-5.1/qtdoc/qtquick-usecase-text.html>

Используйте элемент Text. Можно задавать различные свойства самого текста, а также различные свойства рамки текста, а также можно некоторые фрагменты текста делать форматированными. Также текст можно позиционировать различными способами. *Есть пример кода, который я не изучал.*

### ПОЗИЦИОНИРОВАНИЕ И МАКЕТЫ В QML

<http://qt-project.org/doc/qt-5.1/qtdoc/qtquick-usecase-layouts.html>

Есть несколько способов позиционирования элементов в qml.

* Ручное позиционирование состоит в том, что вы вручную задаёте координаты на экране. *Есть пример кода.*
* Якоря позволяют один элемент соединить с другим элементом. Есть 6 линий, к которым можно прицепить якоря. *Есть пример кода.*
* Позиционеры и макеты используются при желании разместить периодически несколько элементов. Qt Quick определяет несколько типов для этих целей. *Есть пример кода.*

### ПОДДЕРЖКА СТИЛЕЙ И ТЕМ

<http://qt-project.org/doc/qt-5.1/qtdoc/qtquick-usecase-styling.html>

Типы, обеспечиваемый Qt Quick, не являются полными компонентами пользовательского интерфейса сами по себе. Общим прецедентом является разработка набора частных стилизованных компонентов пользовательского интерфейса вне типов модуля Qt Quick. Это легко выполняется при помощи создания ваших собственных повторно используемых компонентов. При данном подходе вы определяете ваш собственный тип с внешностью, которую вы желаете иметь в вашем приложении и стилизуете тип напрямую. Затем вы используете данный тип в вашем приложении вместо нестилизованного типа. *Есть пример кода, но я его пока так и не понял. Придётся прочитать побольше. Теперь понял. Мне сложно запомнить, что название файла и есть наименование объекта. Есть также пример кода для создания стилизованных кнопок.*

### ОБЗОР ЭЛЕМЕНТОВ УПРАВЛЕНИЯ QT QUICK

[*http://qt-project.org/doc/qt-5.1/qtquickcontrols/qtquickcontrols-overview.html*](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qtquickcontrols-overview.html)

Qt Quick элементы управления обеспечивают набор элементов управления для создания интерфейсов пользователя в Qt Quick. Хотя мы традиционно использовали [QQuickView](http://qt-project.org/doc/qt-5.1/qtquick/qquickview.html), чтобы отображать qml файлы в С++ приложениях, такой подход обозначает, что вы можете только установить свойства окна из С++.

При помощи Qt Quick элементов управления вы можете объявить [ApplicationWindow](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-applicationwindow.html) как корневой элемент вашего приложения и запустить его при помощи использования [QQmlApplicationEngine](http://qt-project.org/doc/qt-5.1/qtqml/qqmlapplicationengine.html). Это гарантирует, что вы можете управлять свойствами окна верхнего уровня из qml. *Есть пример кода использования приложения qml в коде С++.*

### ИСПОЛЬЗОВАНИЕ СИСТЕМЫ ЧАСТИЦ QML

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-effects-particles.html>

система частиц содержит 4 главных типов qml:

* [ParticleSystem](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick-particles2-particlesystem.html),
* Painters,
* Emitters and
* Affectors.

Тип [ParticleSystem](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick-particles2-particlesystem.html) связывает другие типы вместе и управляет ими. Painters, Emitters and Affectors должны все иметь одинаковую [ParticleSystem](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick-particles2-particlesystem.html), чтобы взаимодействовать друг с другом. Каждая частица имеет логическое представление внутри системы частиц. Каждая частица является членом группы частиц и каждая группа имеет свой идентификатор. Группы используются в двух целях: для управления частиц и из-за того, что они могут иметь стохастические переходы состояний. Частицы могут изменять группы динамически. Эмиттеры эмитируют логические частицы в систему. Эти частицы имеют траекторию и продолжительность жизни, но они не имеют визуализации. Эти частицы испускаются из локации эмиттера. Рисовальщики являются типами, которые визуализируют логические частицы. Аффекторы являются необязательными компонентами системы частиц. Они могут выполнять множество манипуляций над симуляцией, таких как изменение траектории частиц или заканчивая принудительно их жизнь. Из-за соображений производительности не рекомендуется использовать для систем с большим объёмом частиц. Система частиц предоставляет выгоду от стохастического контроля параметров среди большого числа экземпляров. Направления могут быть определены при помощи угла и амплитуды. Система частиц содержит несколько типов, которые представляют из себя формы.

### ГРАФИЧЕСКИЕ ЭФФЕКТЫ

<http://qt-project.org/doc/qt-5.1/qtgraphicaleffects/graphicaleffects.html>

эффекты – это визуальные элементы, которые могут быть добавлены в Qt Quick как графические компоненты интерфейса пользователя. *Показано, как импортировать типы графических эффектов. Также в данной части перечислены многие эффекты.*

## РЕСУРСЫ РАЗРАБОТКИ QML ПРИЛОЖЕНИЙ

<http://qt-project.org/doc/qt-5.1/qtdoc/qtquick-applicationdevelopers.html>

*далее идут всяческие вещи для отображения веб содержания, что пока мне не нужно. Далее идут модули для обработки ввода пользователя, характерного для мобильных устройств. Далее идёт обзор частей для программирования мультимедиа.*

## ВОПРОСЫ ПРОИЗВОДИТЕЛЬНОСТИ И ПРЕДЛОЖЕНИЯ

<http://qt-project.org/doc/qt-5.1/qtdoc/qtquick-performance.html>

как разработчик приложения вы стараетесь разрешить рисование с достижением скорости обновления 60 кадров в секунду. Такая скорость означает, что между каждым фреймом должно проходить около 16 мс, за которые обработка должна быть сделана, что включает обработку, требуемую для загрузки примитивов рисования на графическое железо. На практике это означает, что разработчик приложения должен:

* использовать асинхронное, управляемое событиями программирование, когда только это возможно;
* использовать рабочие потоки для выполнения значительного количества обработки;
* никогда вручную не прокручивать цикл обработки событий;
* никогда не тратить более пары миллисекунд на фрейм внутри функций блокировки.

Несоответствие данному факту приводит к зависающим фреймам, которые имеют сильное влияние на удобство пользования. *Нельзя никогда вмешиваться в цикл обработки событий вручную. есть объяснения в данной части.*

Наиболее важно – это использовать qml профайлер, включённый в qt Qt Creator. **Знание о том, где затрачивается время в вашем приложении, позволит вам сфокусироваться на проблемных областях, которые действительно существуют, чем на проблемных областях, которые потенциально существуют.** *В документации настоятельно рекомендуется использовать данный отладчик. Также есть пару ценных советов о том, как проводить отладку в данном случае.*

Большинство qml приложений будет иметь большое количество java script кода в них, в форме динамических функций, обработчиков сигналов и выражений для связывания свойств. В qml это оптимизировано и в некоторых случаях может быть быстрее, чем С++. Но при этом следует быть внимательным, чтобы ненужная работа не включилась неожиданно. В qml есть два типа связывания:

* оптимизированное и
* неоптимизированное.

Это работает так: движок qml способен детектировать простые выражения и выполнять их без проникновения в полноценную среду java script. Для этого делайте выражения как можно более простыми. С этой целью лучше, чтобы информация о типе каждого элемента выражения была известна во время компиляции. *Далее есть замечания по поводу того, как провести оптимизацию.* Также говорится, что переменная среды QML\_COMPILER\_STATS может быть установлена при использовании qml приложения, чтобы печатать статистику о том, как много связываний было оптимизировано. *Далее приводится список того, что желательно включать в связывание, что находится в области видимости оптимизации.*

Одним большим преимуществом использования java script является то, что в большинстве случаев, когда осуществляется доступ к свойству qml типа, java script объект с внешними ресурсами, содержащими и С++ данные, создаётся. В большинстве случаев это достаточно недорого, но в других случаях это может быть достаточно дорого. *Далее рассматривается аспект преобразования С++ списка в java script список, на основании чего делается вывод о том, насколько это может быть дорого.* Решение свойств занимает время. Пока в некоторых случаях результат обзора может быть скэширован и повторно использован, всегда лучше избегать производства ненужной работы вовсе, если это возможно. *Есть пример кода, который понятно демонстрирует, что имеется в виду.* Выражение связывания свойств будет переоценено, если любое из свойств, на которые оно ссылается, изменяется. Поэтому выражения для связывания должны быть как можно проще. *Есть пример кода того, как можно упростить оценку свойства.* Некоторые С++ контейнеры работают в qml быстро (*перечислены*), тогда как другие работают медленно. Используйте всегда, где это возможно, первый тип контейнера. При работе с контейнерами обычно создаётся java script копия данных контейнеров, а затем эта копия присваивается С++ контейнеру, так как это работает гораздо быстрее. *Есть прекрасные примеры кода, которые демонстрируют правильную и неправильную запись информации в контейнеры.* Также при связывании свойства со списком лучше избежать этого, так как свойство может обновляться тогда, когда этого совсем не нужно. *Есть замечательные примеры кода, демонстриирующие это, а также то, как с этим бороться.* Всё, что касается списков, также касается и свойств типа шрифтов, цветов и т.д.

Каждый движок java script проводит некоторую собственную оптимизацию, но он проводит её только в том случае, если выполняются некоторые критерии. Что касается движка в Qt Quick 2, то здесь следует избегать использования функции eval(), а также не следует удалять свойства объектов.

Расчёт макета текста может быть медленной операцией. Поэтому используйте, когда это возможно, неформатированный текст. Это уменьшает количество работы, требуемой для движка макетов. Вам следует с форматированным текстом использовать только AutoText.

Изображения также создают немалые проблемы с точки зрения оптимизации. Часто изображения достаточно большие и поэтому важно обеспечить то, чтобы их загрузка не блокировала поток интерфейса пользователя. Установите свойство асинхронности на qml элемент изображения, чтобы предоставить возможность асинхронной загрузки изображений из локальной файловой системы. Тогда всё будет загружаться в рабочий поток меньшего приоритета. Если ваше приложение загружает большое изображение, но изображает его в малой области, то используйте свойство sourceSize, чтобы держать в памяти меньшее изображение. Также избегайте композиции изображений во время выполнения, так как это может уменьшить производительность. Это касается, например, эффектов затенения.

Более эффективно использовать якоря, чем связывание с позициями элементов друг относительно друга. *Есть хороший пример кода.* Но связывание предоставляет большую гибкость. Если ваш макет предполагается быть статическим, то вам следует задавать координаты также статически.

Очень часто возникает желание написать вашу собственную С++ модель, которая бы использовалась совместно с qml изображением. Вот общие рекомендации по данному вопросу:

* Будьте настолько асинхронными, насколько это возможно.
* Делайте всю обработку в рабочем потоке меньшего приоритета.
* Группируйте серверные операции так, чтобы ввод/вывод и IPC были минимизированы.
* Используйте sliding slice window для кэширования результатов, чьи параметры определяются при помощи профилирования.

Также избегайте ненужной блокировки, так как блокировка может существенно ухудшить производительность.

Qml обеспечивает ListModel элемент, который можно использовать для занесения данных в изображение списка. *Также говорится о создании рабочего потока java script, но я пока этого не понимаю. Следует разобраться с данным вопросом.* В Qt Quick 2 модель списка очень производительна, но только если роли элементов не изменяются динамически. Делегаты представлений следует делать настолько простыми, насколько это возможно. Не нужно создавать ту информацию, которая в данный момент времени не очень нужна. *Далее приводится список советов по созданию делегатов.* Qml содержит некоторые особенности, позволяющие создавать очень привлекательные интерфейсы пользователя. Но это требует затрат некоторой производительности. Избегайте запуска java script во время анимации. Не надо использовать java script функции сложной структуры для некоторых параметров во время анимации. Особенно внимательными следует быть со скриптовыми анимациями, так как последние используются в Главном рабочем потоке.

В Qt Quick 2 партиклс модуле можно создавать красивые эффекты частиц, которые могут быть плавно интегрированы в интерфейс пользователя. Но платформы отличаются по возможностям. Чем больше частиц вы используете, тем быстрее должно быть ваше оборудование, чтобы обеспечить 60 FPS. Также требуется более быстрый ЦП. Поэтому нужно данные эффекты протестировать на вашей платформе, чтобы откалибровать число частиц. Также данную систему можно сделать недействительной, например, чтобы не тратить ненужную работу для тех элементов, которые сейчас не видны.

При помощи разделения приложения на простые модульные компоненты, каждый из которых содержится в отдельном qml файле, вы можете достигнуть более быстрой загрузки приложения и лучшего контроля над использованием памяти, а также уменьшить число активных, но невидимых элементов в вашем приложении.

Наиболее простым способом ускорения работы при загрузке является избегание произведения работы, которую вам не нужно делать, а также задерживая выполнение работы до тех пор, пока это вам необходимо. Это может быть сделано при помощи использования Loader или создания компонентов динамически. *Далее описывается использование загрузчика. Далее ведётся речь о динамическом создании объектов, о котором речь велась также и немного ранее.* В большинстве случаев необходимо все элементы, которые невидимы, инициализировать медленно, а удалять тогда, когда они больше не нужны. Но иногда требуется сделать элемент просто невидимым. Qml позволяет делать данные операции.

Есть несколько вещей, которые могут существенно уменьшать скорость рисования и разработчикам следует их избегать. Усечение недоступно по умолчанию и должно использоваться, только если оно очень необходимо. Это визуальный эффект, но не оптимизация. Оно увеличивает сложность рисования. Особенно плохим является усечение внутри делегата, которого следует всегда избегать. Если есть элементы, закрытые другими элементами, то лучше установить их видимость в ложь. Это же касается элементов, которые невидимы, но их необходимо инициализировать во время загрузки. Непрозрачное содержание гораздо быстрее рисовать, чем прозрачное. Это связано с дополнительными операциями для обеспечения прозрачности. ShaderEffect позволяет разместить OpenGL код прямо в Qt Quick приложении с очень малым переполнением. Но при этом следует иметь в виду, что фрагмент кода будет выполняться для каждого пиксела и поэтому для плохого оборудования следует применять особенные инструкции для данного объекта. С использованием данных объектов следует быть осторожным, так как они могут быть немного дорогими.

Количество памяти, которая будет выделена приложением и способ, при помощи которого память будет выделена, очень важны. Кроме всего прочего динамическое выделение памяти в яме также является достаточно затратной расчётной операцией. Java script владеет управляемой ямой памяти, которая снабжена автоматическим сборщиком мусора, что обеспечивает некоторые преимущества, но также и вызывает некоторые последствия. Qml приложение использует память как из ямы С++, так и из управляемой ямы java script. Разработчик приложения должен быть внимательным в тонкостях каждой ямы, чтобы максимизировать производительность. Следующие замечания не являются критическими, но выступают как советы, которые могут оказаться неоправданными в некоторых случаях.

* Если ваше приложение состоит из множественных представлений, но только одно требуется в единицу времени, вы можете использовать медленное инстанцирование, чтобы минимизировать количество памяти, которая вам необходима для ваших потребностей в выделении памяти в любое время.
* Часто лучше вручную уничтожить объекты, нежели ожидать их уничтожения при помощи сборщика мусора.
* Не следует вручную вызывать сборщик мусора, так как он может заблокировать графический поток на длительное время.
* Избегайте сложного связывания свойств.
* Также используйте существующие элементы. Не нужно создавать копии уже существующих элементов.
* Используйте типы одиночки вместо скриптов pragma библиотек.

Использование памяти в qml может быть разделено двумя способами:

* использование ямы С++
* и ямы java script.

*Далее перечислено, что в какой яме хранится. Далее сообщаются некоторые тонкости использования java script ямы, особенно для мобильных приложений.*

Если элемент не определяет некоторых частных свойств, то он считается движками java script и qml явно заданным, в противном случае он считается неявным, т.е. имеющим некоторые свои собственные внутренние свойства. *Есть прекрасный пример кода с пояснениями, который объясняет данный важный аспект.*

Всякий раз при принятии решения касательно компромисса между выделением памяти и производительностью, важно держать в уме вклад производительности кэша ЦПУ, системы страниц ОС и коллекции мусора java script. Лучше выбрать наиболее эффективный из них. Для этого следует проводить экспериментальное тестирование.

Нужно понимать фундаментальные принципы работы компьютерной техники, а также иметь практические знания деталей реализации для некоторой заданной платформы.

Фрагментация – это элемент разработки на С++. Если разработчик не определяет никаких типов или плагинов С++, то данная секция может быть им проигнорирована. Что такое фрагментация. Если в приложении храниться, обрабатывается и удаляется за ненадобностью много данных, то может так оказаться, что целые несмежные куски памяти не используются ОС. Фрагментации можно избежать

* при помощи применения пуловых аллокаторов,
* при помощи уменьшения количества памяти, которая выделяется в единицу времени,
* при помощи внимательного управления временами жизни объектов,
* при помощи периодического очищения и восстановления кэшей,
* или при помощи применения сборщика мусора.

Java script обеспечивает сборщик мусора. Память в яме java script, в отличие от С++, принадлежит java script движку. Движок периодически собирает все данные без ссылок в яме java script, и если фрагментация становится проблемой, то движок делает яму более компактной при помощи удаления живых данных в смежную область, возвращая свободную память ОС. С одной стороны сборщик мусора уменьшает фрагментацию, но с другой стороны он предусматривает выполнение java script движком потенциально долгих операций, на время выполнения которых теряется контроль над приложением. Если сборщик мусора будет автоматически инициироваться java script движком, когда количество доступной свободной памяти мало, то иногда лучше, если разработчик приложения примет решение о том, когда запросить сборщик мусора вручную. но обычно этого не следует делать. *Далее описываются ситуации, в которых важно этим заниматься, а также показано, как вызывать сборщик мусора вручную в java script.*

*В конце есть ссылки на две статьи, посвящённые проблеме компромисса между затратами памяти и производительности. Это интернет ресурсы, которые могут оказаться очень полезными для меня.*

## ИНТЕРНАЦИОНАЛИЗАЦИЯ И ЛОКАЛИЗАЦИЯ ПРИ ПОМОЩИ QT QUICK

<http://qt-project.org/doc/qt-5.1/qtdoc/qtquick-internationalization.html>

в данной секции описываются различные аспекты интернационализации вашего исходного кода.

1. используйте qsTr() для всех строк интерфейса пользователя. *Есть пример кода, а также перечислены все средства для осуществления перевода слов.*

2. добавляйте некоторый контекст для переводчика. Человеку, который будет переводить текст, важно понимать контекст текста, чтобы его работа была сделана качественнее и быстрее. Это делается при помощи экстра дескрипторов описания текста, которые далее включаются в .ts файлы перевода. Данные файлы являются xml файлами, а затем конвертируются в двоичные файлы и включаются в конечное приложение. *Есть пример кода.*

3. устраняйте неоднозначность текста. Часто текст одинаков, но имеет различное значение. Для данных случаев следует указывать системе перевода, чтобы она создала два отдельных перевода. Для этого обеспечивается определённый идентификатор, который задаётся также некоторым экстра описанием. *Есть пример кода.*

4. используйте %x для вставки параметров в строку. *Есть пример кода.*

5. используйте %Lx чтобы локализовать числа. *Есть пример кода.*

6. интернационализированные даты, времена и валюты. Для интернационализации даты нужно запрашивать специальные методы. *Есть пример кода.*

7. используйте QT\_TR\_NOOP() для переводимых строковых данных. Если пользователь изменяет систему языка без перезагрузки, то строки не будут обновляться автоматически. Для того, чтобы сделать их обновляемыми, их необходимо задекларировать при помощи данного макроса. *Есть пример кода, но по нему у меня возникают некоторые вопросы.*

8. используйте локацию для расширения особенностей локализации. Если вы желаете разную графику, или звуки для разных мест, вы можете использовать функцию Qt.[locale](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-locale.html)(), чтобы получить текущую локацию. Далее вы выбираете подходящую графику или звуки для данной локации. *Есть очень замечательный пример кода, демонстрирующий данный приём.* Qt Quick приложение имеет ту же систему локализации, что и приложение qt С++. Также можно в одном и том же приложении использовать совместно строки в исходном коде qml и С++. Система создаст единственный общий файл перевода, доступный и из qml, и из С++.

lupdate считывает файл проекта, чтобы идентифицировать, какие файлы ресурсов содержат тексты для перевода. При помощи переменных SOURCES или HEADERS. Если qml файлы разместить в данных переменных, то С++ подумает, что они являются его файлами. Чтобы этого избежать, следует использовать некоторое условное выражение в файле проекта. *В данной части есть пример кода, демонстрирующий данное обстоятельство.*

## QT QML ТИПЫ

<http://qt-project.org/doc/qt-5.1/qtqml/qmlmodule-qtqml2-qtqml-2.html>

qml модуль обеспечивает определение и реализацию различных удобных типов, которые могут быть использованы вместе с языком qml. [QtObject](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-qtobject.html) and [Component](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-component.html) типы объектов не являются визуальными и обеспечивают строительные блоки для расширений qml. *Далее показано, как производить импорт. Далее перечислены базовые типы qml, а также объектные типы qml. Рассмотрим некоторые, наиболее важные или непонятные из них.*

### [Component](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-component.html)

<http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-component.html>

[Component](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-component.html) инкапсулирует определение qml компонента. Компонент инстанцирует С++ класс [QQmlComponent](http://qt-project.org/doc/qt-5.1/qtqml/qqmlcomponent.html). Компоненты – это повторно используемые qml типы с хорошо определёнными интерфейсами. Компоненты часто определяются типами компонентов: qml файлами. Также можно создавать компоненты внутри файла. Это может быть полезным для повторного использования маленьких компонентов вместе с qml файлом или для определения компонента, который логически принадлежит вместе с другими qml компонентами файлу. Если внутри компонента определены некоторые графические элементы, то они не отобразятся, пока компонент не будет загружен при помощи Loader. Данный тип обычно используется для обеспечения графических компонентов для представлений. Также с использованием [Qt.createComponent()](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-qt.html#createComponent-method) компоненты могут создаваться динамически. Контекст создания компонента соответствует контексту, где компонент был объявлен. Этот контекст используется как родительский контекст, когда компонент инстанцируется объектом, таким как модель списка или загрузчик. *Есть пример кода.*

### QtObject

<http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-qtobject.html>

QtObject является базовым qml типом. Он инстанцирует объект [QObject](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html). Это невизуальный элемент, который содержит только имя. Он может быть использован, если вам необходим легковесный тип, чтобы окружить набор частных свойств. Он также может быть полезен для интеграции с С++, так как он является только простым QObject.

### Binding

<http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-binding.html>

Binding предоставляет возможность произвольного создания связываний свойств. Иногда необходимо связать со свойством объект, который не был напрямую инстанцирован qml, в общем, это касается свойства класса, экспортированного qml от С++. В этих случаях, регулярное связывание свойств не работает. Связывание в данном случае позволяет вам связать любое значение с любым свойством. *Есть пример использования данного типа. Довольно понятный.* В некоторых случаях вы желаете контролировать значение свойства, только когда определённое условие является истиной. Это обычно невозможно сделать при помощи обычного связывания, так как вам необходимо обеспечить значения для всех возможных веток. *Есть пример использования.*

### Connections

<http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-connections.html>

Connections описывает соединения к сигналам. Данный объект создаёт соединение с сигналом. Иногда недостаточно просто определить обычный обработчик сигнала: если требуются множественные соединения с сигналом; необходимо создавать соединения вне области видимости отсылателя сигнала; соединение с целями не определено в qml. Тогда используется данный тип. *Есть пример кода.*

### Instantiator

<http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-instantiator.html>

Instantiator динамически создаёт объекты. Он используется для управления динамическим созданием объектов, или чтобы динамически создавать много объектов, исходя из шаблона. Данный элемент будет управлять объектами, которые он создал. Эти объекты могут быть удалены данным типом.

### Timer

<http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-timer.html>

Timer переключает обработчик в определённый интервал. Данный тип таймера синхронизирован с типом таймера анимации.

### Qt объект

<http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-qt.html#details>

Qt объект является Глобальным объектом с утилитами, свойствами и перечислениями. Он не инстанцируется; чтобы его использовать, вызовите члены Глобального объекта напрямую. *Есть пример кода.*

## Qt Quick

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-qmltypereference.html>

Qt Quick модуль обеспечивает разнообразие qml типов для создания интерфейсов пользователя и приложений при помощи qml. Эта страница перечисляет каждый qml тип, обеспечиваемый данным модулем, которые организованы согласно категориям и целям. Qt Quick включает несколько подмодулей, которые содержат дополнительные типы. Это [XML List Model](http://qt-project.org/doc/qt-5.1/qtquick/qmlmodule-qtquick-xmllistmodel2-qtquick-xmllistmodel-2.html), [Local Storage](http://qt-project.org/doc/qt-5.1/qtquick/qmlmodule-qtquick-localstorage2-qtquick-localstorage-2.html), [Particles](http://qt-project.org/doc/qt-5.1/qtquick/qmlmodule-qtquick-particles2-qtquick-particles-2.html), [Window](http://qt-project.org/doc/qt-5.1/qtquick/qmlmodule-qtquick-window2-qtquick-window-2.html), [Dialogs](http://qt-project.org/doc/qt-5.1/qtquickdialogs/qmlmodule-qtquick-dialogs1-qtquick-dialogs-1.html), [Controls](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qmlmodule-qtquick-controls1-qtquick-controls-1-0.html), [Layouts](http://qt-project.org/doc/qt-5.1/qtquicklayouts/qmlmodule-qtquick-layouts1-qtquick-layouts-1.html). *Далее перечислены некоторый дополнительные базовые типы, которые добавлены ко встроенным qml базовым типам.* Все объектные типы в Qt Quick наследуют Item, который наследует в свою очередь [QtObject](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-qtobject.html). Qml объектные типы также доступны при импорте данного модуля. *Далее перечислены объектные типы в Qt Quick. Рассмотрим типы, обеспечиваемые Qt Quick.*

### Типы окон и экранов

<http://qt-project.org/doc/qt-5.1/qtquick/qmlmodule-qtquick-window2-qtquick-window-2.html>

qml модуль содержит типы для создания высокоуровневых окон и доступа к экранной информации. *Показан импорт данного модуля.*

#### Screen

<http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick-window2-screen.html>

Screen присоединённый объект обеспечивает информацию об экране или об окне, которое изображено на нём. Он доступен внутри Item или производного от него типа, после завершения компонента. Внутри элемента он указывает на экран, на котором в настоящее время изображён элемент. Он также доступен внутри Window или производных от него типов. *Есть пока непонятное замечание относительно систем с многими экранами.* Заметьте, что данный тип недействителен при Component.onCompleted, потому что элемент или окно к тому времени ещё не изображены.

#### Window

<http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick-window2-window.html>

Window создаёт новое окно высокого уровня. Данный тип инстанцирует С++ класс QQuickWindow. Он создаёт окно для Qt Quick экрана. Если не импортировать данный модуль, то это позволит иметь qml среду без доступа к оконной системе. Окно может быть объявлено внутри элемента или другого окна. *Также в данной части указана некоторая другая информация касательно окон.*

#### CloseEvent

[*http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick-window2-closeevent.html*](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick-window2-closeevent.html)

CloseEvent уведомляет, что окно скоро закроется. Содержит свойство, которое может отменить закрытие окна.

### Типы моделей данных

<http://qt-project.org/doc/qt-5.1/qtquick/qmlmodule-qtquick-xmllistmodel2-qtquick-xmllistmodel-2.html>

qt qml модуль содержит типы для создания моделей из xml данных. *Далее показан импорт.*

#### XmlRole

<http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick-xmllistmodel2-xmlrole.html>

XmlRole используется для определения роли для [XmlListModel](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick-xmllistmodel2-xmllistmodel.html).

#### XmlListModel

<http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick-xmllistmodel2-xmllistmodel.html>

XmlListModel используется для модели только для чтения с использованием XPath выражений. Используется для создания модели для чтения из xml-данных. Может быть использован в качестве модели для различных представлений. Объекты XmlRole определяют атрибуты элемента модели. Данные данной модели загружаются асинхронно. Некоторые роли можно определить как ключи, такие, что при перезагрузке модели изменяются данные только для этих ключей. *Есть пример кода.*

#### [FolderListModel](http://qt-project.org/doc/qt-5.1/qtquick/qml-qt-labs-folderlistmodel1-folderlistmodel.html)

<http://qt-project.org/doc/qt-5.1/qtquick/qmlmodule-qt-labs-folderlistmodel1-qt-labs-folderlistmodel-1-0.html>

[FolderListModel](http://qt-project.org/doc/qt-5.1/qtquick/qml-qt-labs-folderlistmodel1-folderlistmodel.html) предоставляет модель содержания файловой системы.

#### SQLite

<http://qt-project.org/doc/qt-5.1/qtquick/qmlmodule-qtquick-localstorage2-qtquick-localstorage-2.html>

Это синглетонный тип для чтения и записи SQLite базы данных. Эти базы данных определяются пользователем и являются специфическими для qml, но также доступны всем qml приложениям. Они сохраняются в поддиректории Databases как SQLite базы данных. Соединения с базой данных автоматически закрываются java script сборщиком мусора. *Есть примеры создания и использования данной базы данных. Есть пример открытия базы данных. Далее перечислены функции, используемые с данной базой данных в qml.*

### Модуль частиц

<http://qt-project.org/doc/qt-5.1/qtquick/qmlmodule-qtquick-particles2-qtquick-particles-2.html>

модуль qml частиц содержит систему частиц для Qt Quick. *Далее показан импорт. Вот систему частиц я не понимаю. Я изучу данный модуль позднее, когда перейду к практическому освоению возможностей qml.*

### Модуль диалогов

<http://qt-project.org/doc/qt-5.1/qtquickdialogs/qmlmodule-qtquick-dialogs1-qtquick-dialogs-1.html>

Модуль диалогов содержит типы для создания и взаимодействия с системой диалогов. *Показан импорт.*

#### ColorDialog

<http://qt-project.org/doc/qt-5.1/qtquickdialogs/qml-qtquick-dialogs1-colordialog.html>

ColorDialog компонент используется для выбора цвета. *Есть пример использования данного типа.*

#### FileDialog

<http://qt-project.org/doc/qt-5.1/qtquickdialogs/qml-qtquick-dialogs1-filedialog.html>

FileDialog используется для выбора файлов из локальной файловой системы. Он используется для выбора пользователем текущих директорий или файлов, а также для создания новых имён файлов. *Далее есть пример кода и детальное описание данного типа.*

### Модуль элементов управления

<http://qt-project.org/doc/qt-5.1/qtquickcontrols/qmlmodule-qtquick-controls1-qtquick-controls-1-0.html>

модуль элементов управления Qt Quick обеспечивает создание компонентов интерфейса пользователя. *Далее показан импорт данного модуля. Изучаем элементы данного модуля.*

#### ApplicationWindow

<http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-applicationwindow.html>

ApplicationWindow обеспечивает высокоуровневое окно приложения. Предоставляет удобства для размещения всяческих меню и строк. *Есть пример кода.*

#### Некоторые типы элементов управления

<http://qt-project.org/doc/qt-5.1/qtquickcontrols/qmlmodule-qtquick-controls1-qtquick-controls-1-0.html>

* [Button](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-button.html) – кнопка с текстом.
* [CheckBox](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-checkbox.html) – флажок с текстом.
* [ComboBox](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-combobox.html) – для списка с прокруткой.
* [GroupBox](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-groupbox.html) – группа с заглавием.
* [Label](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-label.html) – метка.
* [Menu](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-menu.html) – обеспечивает компонент меню для использования в строках меню, контекстных меню и всплывающих меню.
* [MenuBar](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-menubar.html) – горизонтальное меню.
* [ProgressBar](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-progressbar.html)
* [RadioButton](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-radiobutton.html).
* ScrollView – прокручивающееся изображение внутри другого элемента.
* [Slider](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-slider.html) – вертикальный или горизонтальный ползунок.
* [SpinBox](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-spinbox.html) – счётчик.

#### SplitView

<http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-splitview.html>

SplitView – это контроль, который группирует элементы горизонтально или вертикально при помощи перетаскиваемого разделителя между каждым элементом. *Есть пример кода с использованием данного типа.*

#### StackView

<http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-stackview.html>

StackView обеспечивает основанную на стеке модель навигации. Элементы проталкиваются в стек, когда пользователь продвигается Глубже по материалу и извлекаются оттуда, когда он возвращается назад. Обычно данный контроль делают дочерним объектом окна. Он обычно помещается на границах окна, за исключением верхней. Далее стек можно использования для вызова его методов навигации. Элементы, помещённые в стек, имеют присоединённые свойства Stack. *Далее идёт длинное описание данного типа. Его можно будет изучить, если понадобится использовать данный тип. В целом, его назначение мне понятно.*

#### Некоторые типы элементов управления

<http://qt-project.org/doc/qt-5.1/qtquickcontrols/qmlmodule-qtquick-controls1-qtquick-controls-1-0.html>

[StackViewDelegate](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-stackviewdelegate.html), [StatusBar](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-statusbar.html), [TableView](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-tableview.html), [TableViewColumn](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-tableviewcolumn.html).

[Tab](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-tab.html), [TabView](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-tabview.html) – *непонятно, пока*.

[TextArea](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-textarea.html) – отображает много строк редактируемого форматированного текста.

[TextField](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-textfield.html) – одна линия простого редактируемого текста.

[ToolBar](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-toolbar.html), [ToolButton](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-toolbutton.html) – типы для панели инструментов.

[Action](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-action.html) – обеспечивает абстрактный интерфейс пользователя для действия, которое может быть связано с элементами. <http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-action.html#details> – *содержится понятный пример кода использования данного типа*.

[ExclusiveGroup](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-exclusivegroup.html) – обеспечивает способ объявления нескольких выбираемых элементов управления как взаимно исключаемых.

[MenuSeparator](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-menuseparator.html) – разделитель для элементов внутри меню.

[MenuItem](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-menuitem.html) – обеспечивает элемент для добавления в меню или в строку меню.

[Stack](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-stack.html) – обеспечивает присоединённые свойства для элементов, помещённых в стек вью. *Изучу этот стек вью более подробно*.

#### StackView

<http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-stackview.html#details>

StackView применяет основанную на стеке модель навигации, которая может быть использована вместе с набором взаимосвязанных страниц информации. Элементы модели проталкиваются в стек, когда пользователь проходит Глубже в материал, и извлекается оттуда снова, когда он выбирает возврат назад. Использование данной модели в приложении происходит при помощи добавления данной модели в качестве дочки некоторого окна. Стек может быть использован при помощи запроса его методов навигации. Первый элемент, который показывается в стеке, загружается с присваиванием его к [initialItem](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-stackview.html#initialItem-prop). Элементы, которые протолкнули в стек, имеют стековые присоединённые свойства. Есть три первичных операции навигации по стеку: [push()](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-stackview.html#push-method), [pop()](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-stackview.html#pop-method), and replace (replace by specifying argument replace to [push()](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-stackview.html#push-method)). Первая добавляет элемент в вершину стека, второй удаляет верхний элемент из стека, а последняя заменяет верхний элемент стека другим. Наивысший элемент стека соответствует тому, который виден сейчас на экране. Это значит, что при помощи стека можно осуществлять навигацию по приложению. Иногда необходимо отходить назад более, чем на один шаг. Для этого можно задать элемент как параметр функции pop(). Эта функция разворачивает стек. *Есть замечательные примеры использования операций стека.* Глубокое связывание означает вызов приложения в определённое состояние. В терминах стека это означает возможность модифицировать состояние стека, также как и возможность протолкнуть набор элементов наверх стека, или полностью переустановить стек в данное состояние. Это просто добавление в стек нескольких состояний за раз. *Есть замечательная демонстрация использования данной возможности.* Элемент, который можно протолкнуть в стек, может быть или элементом, или url, или строкой с url, или компонентом. *Показано, как протолкнуть элемент в стек. Далее перечислены некоторые свойства, которые задают, как элемент должен быть помещён в стек. Также далее показаны упрощённые семантики методов стека для некоторых частных случаев проталкивания в стек.* Если встроенный элемент проталкивается в стек, то он временно становится наследником стека. А когда он извлекается из стека, то его родителями становятся прежние. Компоненты и url уничтожаются при извлечении из стека, но если компонент объявлен как элемент, то этого не происходит. Также помните, что если родитель некоторого элемента будет уничтожен, то и сам элемент также будет уничтожен.Жизненный цикл элемента в стеке может иметь следующие переходы: инстанцирование, неактивен, активация, активен, деактивация, неактивен, уничтожение. Между активным и неактивным он может двигаться сколь угодно много раз. Если элемент активируется, то он становится виден на экране и становится текущим элементом. Если элемент не виден на экране, то он неактивен, даже если находится наверху стека. Если стек становится видимым, то верхний элемент в стеке становится активным, а если стек прячется, то и элемент становится неактивным. [Stack.status](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-stack.html#status-prop) свойство следит за жизненным циклом. *Далее перечислены значения статуса, которые могут быть.* Иногда необходимо найти некоторый элемент, чтобы, например, развернуть до него стек. Для этого есть функция find(). Есть также функция get(), которую следует использовать, если ваш элемент зависит от другого элемента в стеке, так как функция гарантирует, что элемент с заданным индексом будет вначале загружен, прежде чем возвращён. Переходы осуществляются каждый раз, когда элемент проталкивается или извлекается, и состоит из двух элементов: enterItem and exitItem. Стек сам по себе никогда не двигает элементы, но вместо этого делегирует работу внешнему набору анимаций, обеспечиваемому стилем разработчика приложения. Поэтому то, как элементы входят или покидают стек, контролируется извне. Когда переход начинается, стек ищет переход, который совпадает с выполняемой операцией. Есть три возможных перехода: pushTransition, popTransition, and replaceTransition. Каждое реализует то, как enterItem следует анимировать и как exitItem следует извлечь. Переходы собираются внутри объекта [StackViewDelegate](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-stackviewdelegate.html), присваиваемого [delegate](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qml-qtquick-controls1-stackview.html#delegate-prop). *Далее есть пример кода, но он пока мне понятен.* Некоторые переходы должны быть наследниками StackViewTransition, который является [ParallelAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-parallelanimation.html), которая содержит свойства enterItem и exitItem. *Есть пример кода использования более продвинутого перехода.* Когда стек нуждается в новом переходе, то он вызывает StackViewDelegate.getTransition(properties). Базовое применение данной функции только ищет свойство, именованное как properties.name внутри корневого элемента. Вы можете переопределить эту функцию для вашего делегата, если вам необходима некоторая экстра логика для решения, какой переход следует возвратить. Вы, например, желаете проанализировать элемент и возвратить разные анимации в зависимости от его внутреннего состояния. Стек ожидает, что вы вернёте компонент, который состоит из StackViewTransition, или a StackViewTransition напрямую. Первый вариант проще, так как переход будет уничтожен по завершении. *Далее приведён понятный пример кода, а в конце приведены присоединённые к элементам стека свойства. С данным типом было бы интересно поэкспериментировать.*

#### Типы для обработки стилей

<http://qt-project.org/doc/qt-5.1/qtquickcontrolsstyles/qmlmodule-qtquick-controls-styles1-qtquick-controls-styles-1-0.html>

модуль [Qt Quick Controls](http://qt-project.org/doc/qt-5.1/qtquickcontrols/qtquickcontrols-index.html) обеспечивает набор qml типов для обработки стилей. Объект стиля является обычно набором свойств, которые управляют поведением и простых делегаты, которые могут быть замещены qml снипетами.

[ButtonStyle](http://qt-project.org/doc/qt-5.1/qtquickcontrolsstyles/qml-qtquick-controls-styles1-buttonstyle.html) обеспечивает частные стили кнопок.

[CheckBoxStyle](http://qt-project.org/doc/qt-5.1/qtquickcontrolsstyles/qml-qtquick-controls-styles1-checkboxstyle.html), [ComboBoxStyle](http://qt-project.org/doc/qt-5.1/qtquickcontrolsstyles/qml-qtquick-controls-styles1-comboboxstyle.html), [ProgressBarStyle](http://qt-project.org/doc/qt-5.1/qtquickcontrolsstyles/qml-qtquick-controls-styles1-progressbarstyle.html), [RadioButtonStyle](http://qt-project.org/doc/qt-5.1/qtquickcontrolsstyles/qml-qtquick-controls-styles1-radiobuttonstyle.html), [ScrollViewStyle](http://qt-project.org/doc/qt-5.1/qtquickcontrolsstyles/qml-qtquick-controls-styles1-scrollviewstyle.html), [SliderStyle](http://qt-project.org/doc/qt-5.1/qtquickcontrolsstyles/qml-qtquick-controls-styles1-sliderstyle.html), [TabViewStyle](http://qt-project.org/doc/qt-5.1/qtquickcontrolsstyles/qml-qtquick-controls-styles1-tabviewstyle.html), [TableViewStyle](http://qt-project.org/doc/qt-5.1/qtquickcontrolsstyles/qml-qtquick-controls-styles1-tableviewstyle.html), [TextFieldStyle](http://qt-project.org/doc/qt-5.1/qtquickcontrolsstyles/qml-qtquick-controls-styles1-textfieldstyle.html).

### Модуль макетов

<http://qt-project.org/doc/qt-5.1/qtquicklayouts/qmlmodule-qtquick-layouts1-qtquick-layouts-1.html>

модуль Qt Quick макетов обеспечивает qml типы для организации qml элементов в пользовательском интерфейсе. Эти типы qml работают вместе с Qt Quick и элементами управления Qt Quick модулями. *Далее показан импорт.*

[Layout](http://qt-project.org/doc/qt-5.1/qtquicklayouts/qml-qtquick-layouts1-layout.html) предоставляет присоединённые свойства к элементам, помещённым в другие типы макетов.

[RowLayout](http://qt-project.org/doc/qt-5.1/qtquicklayouts/qml-qtquick-layouts1-rowlayout.html) – макет в одну строку.

[ColumnLayout](http://qt-project.org/doc/qt-5.1/qtquicklayouts/qml-qtquick-layouts1-columnlayout.html) – макет в одну колонну.

[GridLayout](http://qt-project.org/doc/qt-5.1/qtquicklayouts/qml-qtquick-layouts1-gridlayout.html) – обеспечивает способ динамической организации элементов в сетку.

*Теперь рассмотри визуальные типы Qt Quick.*

### Визуальные типы

#### Item

<http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-item.html>

Item является базовым визуальным типом qml. Тип инстанцирует класс С++ [QQuickItem](http://qt-project.org/doc/qt-5.1/qtquick/qquickitem.html). Данный объект сам по себе не имеет визуальной внешности. Он определяет все атрибуты, которые являются общими среди всех визуальных элементов: координаты, размеры, якоря и поддержку ключей. Данный тип может быть полезен для группировки нескольких визуальных элементов в группу при помощи одного корневого элемента. *Есть пример кода.* Поддержка клавиш доступна всем наследующим данный типам через присоединённое свойство [Keys](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-keys.html). Данные свойства показывают, нажат ли элемент, находится ли в нём курсор и т.д. *есть пример кода.* При помощи присоединённого свойства [LayoutMirroring](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-layoutmirroring.html) макеты могут быть зеркально отражены.

#### [Rectangle](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-rectangle.html), [Image](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-image.html)

[*http://qt-project.org/doc/qt-5.1/qtquick/qtquick-qmltypereference.html*](http://qt-project.org/doc/qt-5.1/qtquick/qtquick-qmltypereference.html)

[Rectangle](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-rectangle.html), [Image](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-image.html).

BorderImage

<http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-borderimage.html#details>

BorderImage тип используется для создания границ за пределами изображений или для покрытия зонами каждого изображения. Данный тип разбивает изображение на 9 областей. Затем при помощи масштабирования различные зоны изображения масштабируются по-разному, что очень удобно для создания границ.

#### AnimatedImage

<http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-animatedimage.html#details>

AnimatedImage расширяет особенности типа изображения, обеспечивая способ проигрывания анимаций, сохранённых как изображения, содержащие наборы фреймов, такие как GIF файлы.

#### AnimatedSprite

<http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-animatedsprite.html#details>

AnimatedSprite обеспечивает рисование и контроль над анимацией, которая обеспечивается как множественные фреймы одном и том же файле изображений.

#### Другие визуальные типы

<http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-spritesequence.html#details>

SpriteSequence рисует и управляет набором анимаций, определённым типом Sprite.

Text – для вставки форматированного текста на экран.

Window – обеспечивает окно высокого уровня в приложении.

Accessible – присоединённое свойство, которое делает элементы доступными.

Gradient, GradientStop (определение цвета для градиента).

SystemPalette – доступ к палитрам qt.

Screen – информация о экране, на котором изображен элемент.

QQuickSprite рисует спрайты одного или более фреймов и анимирует их. Спрайты могут быть в середине файлов изображения, или вдоль множественных рядов и т.д.

FontLoader загружает шрифты при помощи имени url.

#### Repeater

<http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-repeater.html#details>

Repeater используется для создания большого количества похожих элементов. Он имеет модель и делегат. Элемент данного типа обычно заключён в типе позиционирования, таком как строка или колонка, для того чтобы визуально размещать множественные делегаты, созданные данным типом. *Есть пример кода, из которого становится всё ясно.*

#### Loader

<http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-loader.html#details>

Loader используется для динамической загрузки qml компонентов. Он может загружать qml файл или объект компонента. Он полезен для задержки создания компонентов, пока они действительно не понадобятся. *Есть пример кода использования загрузчика. Далее приводится описание и пример кода касательно правил изменения размера загружаемого элемента.* Результат таков: размеры загрузчика и загружаемого элемента оказываются идентичными. Все сигналы от загруженного объекта могут быть получены через тип Connections. *Есть понятный пример кода. Есть некоторое предостережение при использовании данного класса внутри делегата. Я пока его не понял, но в будущем, возможно, разберусь.*

#### Визуальные типы для трансформаций

[**http://qt-project.org/doc/qt-5.1/qtquick/qtquick-qmltypereference.html**](http://qt-project.org/doc/qt-5.1/qtquick/qtquick-qmltypereference.html)

Transform – позволяет определение продвинутых преобразований элементов.

Scale – присваивает поведения масштабирования элементов.

Rotation – присваивает поведения вращения элементов.

Translate – присваивает поведения трансляции элементов.

### Модуль ввода пользователя

[**http://qt-project.org/doc/qt-5.1/qtquick/qtquick-qmltypereference.html**](http://qt-project.org/doc/qt-5.1/qtquick/qtquick-qmltypereference.html)

MouseArea – устанавливает область взаимодействия с мышью.

Keys – обеспечивает компоненты присоединёнными свойствами для обработки ввода с клавиш.

KeyNavigation – поддерживает клавишную навигацию при помощи стрелок.

FocusScope создаёт область видимости фокуса. Помогает в обработке фокуса ввода клавиатуры при построении повторно используемых qml компонентов.

#### Flickable

<http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-flickable.html#details>

Flickable размещает своих детей на поверхности, которая может быть перетащена и щёлкнута, вызывая прокрутку представления на дочерних элементах. Это поведение образует основу элементов, которые разработаны, чтобы показать большое количество дочерних элементов, таких как [ListView](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-listview.html) and [GridView](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-gridview.html). Иногда возможно перемещать представления при помощи мыши. Это полезно, например, при организации прокрутки на мобильном устройстве.

#### Типы ввода-вывода

PinchArea – обработка щипков.

MultiPointTouchArea – обеспечивает обработку множественных точек прикосновения.

Drag – для определения событий перетаскивания и отпускания для визуальных элементов.

DropArea – для определения события перетаскивания и отпускания в некоторой области.

TextInput – захватывает ввод пользователя с клавиатуры.

TextEdit – отображает много линий форматируемого редактируемого текста.

IntValidator, DoubleValidator, RegExpValidator.

TouchPoint – описывает точку прикосновения в [MultiPointTouchArea](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-multipointtoucharea.html).

PinchEvent, WheelEvent, MouseEvent, KeyEvent, DragEvent.

Теперь переходим к блоку типов позиционирования.

### Типы расположения элементов

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-qmltypereference.html>

[Positioner](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-positioner.html) – присоединённое свойство, которое обеспечивает информацию о том, где элемент размещается.

[Column](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-column.html) – организует его наследников вертикально.

[Row](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-row.html) – организует его наследников горизонтально.

[Grid](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-grid.html) – то же, но по сетке.

[Flow](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-flow.html) – *для понимания данного типа лучше посмотреть наглядный пример по ссылке* [*http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-flow.html*](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-flow.html)*.*

[LayoutMirroring](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-layoutmirroring.html) – присоединённое свойство, для поведения зеркального отражения макетов.

### Состояния, переходы и анимации

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-qmltypereference.html>

[State](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-state.html) – определяет и устанавливает конфигурацию объектов и свойств.

[PropertyChanges](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-propertychanges.html) – описывает изменения свойств внутри состояния.

[StateGroup](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-stategroup.html) – содержит набор состояний и переходов состояний.

[StateChangeScript](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-statechangescript.html) – позволяет скриптовое связывание в состоянии (некоторый скрипт, который описывает переход).

[ParentChange](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-parentchange.html) – описывает, как в некотором состоянии изменить родителя.

[AnchorChanges](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-anchorchanges.html) – изменяет якоря элемента в некотором состоянии.

[Transition](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-transition.html) – анимирует переходы во время изменения состояний.

[ViewTransition](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-viewtransition.html) – определяет элементы для перехода в представлении (тип используется для организации анимаций в таких представлениях как ListView при изменении модели. В документации по данному типу есть множество примеров кода).

[SequentialAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-sequentialanimation.html) – запускает анимацию последовательно.

[ParallelAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-parallelanimation.html)

[Behavior](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-behavior.html) – определяет анимацию по умолчанию для изменения свойств (если свойство изменяется, то включается данная анимация).

[PropertyAction](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-propertyaction.html) – устанавливает немедленные изменения свойства во время анимации.

[PauseAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pauseanimation.html) – вводит паузу в анимации.

[SmoothedAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-smoothedanimation.html) – позволяет Гладко изменять значение свойства.

[SpringAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-springanimation.html) – позволяет свойству изменять значение наподобие движения пружины.

[ScriptAction](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-scriptaction.html) – запускает скрипты во время анимации (во время анимации запускает некоторый скрипт).

[PropertyAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-propertyanimation.html) – анимирует изменения свойств.

[NumberAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-numberanimation.html) – изменяет свойства типа qreal.

[Vector3dAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-vector3danimation.html), [ColorAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-coloranimation.html), [RotationAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-rotationanimation.html), [ParentAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-parentanimation.html), [AnchorAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-anchoranimation.html).

[PathAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pathanimation.html) - анимирует позиции вдоль пути (видимо, изменения значений).

[PathInterpolator](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pathinterpolator.html) – позволяет создавать анимацию вручную вдоль пути.

[AnimationController](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-animationcontroller.html) – позволяет ручной контроль за прогрессом в анимации.

[Path](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-path.html) – определяет путь, используемый [PathView](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pathview.html), он состоит из одного или нескольких сегментов пути, таких как [PathLine](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pathline.html), [PathQuad](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pathquad.html), [PathCubic](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pathcubic.html), [PathArc](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-patharc.html), [PathCurve](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pathcurve.html), [PathSvg](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pathsvg.html).

[PathLine](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pathline.html) – определяет линию в пути.

[PathQuad](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pathquad.html) – определяет квадратичную кривую Безе в пути.

[PathCubic](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pathcubic.html) – кубическую.

[PathArc](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-patharc.html), [PathCurve](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pathcurve.html), [PathSvg](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pathsvg.html).

[PathAttribute](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pathattribute.html) – позволяет устанавливать атрибуты вдоль пути.

[PathPercent](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pathpercent.html) – изменяет распределение элементов вдоль пути (в документации к данному типу есть хороший пример, иллюстрирующий его возможности).

### Типы моделей и представлений, а также хранилищ данных и доступа

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-qmltypereference.html>

[ListModel](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml-models2-listmodel.html) – определяет список данных.

[ListElement](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml-models2-listelement.html) – определяет элемент данных в модели списка.

[ListView](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-listview.html), [GridView](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-gridview.html), [PathView](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pathview.html) (визуализация содержимого модели вдоль пути).

[Package](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtquick2-package.html) – коллекция, которая позволяет разделять элементы между различными представлениями.

[QtQuick.LocalStorage 2](http://qt-project.org/doc/qt-5.1/qtquick/qmlmodule-qtquick-localstorage2-qtquick-localstorage-2.html) – синглетонный тип для упрощённого доступа SQL.

#### DelegateModel

<http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml-models2-delegatemodel.html#details>

DelegateModel тип инкапсулирует модель и делегат, который будет инстанцирован для элементов модели. Обычно не нужно создавать данный тип модели. Он может быть полезен для манипулирования и доступа к индексам модели, когда подкласс [QAbstractItemModel](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html) используется в качестве модели. Также данный класс используется с типом [Package](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtquick2-package.html), чтобы обеспечить делегатами многие представления, а также вместе с [DelegateModelGroup](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml-models2-delegatemodelgroup.html) для сортировки и фильтрации элементов делегатов. *Есть пример кода использования данного класса.*

#### DelegateModelGroup

<http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml-models2-delegatemodelgroup.html#details>

DelegateModelGroup тип обеспечивает средства для адресации к модели данных [DelegateModel](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml-models2-delegatemodel.html), а также средства для их сортировки и фильтрации.

#### ListModel

<http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml-models2-listmodel.html#details>

тип ListModel является простым контейнером для определения [ListElement](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml-models2-listelement.html), каждый из которых также содержит роль. Содержание может быть определено динамически или явно в qml. Есть всяческие функции, характерные для списка. *Далее рассматривается многочисленный код для демонстрации работы с данной моделью.*

#### List elements

<http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml-models2-listelement.html#details>

List elements определяются внутри [ListModel](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml-models2-listmodel.html) и представляют элементы в списке, которые будут отображены с использованием [ListView](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-listview.html) или [Repeater](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-repeater.html) элементов. Данные элементы определяются также как и другие qml элементы за тем лишь исключением, что они имеют не свойства, а роли. Роль определяет и как осуществляется доступ к данным, и сами данные. Имена ролей должны начинаться со строчных букв, и должны быть общими для всех элементов в данной модели. Значения должны быть простыми константами: строки, булевы значения, числа или перечисления. Имена ролей используются делегатами для получения данных из элементов списка. Каждое имя роли доступно в области видимости делегата. *Есть пример кода использования данного класса. Очень хороший и понятный пример.*

#### ObjectModel

<http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml-models2-objectmodel.html#details>

ObjectModel тип содержит визуальные элементы, которые используются в представлении. Когда данная модель используется в представлении, представление не требует делегата, так как модель уже не будет содержать визуальный делегат. Элемент может определить его индекс в модели при помощи присоединённого свойства index. *Есть пример кода.*

### Графические эффекты.

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-qmltypereference.html>

[Flipable](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-flipable.html) – обеспечивает поверхность, которая производит эффект листания.

[ShaderEffect](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-shadereffect.html) – разрешает GLSL шэйдеры в качестве графических эффектов.

[ShaderEffectSource](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-shadereffectsource.html) – используется как текстура в [ShaderEffect](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-shadereffect.html).

[GridMesh](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-gridmesh.html) – генерирует сетку ячеек узлов для использования [ShaderEffect](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-shadereffect.html).

[QtQuick.Particles 2](http://qt-project.org/doc/qt-5.1/qtquick/qmlmodule-qtquick-particles2-qtquick-particles-2.html) – модуль обеспечивает систему частиц Qt Quick.

### Некоторые удобные типы.

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-qmltypereference.html>

Connections, Binding, Timer

[WorkerScript](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtquick2-workerscript.html) – предоставляет возможность использования потоков в Qt Quick приложении. Он запускает некоторые операции в новом потоке.

### Типы для канвы (наподобие HTML5)

[Canvas](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-canvas.html) – обеспечивает двумерную канву для рисования средствами java script.

[Context2D](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-context2d.html) – обеспечивает двумерный контекст для форм ка канве.

[CanvasGradient](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-canvasgradient.html) – градиент для канвы.

[CanvasPixelArray](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-canvaspixelarray.html) – пиксельный массив канвы.

[CanvasImageData](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-canvasimagedata.html) – позволяет определять данные изображения для использования на канве.

[TextMetrics](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-textmetrics.html) – обеспечивает данные метрики текста и шрифта для использования в канве.

Теперь просматриваем оставшиеся модули qt qml.

### Модуль сенсоров

<http://qt-project.org/doc/qt-5.1/qtsensors/qmlmodule-qtsensors5-qtsensors-5-0.html>

если модуль импортируется в пространства имён, некоторые дополнительные методы становятся доступными. Это касается модуля qt сенсорс qml типы. Этот модуль обеспечивает определённые типы для сенсоров. *Далее следует много типов, предназначение которых для меня пока остаётся загадкой. Эти типы связаны с классом QSensor.*

## QT QUICK

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-index.html>

Qt Quick модуль является стандартной библиотекой для написания qml приложений. Если qml модуль обеспечивает qml движок и инфраструктуру языка, Qt Quick модуль обеспечивает базвые типы, необходимые для создания интерфейсов пользователя при помощи qml. Qt Quick модуль обеспечивает как qml программный интерфейс, так и С++ программный интерфейс для расширения qml приложений при помощи С++ кода.

Qt Quick содержит всё, что необходимо для написания богатого приложения с плавным и динамичным интерфейсом пользователя. Он предоставляет кроме прочего собственную канву с собственным движком рисования. *Затем идут ссылки на информацию, которую я уже прочитал, а в конце идут ссылки на расширения С++, которые я почитают в дальнейшем.*

### ВИЗУАЛЬНАЯ КАНВА QT QUICK

[*http://qt-project.org/doc/qt-5.1/qtquick/qtquick-visualcanvas-topic.html*](http://qt-project.org/doc/qt-5.1/qtquick/qtquick-visualcanvas-topic.html)

визуальная канва, обеспечиваемая Qt Quick, является двумерной канвой с z-ориентацией. Есть два разных вида родительства в qml приложениях, которые использует Qt Quick.

* Первый вид является владением, который определяет семантику времени жизни объекта.
* Второй вид является визуальным, который определяет, где на канве следует рисовать элемент, а также некоторые свойства.

Почти во всех случаях они идентичны.

Современные компьютерные системы и устройства используют OpenGL для рисования графики. Qt Quick требует OpenGL и он используется для отображения приложений, разработанных при помощи Qt Quick в qml. В частности, Qt Quick определяет граф синов, который затем рисуется.

#### ВИЗУАЛЬНЫЕ КООРДИНАТЫ В QT QUICK

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-visualcanvas-coordinates.html>

в Qt Quick системой координат по умолчанию является картезианская система координат с началом в левом верхнем углу. Позиция отдельного элемента определяется на основании системы координат его родителей.

Scene координаты являются координатами, где (0,0) соответствует верхнему левому углу окна, которое в настоящий момент времени нарисовано. Они обычно такие же, как координаты корневого элемента окна. Вы можете преобразовать координаты элемента в координаты scene с использованием специальных функций. *Есть прекраснейший пример кода для демонстрации разницы данных координат.*

#### ВИЗУАЛЬНЫЕ РОДИТЕЛИ В QT QUICK

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-visualcanvas-visualparent.html>

при создании визуальных синов при помощи Qt Quick важно понять понятие визуального родителя. Концепция визуального родителя в Qt Quick отдельная, но связанная с понятием объектного родителя для QObject иерархии. Все qml объекты имеют объекта-родителя, который определяется иерархией объектов, в которой объект объявлен. При работе с модулем Qt Quick тип Item является основным типом для всех визуальных элементов, обеспечиваемых данным модулем, и именно он обеспечивает понятие дополнительного визуального родителя, как определено свойством [parent](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-item.html#parent-prop) элемента. Каждый элемент имеет визуального родителя, так как если значение этого свойства равняется нулю, то элемент не будет отображён.

Любой объект, присвоенный свойству элемента data, становится наследником элемента внутри QObject иерархии с целью управления памятью. Вдобавок, если объект, добавленный к данному свойству, также является типом элемента, он также присваивается к свойству [Item::children](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-item.html#children-prop) и становится наследником элемента внутри иерархии визуальных декораций. Для удобства свойство [data](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-item.html#data-prop) элемента является свойством по умолчанию. Визуальный родитель элемента может быть изменён в любой момент времени при помощи присваивания нового значения его свойству property. Поэтому они могут отличаться от родителей объектов.

Присваивание элемента в качестве наследника некоторого другого элемента не означает автоматически, что наследник будет соответственно размещён или изменён в размере, чтобы приблизиться к родителю. Некоторые qml типы имеют встроенные поведения, которые виляют на размещение их дочерних элементов, - например, Row автоматически размещает его детей в горизонтальной формации. Вдобавок, родитель не будет автоматически усекать его детей так, чтобы визуально содержать их внутри своих границ, если только не установлено свойство [clip](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-item.html#clip-prop).

Координаты элемента являются относительными к визуальному родителю. Поэтому они могут подвергнуться воздействию при изменении визуальной иерархии. Qt Quick использует рекурсивный алгоритм рисования для определения, какие элементы нарисованы на вершине в случае их совмещения. В общем, элементы рисуются наверху своих родителей в том порядке, в котором они были созданы. *Есть очень хороший пример кода.* Порядок наложения может быть изменён при помощи свойства [Item::z](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-item.html#z-prop). Значения меньше нуля будут находиться ниже родителя, а наложение будет происходить в порядке определения данного свойства элементов. Данная координата изменяет порядок только для соседей в иерархии и для родительских элементов.

Определение того, что рисуется в Qt Quick синах, является визуальным деревом, растущим в [QQuickWindow::contentItem](http://qt-project.org/doc/qt-5.1/qtquick/qquickwindow.html#contentItem-prop). Поэтому для добавления элемента в специфическую Qt Quick декорацию для рисования необходимо стать наследником визуальной иерархии элемента, который уже находится в иерархии визуальных элементов, такой как [QQuickWindow::contentItem](http://qt-project.org/doc/qt-5.1/qtquick/qquickwindow.html#contentItem-prop).

#### ТИПЫ ПРЕОБРАЗОВАНИЙ QT QUICK

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-effects-transformations.html>

преобразования применяются к дочерним иерархиям и также преобразуют ввод мыши, так что координаты в обработчиках событий ведут себя, как ожидается. *Далее перечислены свойства, используемые для преобразования элементов.* В дополнение к удобным типам, также возможно определить более сложные трансформации с использованием списка [Scale](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-scale.html), [Rotation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-rotation.html), and [Translate](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-translate.html) объектов.

### ВВОД ПОЛЬЗОВАТЕЛЯ В QT QUICK

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-input-topic.html>

способность отвечать на ввод пользователя является фундаментальной частью дизайна интерфейса пользователя. В зависимости от того прецедента, который решает приложение, а также от форм-фактора устройства, на котором выполняется приложение, лучший способ получения ввода пользователя может отличаться. В Qt Quick есть несколько типов для обработки взаимодействия пользователя напрямую с экраном.

Qt Quick обеспечивает тип MouseArea для, который автоматически получает события от мыши. Любой визуальный элемент получает ввод с клавиатуры через Keys присоединённый тип. В дополнение вырастает вопрос о фокусе клавиатуры, когда множественные элементы должны получить события клавиш, чтобы эти события могли быть переданы правильному элементу.

Модуль Qt Sensors обеспечивает qml типы для поддержки движений самого устройства, которые фиксируются при помощи акселерометра или камеры.

#### СОБЫТИЯ МЫШИ

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-input-mouseevents.html>

qml использует сигналы и обработчики, чтобы получать взаимодействия с мышью. В частности, Qt Quick обеспечивает типы [MouseArea](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-mousearea.html) and [MouseEvent](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-mouseevent.html), которые позволяют разработчикам определять обработчики сигналов, которые принимают события мыши внутри определённой области. Одним из быстрых способов определения MouseArea является определение данного объекта при помощи якоря в области его родителя. *Есть пример кода. Далее перечислены сигналы данного типа, и приведён пример их обработки. Также перечислены некоторые специфические свойства.* Сигналы и их обработчики получают объект [MouseEvent](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-mouseevent.html) как параметр. Объект мыши содержит информацию о событии мыши. Например, о кнопке, которая инициировала событие. Многие сигналы высылаются много раз, чтобы отразить различные события мыши, такие как двойной щелчок. Чтобы облегчить классификацию щелчков мыши, объект [MouseEvent](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-mouseevent.html) имеет допустимое свойство, чтобы запретить распространение событий. *Последнее я не совсем понял.*

#### ФОКУС КЛАВИАТУРЫ В QT QUICK

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-input-focus.html>

когда клавиша нажата или отпущена, то событие клавиши генерируется и доставляется элементу Qt Quick с фокусом. Чтобы облегчить конструирование повторно используемых компонентов и адресовать некоторые из случаев, уникальных для текучего пользовательского интерфейса, Qt Quick элементы добавляют расширение, основанное на области видимости, к традиционной модели фокуса ввода с клавиатуры.

Когда пользователь нажимает или освобождает клавишу, происходит следующее:

1. qt получает действие клавиши и генерирует событие клавиши.
2. если [QQuickWindow](http://qt-project.org/doc/qt-5.1/qtquick/qquickwindow.html) является активным окном, событие клавиши доставляется ему.
3. событие клавиши доставляется при помощи сина элементу с активным фокусом. Если нет элементов с активным фокусом, то событие клавиши игнорируется.
4. если [QQuickItem](http://qt-project.org/doc/qt-5.1/qtquick/qquickitem.html) с активным фокусом допускает событие клавиатуры, распространение останавливается. С другой стороны, событие высылается родителю элемента, пока оно не будет принято, или пока не достигнет корневого элемента. *Есть пример кода.*
5. если корневой элемент достигается, событие клавиши игнорируется и регулярная обработка qt клавиши продолжается.

Есть свойство, которое представляет, есть ли у элемента активный фокус. *Далее показан код, который не следует использовать для установки фокуса ввода клавиатуры для некоторого типа.*

Для решения проблемы перекрытия фокусов ввода различных объектов вводится понятие области видимости ввода. Он создаётся при помощи объявления типа [FocusScope](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-focusscope.html). Работает он как удалённый доступ фокуса. Концептуально области видимости фокуса являются достаточно простыми (*есть пример кода*):

* внутри каждой области видимости фокус может иметь установленное в правду свойство focus. Если это имеет место для более одного элемента, то фокус будет иметь только последний элемент, также как если бы не было фокуса ввода.
* Когда область видимости фокуса получает активный фокус, содержимый тип с установленным фокусом также получает активный фокус. Если этот тип также является типом области видимости фокуса, то поведение удалённого доступа продолжается. Свойство фокуса будет установлено активным и для области видимости фокуса, и для содержащегося в нём элемента.

Так как область видимости фокуса не является визуальным типом, то в ней следует предусматривать свойства, которые будут использоваться другими типами для отображения содержимого данной области видимости. *Есть пример кода.* Области видимости фокуса позволяют легко разделять фокус. Некоторые элементы qml используют его именно из-за данного эффекта. *Следует экспериментально разобраться с данным классом.*

#### ОБРАБОТКА ВВОДА ТЕКСТА В QT QUICK И ВАЛИДАТОРЫ

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-input-textinput.html>

Qt Quick обеспечивает несколько типов для отображения текста на экране. [Text](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-text.html) тип будет отображать форматированный текст на экране; [TextEdit](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-textedit.html) тип размещает множественные линии редактора текста на экране; [TextInput](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-textinput.html) будет размещать единичное редактируемое поле на экране. Типы validator ограничивают тип и формат объектов [TextInput](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-textinput.html). *Есть пример кода.* Также следует отметить, что qml анализирует java script регулярные выражения, тогда как qt QRegExp класс основан на регулярных выражениях Perl.

### ПОЗИЦИОНИРОВАНИЕ

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-positioning-topic.html>

визуальные элементы qml могут быть размещены различными способами. Наиболее важным понятием, связанным с позициями, является якорение, форма относительного размещения, где элементы могут быть присоединены к любой другой грани. Другие понятия размещения включают абсолютное размещение, размещение с координатным связыванием, позиционеры и макеты.

В Qt Quick любой визуальный объект размещается внутри координатной системы, обеспечиваемой визуальной канвой Qt Quick. Как описано в документе, координаты визуального объекта являются относительными к положению его визуального родителя. *Есть пример кода.*

Элементы можно размещать вручную. если интерфейс пользователя является статическим, то это наиболее оптимальный путь. К тому же для таких интерфейсов такой метод является наиболее оптимальным.

Также элементы можно разместить при помощи присваивания выражений связывания к свойствам, связанным с их положением на визуальной канве. Данный тип является наиболее динамичным для интерфейса пользователя, однако он также и является затратным с точки зрения производительности. Поэтому лучше предпочитать якорение.

Якоря позволяют разместить элемент либо прилегающим, либо внутри другого элемента, при помощи присоединения одной или более точек якорения элемента к точкам якорения другого элемента. Эти якоря будут оставаться, даже если измерение или положение одного из элементов изменится, позволяя создавать высоко динамичные интерфейсы пользователя. Qt Quick предоставляет якоря как понятия высокого уровня. Связывание при помощи якорей является значительно более производительным, чем связывание при помощи выражений. Чистый макет якорей – это тот, который использует только якоря, а загрязнённый макет – это тот, который использует как якоря, так и связывание.

Также Qt Quick предоставляет несколько встроенных типов позиционеров. Для большинства прецедентов лучшим позиционером является сетка, ряд и колонна, и Qt Quick обеспечивает элементы, которые будут размещать детей в этих формациях наиболее возможным эффективным способом.

Модуль Qt Quick макетов также можно использовать для организации Qt Quick элементов в интерфейсе пользователя. В отличие от позиционеров, типы данного модуля управляют как позициями, так и размерами элементов в интерфейсе. Они хорошо подходят для интерфейсов пользователей, которые могут изменять размер.

#### ПОЗИЦИОНИРОВАНИЕ ПРИ ПОМОЩИ ЯКОРЕЙ

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-positioning-anchors.html>

предполагается, что каждый элемент имеет 7 невидимых линий (*перечислены в данной части*). Система якорения Qt Quick позволяет вам определить взаимоотношения между якорными линиями разных элементов. *Есть пример кода. Далее следуют различные примеры для якорения.* При помощи определения горизонтальных и вертикальных якорей вы можете контролировать размер элемента (*есть пример кода*). Есть якоря для удобства*, которые описаны в данной части*: fill и centerIn. Система якорения также предоставляет поля и смещения, которые определяются для якорей элемента. Поле определяет пустое место, которое следует оставить между якорями, а смещение позволяет манипулировать размещением с использованием центральных якорных линий. *Есть примеры использования полей.* Поля применяются только к якорям, а не к элементу.

Qt Quick обеспечивает тип [AnchorChanges](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-anchorchanges.html) для определения якорей в состоянии. *Есть пример кода.* Изменения якорей можно анимировать с использованием типа [AnchorAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-anchoranimation.html). *Есть пример кода.*

Также якоря можно изменять императивно с использованием java script. Но они должны быть внимательно выставлены в соответствующем порядке, или они могут производить неожиданные результаты. *Есть хороший иллюстративный пример.*

По соображениям производительности вы можете прикреплять с помощью якоря элемент к его соседям или прямому родителю. *Есть понятный иллюстративный пример кода.*

Также основанные на якорях макеты не могут быть смешаны с абсолютным размещением. То есть нельзя одновременно, например, выставлять координату х и горизонтальный якорь. Если вы желаете сменить использование якорей на использование абсолютных позиций, то вам следует очистить якоря при помощи установки их в undefined состояние.

#### ПОЗИЦИОНЕРЫ ЭЛЕМЕНТОВ

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-positioning-layouts.html>

позиционеры – это элементы контейнеры, которые управляют позициями элементов в декларативном интерфейсе пользователя. Они работают также, как и менеджеры макетов, которые используются в стандартных qt виджетах за исключением того, что они также являются контейнерами сами по себе. Модуль Qt Quick макетов же управляет как позициями, так и положениями элементов. *Есть примеры кода использования трёх позиционеров.* [Flow](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-flow.html) тип используется для размещения элементов как слов в тексте. *Есть пример использования данного позиционера.*

### СОСТОЯНИЯ, ПЕРЕХОДЫ И АНИМАЦИИ

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-statesanimations-topic.html>

в любом современном интерфейсе пользователя переходы между состояниями и анимация интерфейса пользователя являются очень желательными. Это первостепенные понятия в Qt Quick. Состояние некоторого визуального элемента – это набор информации, которая описывает, как и где отдельные части визуального элемента отображаются внутри него, и о всех данных, связанных с ним. Большинство визуальных элементов в интерфейсе пользователя имеют ограниченное число состояний, каждое с хорошо определёнными свойствами. Например, элемент в списке может быть выбранным или нет, а если выбран, то он может быть выбран только один или же входить в некоторое выделение. Каждое из данных состояний может иметь некоторый визуальный вид. Qt Quick обеспечивает тип State со свойствами, которые определяют его семантику и могут быть использованы для переключения поведения или анимации.

Когда визуальный элемент переходит из одного состояния в другое, то вид этого элемента изменяется. Переход – это граница между двумя состояниями. Он может инициировать совершение других событий, так же как другие части приложения могут иметь поведение, которое инициируется, когда осуществляется вход в определённое состояние или выход из него. Во время перехода между состояниями, плавная анимация может быть использована для помощи пользователю во время перехода. Резкие и неожиданные изменения в визуальной канве приводят к неоптимальному опыту пользователя и их следует избегать. Эти типы анимации поддерживаются Qt Quick через различные типы анимации и переходов. Анимации не только связываются с состояниями и переходами между ними. Например, анимация может быть инициирована другими событиями, которые не связаны с данным состоянием. Также предпочтительно всегда анимировать изменения определённых свойств визуальных элементов, несмотря на причину вызова. Для этого Qt Quick обеспечивает тип [Behavior](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-behavior.html), который позволяет клиенту определить поведение анимации для изменений свойств. Это пример Qt Quick модификатора свойств.

Важно отметить, что использование анимации свойств по умолчанию (через тип Behaviour) в совокупности с анимациями состояний-переходов может иногда приводит к неопределённому поведению. Понятие анимированных спрайтов отличается от понятия анимаций, как оно используется в данной части.

#### СОСТОЯНИЯ QT QUICK

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-statesanimations-states.html>

многие дизайны интерфейса пользователя являются управляемыми при помощи состояний: они имеют настройки, которые зависят от текущего состояния. В qml состояние – это набор конфигураций свойств, которые определены в типе State. Разные конфигурации могут, например: показывать одни компоненты и прятать другие; представлять пользователю различные доступные акции; начинать, заканчивать или ставить на паузу анимации; выполнять некоторый скрипт, требуемый в новом состоянии; изменять значение свойства для частного элемента; показывать различное представление на экране. Все основанные на элементах объекты имеют свойство состояния и могут добавлять дополнительные состояния, добавляя дополнительные объекты состояния. Каждое состояние компонента имеет определённое уникальное имя. Для изменения текущего состояния элемента следует присвоить его свойству состояния определённое имя состояния. Объекты, не являющиеся Item, могут использовать состояния через тип [StateGroup](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-stategroup.html). *Есть пример кода для создания состояний некоторого элемента.* [PropertyChanges](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-propertychanges.html) тип будет изменять значения свойств объекта. На объекты ссылаются через их идентификаторы. *Далее есть пример кода изменения состояния.*

Тип состояния не ограничен в выполнении модификаций значений свойств. Он также может:

* Запускать некоторый скрипт с использованием [StateChangeScript](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-statechangescript.html).
* Переписывать существующие обработчики сигналов для объекта с использованием [PropertyChanges](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-propertychanges.html).
* Изменить родителя элемента с использованием [ParentChange](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-parentchange.html).
* Модифицировать значения якорей с использованием [AnchorChanges](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-anchorchanges.html).

Состояние по умолчанию определяется пустой строкой и содержит начальные значения свойств элемента.

Для удобства тип State имеет свойство when, которое может связывать выражения, чтобы изменять состояние всякий раз, когда связанное выражение является истинным. А если оно не является истинным, то состояние возвратится в состояние по умолчанию.

*При изменении состояния движок быстро загружает его, чтобы просмотреть конечные значения, после чего он выполняет переход непосредственно. Если при этом следует загружать целые списки, то это может выполняться довольно медленно. Данный подход может быть пересмотрен в будущих версиях.*

#### ТИПЫ АНИМАЦИЙ И ПЕРЕХОДОВ

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-statesanimations-animations.html>

[Transition](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-transition.html) – анимация переходов во время изменения состояний;

[SequentialAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-sequentialanimation.html), [ParallelAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-parallelanimation.html).

[Behavior](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-behavior.html) – определяет анимацию по умолчанию для изменения свойства.

[PropertyAction](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-propertyaction.html) – устанавливает немедленные изменения свойств во время анимации.

[PauseAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pauseanimation.html) – вводит паузу во время анимации.

[SmoothedAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-smoothedanimation.html) – позволяет значению сглажено изменять значение.

[SpringAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-springanimation.html) – позволяет свойству установить значение при помощи движения наподобие пружины.

[ScriptAction](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-scriptaction.html) – запускает скрипты во время анимации.

Типы, которые анимируют свойства и типы данных:

[ParentAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-parentanimation.html) – анимация изменений родителя;

[AnchorAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-anchoranimation.html) – анимация якорей;

[PathAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pathanimation.html) – анимация элемента вдоль пути;

[ColorAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-coloranimation.html) – анимация изменений значений цвета;

[NumberAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-numberanimation.html) – анимация изменения вещественных значений;

[Vector3dAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-vector3danimation.html) – анимация значений QVector3d.

[RotationAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-rotationanimation.html) – анимация значений вращения.

[PropertyAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-propertyanimation.html) – анимация значений свойств.

Типы анимации будут интерполировать значения свойств для создания гладких переходов. Также переходы состояний могут присваивать анимации изменениям состояний. Для создания анимации следует использовать подходящий тип анимации для типа свойства, который следует анимировать, и применить анимацию в зависимости от типа поведения, которое требуется. Есть несколько способов установки анимаций на объект.

Прямая анимация свойств. Анимации создаются при помощи применения объектов анимации к значениям свойств для постепенного изменения свойств во времени. Можно устанавливать различные интерполяции. *Есть хороший пример кода.* Использование предопределённых целей и свойств. Есть специальный синтаксис, который определяет анимацию, которая применяется как источник значений свойств. *Есть пример кода.* Также можно использовать группы анимаций, чтобы быть уверенным, что они все применяются к некоторому одному свойству. Переходы во время изменения состояний.

Тип Transition может содержать тип Animation для управления изменением свойства. Для связи перехода с некоторым объектом его следует связать со свойством transitions. *Есть замечательный пример кода, как это работает.* Если в переходе определить в качестве конечного состояния «\*», то это означает, что данный переход выполняется при любом изменении состояний.

Анимации свойств по умолчанию устанавливаются при помощи использования анимаций поведения. Анимации в типе Behavior применяются к свойству и анимируют любое изменение его значения. Но при этом можно разрешать или запрещать данное поведение. *Есть пример, который до конца мне непонятен. Каким образом там осуществляется упругий эффект.*

Также есть специальный синтаксис присваивания поведения свойству. Анимации могут выполняться параллельно или последовательно. *Есть примеры кода.*

Есть несколько методов управления анимациями. Воспроизведение анимации. Все типы анимации наследуют [Animation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-animation.html) тип. Растягивание. Кривые ослабления показывают, как анимация будет интерполировать переход между начальным и конечным значениями. Данные кривые упрощают создание эффектов анимации, таких как эффект отскока, ускорение, замедление, и циклические анимации. Qml объект может иметь разные кривые ослабления для каждой анимации свойств.

Вдобавок qml обеспечивает другие типы, полезные для анимаций:

* [PauseAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-pauseanimation.html) позволяет создавать паузы во время анимаций.
* [ScriptAction](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-scriptaction.html) позволяют выполнять java script во время анимации, и могут быть использованы вместе с [StateChangeScript](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-statechangescript.html), чтобы повторно использовать существующие скрипты.
* [PropertyAction](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-propertyaction.html) изменяет свойство немедленно во время анимации, без анимирования изменения свойства.

Также есть некоторые специализированные типы анимаций, которые анимируют различные свойства типов:

* [SmoothedAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-smoothedanimation.html) – специальный [NumberAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-numberanimation.html), который обеспечивает Гладкие изменения в анимации, когда целевое значение изменяется.
* [SpringAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-springanimation.html) – обеспечивает анимацию наподобие пружины со специальными атрибутами, такими как масса, модуль упругости и т.д.
* [ParentAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-parentanimation.html) – используется для анимации изменения родителей.
* [AnchorAnimation](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-anchoranimation.html) – используется для анимации изменения якорей.
* PropertyAnimation анимирует изменения значений свойства. *В данной части приведены различные варианты кривых ослабления.*

Разделяемые экземпляры анимаций между переходами и поведениями не поддерживаются, и могут привести к неопределённому поведению. *Есть пример кода, который демонстрирует ошибку и её решение.*

#### ИСПОЛЬЗОВАНИЕ QT QUICK ПОВЕДЕНИЙ ВМЕСТЕ С СОСТОЯНИЯМИ

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-statesanimations-behaviors.html>

Иногда вы можете выбрать поведение для изменения свойства, вызванного изменением состояния. Хотя для некоторых ситуаций это работает хорошо, в других ситуациях это может привести к неожиданному поведению. *Есть пример, который показывает проблему. Не понял, о какой проблеме идёт речь. Нужно будет протестировать данную часть.* Авторы заявляют, что в будущем данная проблема будет устранена.

#### СПРАЙТОВЫЕ АНИМАЦИИ

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-effects-sprites.html>

Qt Quick спрайт движок – это стохастическая машина состояний, объединённая со способностью нарезки изображений, содержащих множественные фреймы или анимацию. Первичная функция спрайт движка – это его внутренняя машина. Это не то же самое, что состояния и переходы в Qt Quick, но более похоже на конвенциональную машину состояний. Спрайты могут иметь весовые переходы на другие спрайты или назад. После окончания анимации спрайт выберет следующий спрайт произвольно, основанный на взвешенных переходах, доступных для спрайта, который только что закончился.

Вы можете повлиять на уже играющий спрайт двумя способами. Вы можете произвольно заставить его немедленно начать проигрывание некоторого спрайта или вы можете сказать ему постепенно перейти к заданному спрайту. Тогда он подойдёт к нужному спрайту по кратчайшему пути с игнорированием весовых коэффициентов. Это позволяет вам легко вставить переходную анимацию между двумя произвольными спрайтами. *Потом идёт описание некоторой диаграммы, которое кажется понятным.*

Форматы файлов, доступных при помощи спрайт движка являются такими же, как и форматы файлов, доступные для других типов qml. *Есть пример использования спрайтов, как в качестве описания, так и в качестве кода.*

Фреймы внутри одной анимации должны быть одинакового размера, но многие анимации внутри одного файла – не обязательно. *Также описано, как спрайт тип интерпретирует отсутствие некоторых параметров. Даны какие-то ссылки на какие-то непонятные инструменты.*

Спрайты для спрайт движка могут быть определены с использованием типа Sprite. Это просто класс данных и он ничего не рисует. [SpriteSequence](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-spritesequence.html) – это тип, которые использует спрайт движок, чтобы рисовать спрайты, определённые в нём, и не взаимодействует с другими движками спрайта. Спрайты могут разделяться между многими [SpriteSequence](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-spritesequence.html), но это не делается автоматически. [ImageParticle](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick-particles2-imageparticle.html) может использовать тип спрайт, чтобы определять спрайты для каждой частицы. Для прецедентов, в которых не нужно осуществлять переходы между анимациями, рассмотрите тип AnimatedSprite. Этот тип отображает анимации спрайта с одинаковым форматом ввода, но только один раз. Он также обеспечивает более утончённый контроль, так как здесь нет управления движком спрайтов, который определяет время перехода за кулисами.

### МОДЕЛИ, ПРЕДСТАВЛЕНИЯ И СОХРЕНЕНИЕ ДАННЫХ В QT QUICK

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-modelviewsdata-topic.html>

большинство приложений будет иметь данные, которые необходимо отображать пользователю. Эти данные могут приходить из различных источников: сетевые ресурсы, локальные файлы, базы данных. Представление будет отображать делегата для каждого элемента модели данных. Данные обычно используются для сохранения информации в приложении.

#### МОДЕЛИ И ПРЕДСТАВЛЕНИЯ В QT QUICK

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-modelviewsdata-modelview.html>

Qt Quick имеет понятие о моделях, представлениях и делегатах. Инкапсулируя экземпляр данных в делегате, можно диктовать то, как его следует отображать в приложении. *Далее даются описания данных понятий.*

Чтобы визуализировать данные, следует связать свойство представления модели с моделью и свойство делегата с компонентом или другим совместимым типом. Представления – это контейнеры для коллекций элементов. Они являются богатыми на особенности и могут быть настроены, чтобы удовлетворить требования к стилю или поведению. *Далее перечислены стандартные представления.* Представления позволяют настройку через свойства декорирования. *Есть пример кода настройки внешнего вида представления. Хороший пример.* Представления обрабатывают перетаскивание и опускание их содержания, однако они не поддерживают взаимодействие при помощи прикосновений с отдельными делегатами. Также есть свойство, которое позволяет при перетаскивании изменять или не изменять индекс элемента модели. Содержание ListView можно группировать в секции, где связанные элементы списка помечены в соответствии с их секцией. Затем секции можно декорировать с помощью делегатов. *Есть пример кода. Немного непонятно, почему одна строка подкрасилась в голубой цвет.* Представление будет визуализировать каждый элемент списка в соответствии с шаблоном, определённом делегатом. Элементы в модели доступны через свойство индекса, также как и свойства элемента. *Есть пример кода делегата.* Список доступен из делегата. Модель может быть доступна из представления. Этот механизм может оказаться полезным, если вы хотите использовать одних и тех же делегатов для разных представлений, например, но вы хотите, чтобы декорации или другие особенности были разными для разных представлений. Также иногда важно найти и показать некоторые свойства моделей. *Есть пример кода, где делегат зависит от параметров представления. Фактически здесь можно сделать делегат зависящим от параметров представления.* Данные обеспечиваются делегатом через именованные роли данных, которые делегат может связывать с ними. *Также есть пример кода, в котором делегат комбинирует различные элементы модели данных.* Можно использовать также и модель данных из С++, для чего её следует передать классу [QMLEngine](http://qt-project.org/doc/qt-5.1/qtqml/qqmlengine.html). *Затем описан тип* [ListModel](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml-models2-listmodel.html). *Затем описан тип* [XmlListModel](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick-xmllistmodel2-xmllistmodel.html). *Данный тип в будущем следует изучить подробнее.* [VisualItemModel](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtquick2-visualitemmodel.html) позволяет обеспечивать qml элементы в качестве модели. Данная модель содержит как модель, так и делегаты. Целочисленный тип может рассматриваться как модель, которая содержит определённое число. В данном случае модель не имеет никаких ролей данных. *Есть пример кода, но я не понимаю смысл данного решения.* Экземпляры объектов могут использоваться для определения модели с одним типом объекта. Свойства объекта обеспечиваются как роли. *Есть пример кода.*

Повторители и позиционеры предоставляют лёгкий способ организации при помощи макета большого количества элементов. Элемент повторителя размещается внутри позиционера и генерирует элементы, которые заключающий позиционер организует. *Есть пример кода их использования.* Переходы могут быть использованы для анимации элементов, которые добавляются, перемещаются или удаляются из позиционера. Переходы для добавления элементов применяются к элементам, которые создаются как часть позиционера, также как и к тем, кто изменяет родителя, также как и к тем, кто удаляется, также как и к тем, кто убирается из позиционера и передаётся к новым родителям в документе.

#### ИСПОЛЬЗОВАНИЕ МОДЕЛЕЙ С++ С QT QUICK ПРЕДСТАВЛЕНИЯМИ

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-modelviewsdata-cppmodels.html>

модели можно определить в С++ и затем сделать доступными для qml. Это полезно для показа существующих С++ данных или других сложных наборов данных в qml. С++ класс модели может быть определён как [QStringList](http://qt-project.org/doc/qt-5.1/qtcore/qstringlist.html), [QObjectList](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html#QObjectList-typedef) или [QAbstractItemModel](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html). *Далее рассмотрены некоторые аспекты использования и показаны некоторые примеры кода для первых двух типов моделей. Для последнего типа модели также следует определить соответствующие роли для qml. Есть пример кода использования данного типа модели данных. В технические детали не вдавался.* Qml представления автоматически обновляются при изменении модели, для чего следует использовать обычные уведомления классов моделей данных.

[QAbstractItemModel](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html) представляет иерархию таблиц, но представления в qml предназначены для отображения списков. Для отображения иерархических списков следует использовать тип [VisualDataModel](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtquick2-visualdatamodel.html). *Далее перечислены некоторые свойства и функции, связанные с реализацией.*

Для установки значений модели прямо в qml компоненты (также как и при обновлении) используется функция [QQmlContext::setContextProperty](http://qt-project.org/doc/qt-5.1/qtqml/qqmlcontext.html#setContextProperty)(). Альтернативой использования данной функции является регистрация С++ класса в качестве qml типа. *Есть пример кода. Я уже изучал данную часть.*

### ГРАФИЧЕСКИЕ ЭФФЕКТЫ

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-effects-topic.html>

внешне привлекательные интерфейсы пользователя более зацепляют, чем блеклые интерфейсы. Предположим, что Qt Designer должен держать в уме, что визуальные эффекты просто обеспечивают полезный способ утончённой коммуникации с пользователем. Чрезмерное использование визуальных эффектов может в действительности оттолкнуть пользователя. Визуальные объекты можно преобразовать. Они могут быть увеличены или повёрнуты. Эти трансформации обеспечивают интуитивные подсказки того, что происходит в приложении. Шейдеры позволяют использовать всю графическую силу. Система частиц позволяет производить взрывы, фейерверки, дым, туман и ветер, чтобы они создавали некоторые эффекты и отображались пользователю. Система частиц позволяет двумерную симуляцию данных эффектов. Частицы обычно используются для создания эффектов выбора элементов, либо для возникновения некоторых уведомлений, либо для игр. Спрайт – это анимированное изображение, состоящее из фреймов. Они обычно обнаруживаются в играх. Qt Quick обеспечивает визуальный тип для отображения спрайтов, а также QtQuick обеспечивает визуальный тип, который отображает спрайты, также как и контроллер сложных, стохастических переходов между фреймами для более сложных приложений, которые используют экстенсивно спрайты (такие, как игры).

Визуальные объекты могут быть непрозрачными или полупрозрачными. Например, приложение может сделать один визуальный объект непрозрачным, а остальные – полупрозрачными, чтобы сконцентрировать внимание пользователя на первом объекте. Это контролируется при помощи свойства непрозрачности элемента.

### УДОБНЫЕ ТИПЫ В QT QUICK

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-convenience-topic.html>

в высоко динамичном интерфейсе пользователя разработчик приложения будет часто желать реагировать на события и инициировать различную логику в ответ. Qml имеет встроенную поддержку для данных понятий через связывание, сигналы и обработчики сигналов, а также динамическое инстанцирование объектов, но Qt Quick расширяет поддержку, обеспечиваемую при помощи языка с различными удобными типами. *Далее перечислены средства для динамического инстанцирования объектов.*

[Binding](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-binding.html) тип позволяет цели связывания быть определённой явно и отдельно от определения выражения связывания. При помощи объявления экземпляра [Binding](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-binding.html) клиент может динамически связывать свойства от произвольных объектов во время выполнения, и может модифицировать связывание цели, когда требуется.

Qt Quick обеспечивает удобный тип [Connections](http://qt-project.org/doc/qt-5.1/qtqml/qml-qtqml2-connections.html), который позволяет устанавливать соединение сигнала, вовлекающее объект, который не является частью иерархии. Он также позволяет динамически изменить цель во время выполнения, которая позволяет приложению обработать различные уведомления сигналов с разными функциями в зависимости от состояния программы. При помощи объявления экземпляра данного типа клиент может динамически вызвать испускание сигналов одним объектом, чтобы инициировать методы для другого объекта, и может модифицировать цель соединения, когда требуется (или когда он становится доступным).

Ещё один общий прецедент – это инициирование функциональности на некоторый определённый период времени после происхождения некоторого частного события. Этот тип основанных на таймерах триггеров обеспечивается в QtQuick при помощи типа Timer. Обеспечиваются как однократные, так и рекуррентные таймеры.

### ГРАФ СИНОВ В QT QUICK

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-visualcanvas-scenegraph.html>

Qt Quick 2.0 использует scene graph, основанный на OpenGL 2.0 для рисования. Данная концепция означает, что scene могут сохраняться во фреймах, а полный набор примитивов для рисования будет известен ещё до рисования. Это открывает дорогу ко многим оптимизациям, таким как batch rendering для минимизации изменений состояний и discarding obscured primitives.

Scene graph тесно связан с Qt Quick 2 и не может использоваться отдельно. Scene graph управляется и рисуется при помощи [QQuickWindow](http://qt-project.org/doc/qt-5.1/qtquick/qquickwindow.html) класса и частные типы элементов могут добавить их типы графических примитивов в данный граф при помощи вызова [QQuickItem::updatePaintNode](http://qt-project.org/doc/qt-5.1/qtquick/qquickitem.html#updatePaintNode)(). scene graph – это независимая структура, которая содержит достаточно информации для рисования всех элементов. Он может быть изменён и нарисован независимо от состояний элементов. scene graph состоит из набора предопределённых узловых типов, каждому из которых посвящена своя цель. Более удачным определением данной структуры было бы дерево узлов. Дерево строится из [QQuickItem](http://qt-project.org/doc/qt-5.1/qtquick/qquickitem.html) типов qml, а внутренне scene обрабатывается рисовальщиком, который его рисует. Узлы сами по себе не содержат кода рисования или виртуальной функции рисования. Пользователи могут добавлять к данному дереву свои поддеревья, которые содержат их собственное содержание.

Наиболее важным узлом для пользователя является [QSGGeometryNode](http://qt-project.org/doc/qt-5.1/qtquick/qsggeometrynode.html). Он используется для определения частной графики при помощи определения его материала и геометрии. Геометрия определяется [QSGGeometry](http://qt-project.org/doc/qt-5.1/qtquick/qsggeometry.html) и описывает формы или сетку графического примитива. Материал определяет, как пикселы в этой форме заполняются. Узел может иметь любое количество детей и геометрические узлы будут рисоваться так, как они появляются в порядке родительства, с родителями вначале их детей. Это касается только визуального вывода, так как ничего не сказано о действительном порядке рисования внутри рисовальщика. *Затем перечислены возможные узлы.*

Частные узлы добавляются в граф при помощи специализации [QQuickItem::updatePaintNode](http://qt-project.org/doc/qt-5.1/qtquick/qquickitem.html#updatePaintNode)() и установки [QQuickItem::ItemHasContents](http://qt-project.org/doc/qt-5.1/qtquick/qquickitem.html#Flag-enum) флага. Критическим является то, что OpenGL операции и взаимодействие с графом случается исключительно в потоке рисования, прямо во время вызова updatePaintNode(). Хорошим правилом поэтому является использование только классов с “QSG” префиксом внутри функции [QQuickItem::updatePaintNode](http://qt-project.org/doc/qt-5.1/qtquick/qquickitem.html#updatePaintNode)(). *Дана далее ссылка на некоторый связанный с этим материал.*

Узлы имеют виртуальную препроцессорную функцию [QSGNode::preprocess](http://qt-project.org/doc/qt-5.1/qtquick/qsgnode.html#preprocess)(), которая будет вызываться перед построением графа. *Написано, что необходимо сделать в частных узлах для конечной подготовки к рисованию.* Лучше, чтобы узлом владел граф.

Материал описывает, как заполняется внутренне узел графа. Он инкапсулирует OpenGL шейдеры и обеспечивает достаточную гибкость, так что большинство Qt Quick элементов только используют очень базовые материалы, такие как цвет или текстура. Если вы желаете применить частные шейдеры к qml типу элемента, то возможно сделать это напрямую в qml с использованием типа [ShaderEffect](http://qt-project.org/doc/qt-5.1/qtquick/qml-qtquick2-shadereffect.html). *Далее перечисляется полный список классов материалов.*

Программный интерфейс scene graph является очень низкоуровневым и фокусируется на производительности более, чем на удобстве. Написание частных материалов или геометрий требует много кода. Поэтому программный интерфейс содержит несколько удобных классов для доступа к частным узлам. *Далее они перечислены.*

*Чем ниже уровень кода, тем его больше необходимо для написания тех или иных вещей.*

Рисование данного графа происходит внутренне в [QQuickWindow](http://qt-project.org/doc/qt-5.1/qtquick/qquickwindow.html) классе, и поэтому нет интерфейса для доступа к нему. *Далее показано, как происходит рисование содержимого. Потоковый цикл обработки событий используется на Linux. Далее показано, как заблокировать наличие потока рисовальщика. Непотоковый цикл обработки событий используется на Windows.* Но даже в этом случае следует писать такой код, будто бы рисовальщик находится в своём потоке, чтобы приложение было переносимым. *Далее показана схема рисования в данном случае.*

Граф имеет два метода для интегрирования OpenGL содержимого: при помощи вызова OpenGL команд напрямую и при помощи создания текстурированного узла в графе. При помощи связывания с сигналами [QQuickWindow::beforeRendering](http://qt-project.org/doc/qt-5.1/qtquick/qquickwindow.html#beforeRendering)() and [QQuickWindow::afterRendering](http://qt-project.org/doc/qt-5.1/qtquick/qquickwindow.html#afterRendering)() приложения могут сделать OpenGL вызовы напрямую в том же контексте, что и рисование графа. Как следует из имён сигналов, пользователь затем может выполнить рисование как под графом, так и над ним. *Есть ссылка на пример того, как использовать данные сигналы.*

Другая альтернатива – это создание объекта FramebufferObject, рисование в него и использование результата в текстурированном узле в графе. Например, используя [QSGSimpleTextureNode](http://qt-project.org/doc/qt-5.1/qtquick/qsgsimpletexturenode.html). *Есть ссылки на примеры рисования. Далее есть также ещё один способ использование этого.*

При связывании графа и OpenGL важно, чтобы приложение не оставляло OpenGL контекст в состоянии со связанными буферами, доступными атрибутами, специальными значениями в z-буфере или stencil-буфере. Выполнение этого может привести к непредсказуемым последствиям. OpenGL код должен быть умным по отношению к потокам, так как рисование может происходить за пределами потока графического интерфейса пользователя.

[QQuickPaintedItem](http://qt-project.org/doc/qt-5.1/qtquick/qquickpainteditem.html) позволяет пользователям рисовать контекст с использованием [QPainter](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html). Но использование графа всегда намного быстрее.

*В конце указывается некоторый плагин, который позволяет влиять на рисование графа. Но я пока ещё не очень понял его.*

### КЛАССЫ ГРАФА СИНОВ

#### QSGNode

<http://qt-project.org/doc/qt-5.1/qtquick/qsgnode.html>

QSGNode класс является базовым классом для всех узлов в графе. Есть функции для добавления детей, причём важен в данном случае также и порядок детей. Узлы графа содержат механизм, чтобы описывать, какие части scene были изменены. Они включают объединённые матрицы, накопленную непрозрачность, изменения в иерархии узлов и т.д. эта информации может быть использована внутри рисовальщика для оптимизации. Для того, чтобы рисовальщик корректно рисовал узлы, важно, чтобы пользователи вызывали [QSGNode::markDirty](http://qt-project.org/doc/qt-5.1/qtquick/qsgnode.html#markDirty)() с правильным флагом, когда узлы изменены. Большинство функций неявно вызывают данную функцию.

Если узлы изменяют каждый фрейм, то можно использовать функцию [preprocess](http://qt-project.org/doc/qt-5.1/qtquick/qsgnode.html#preprocess)(), чтобы применить изменения к узлу для каждого фрейма, где он рисуется. Для этого следует установить в узле флаг [QSGNode::UsePreprocess](http://qt-project.org/doc/qt-5.1/qtquick/qsgnode.html#Flag-enum). *Не понимаю смысл данной функции.*

Виртуальная функция [isSubtreeBlocked](http://qt-project.org/doc/qt-5.1/qtquick/qsgnode.html#isSubtreeBlocked)() может быть использована, чтобы сделать недействительным некоторое поддерево. Узлы в данном поддереве не будут preprocessed() и нарисованы.

Любые происшествия с данным классом должны происходить в потоке рисования scene graph.

#### QSGGeometryNode

<http://qt-project.org/doc/qt-5.1/qtquick/qsggeometrynode.html#details>

класс QSGGeometryNode используется для рисования содержания графа. Он состоит из геометрии и материала. Геометрия определяет сетку, а материал то, как её заполнять. *Показан очень хороший пример кода, который показывает, как рисуется с помощью данного класса линия.* Прежде добавления в граф данный узел должен уже иметь как материал, так и геометрию. Если геометрия или материал изменяются после того, как узел добавлен в граф, то пользователю следует отметить их как грязные с использованием [QSGNode::markDirty](http://qt-project.org/doc/qt-5.1/qtquick/qsgnode.html#markDirty)(). Геометрические узлы поддерживают два типа материалов: непрозрачный материал и нормальный материал. Первый используется, если непрозрачность графа в момент рисования равняется 1. Это позволяет существенно улучшить производительность.

#### QSGClipNode

<http://qt-project.org/doc/qt-5.1/qtquick/qsgclipnode.html#details>

QSGClipNode класс реализует функциональность усечения в scene graph. Усечение применяется к поддереву узлов и может быть вложенным. Многие узлы усечения будут собраны при помощи пересечения всех их геометрий. Аккумуляция происходит как часть рисования. Данные узлы должны иметь геометрию прежде их добавления на экран. Обычно усечение реализуется с использованием stencil буфера.

#### QSGTransformNode

<http://qt-project.org/doc/qt-5.1/qtquick/qsgtransformnode.html#details>

QSGTransformNode класс реализует преобразования в scene graph. Преобразования применяются к поддереву и могут быть вложенными. Множественные преобразования будут собираться при помощи пересечения их матриц. Собирание происходит как часть рисования. Матрица сама по себе предназначена для трёхмерных преобразований, хотя в данном случае данные классы оптимизированы под двумерное рисование. Поэтому трёхмерное рисование должно производиться внимательно. *Я бы вообще не использовал бы трёхмерное рисование.*

#### QSGOpacityNode

<http://qt-project.org/doc/qt-5.1/qtquick/qsgopacitynode.html#details>

класс QSGOpacityNode используется для изменения непрозрачности узлов. Непрозрачность применяется к его поддереву и может быть вложенной. Множественные узлы непрозрачности будут собираться при помощи умножения их непрозрачности. Собирание происходит как часть рисования. Когда множественная непрозрачность станет ниже некоторого порогового значения, поддерево может быть отмечено, как заблокированное. Это делается с целью увеличения производительности.

#### QSGMaterialShader

<http://qt-project.org/doc/qt-5.1/qtquick/qsgmaterialshader.html#details>

класс QSGMaterialShader представляет OpenGL шейдер в рисовальщике. Его программный интерфейс является очень низкоуровневым. Более удобным программным интерфейсом, который обеспечивает почти те же особенности, доступен через [QSGSimpleMaterialShader](http://qt-project.org/doc/qt-5.1/qtquick/qsgsimplematerialshader.html). [QSGMaterial](http://qt-project.org/doc/qt-5.1/qtquick/qsgmaterial.html) and QSGMaterialShader образуют тесные взаимоотношения. Для одного графа есть только один экземпляр данного класса, который инкапсулирует [QOpenGLShaderProgram](http://qt-project.org/doc/qt-5.1/qtgui/qopenglshaderprogram.html), чтобы рисовать тот материал. Каждый [QSGGeometryNode](http://qt-project.org/doc/qt-5.1/qtquick/qsggeometrynode.html) может иметь уникальный [QSGMaterial](http://qt-project.org/doc/qt-5.1/qtquick/qsgmaterial.html), содержащий то, как шейдер должен быть настроен во время рисования узла, например, его действительный цвет, используемый для рисования геометрии. Экземпляр данного класса никогда не создаётся пользователем явно. Он создаётся по требованию графом через метод [QSGMaterial::createShader](http://qt-project.org/doc/qt-5.1/qtquick/qsgmaterial.html#createShader)(). При этом граф уверен, что есть только один шейдер. Функция [vertexShader](http://qt-project.org/doc/qt-5.1/qtquick/qsgmaterialshader.html#vertexShader)() используется для управления тем, что материал делает с узлами данных, которые приходят из геометрии. Исходный код, возвращаемый [fragmentShader](http://qt-project.org/doc/qt-5.1/qtquick/qsgmaterialshader.html#fragmentShader)(), используется для управления тем, как материал должен заполнять каждый индивидуальный пиксел в геометрии. Данные функции используются только во время инициализации. [activate](http://qt-project.org/doc/qt-5.1/qtquick/qsgmaterialshader.html#activate)() вызывается графом, когда шейдер начинает использоваться. Во время активности граф может вызвать функцию [updateState](http://qt-project.org/doc/qt-5.1/qtquick/qsgmaterialshader.html#updateState)(), которая обновит состояние шейдера для каждой отдельной геометрии для рисования. *Также есть информация про некоторые другие функции. Есть полезный пример кода, с которым следует разобраться после изучения OpenGL.* Экземпляры данного класса следует использовать только в потоке рисования, но не графического интерфейса пользователя.

#### QSGMaterialType

<http://qt-project.org/doc/qt-5.1/qtquick/qsgmaterialtype.html#details>

QSGMaterialType класс используется как уникальный знак типа в комбинации с [QSGMaterial](http://qt-project.org/doc/qt-5.1/qtquick/qsgmaterial.html). Он имеет смысл только внутри функции [QSGMaterial::type](http://qt-project.org/doc/qt-5.1/qtquick/qsgmaterial.html#type)().

#### QSGMaterial

<http://qt-project.org/doc/qt-5.1/qtquick/qsgmaterial.html#details>

QSGMaterial класс инкапсулирует состояние рисования для программы шейдера. Его программный интерфейс является очень низкоуровневым. Более удобным является [QSGSimpleMaterialShader](http://qt-project.org/doc/qt-5.1/qtquick/qsgsimplematerialshader.html). В графе каждый узел может иметь свой экземпляр данного класса, в котором содержится информация о том, как следует настроить шейдер для рисования узла, например, действительный цвет, используемый для рисования. В классе есть две виртуальные функции, которые необходимо реализовать. *Есть пример кода, где и показаны данные функции.* Экземпляры данного класса принадлежат к потоку рисования графа, а не к потоку интерфейса пользователя.

#### QSGFlatColorMaterial

<http://qt-project.org/doc/qt-5.1/qtquick/qsgflatcolormaterial.html#details>

QSGFlatColorMaterial класс обеспечивает удобный способ рисования геометрии сплошного цвета в графе. Он заполняет каждый пиксел геометрии с использованием сплошного цвета. Цвет может содержать прозрачность. Геометрия для рисования данным классом требует узлы в местоположении атрибута в 0 в объекте [QSGGeometry](http://qt-project.org/doc/qt-5.1/qtquick/qsggeometry.html), чтобы рисовать правильно. *Не понимаю последнее.* Данный класс принимает во внимание, как текущую непрозрачность, так и текущую матрицу, при обновлении его состояния рисования.

#### QSGSimpleMaterialShader

<http://qt-project.org/doc/qt-5.1/qtquick/qsgsimplematerialshader.html#details>

QSGSimpleMaterialShader класс обеспечивает удобный способ построения частных материалов для графа. Если классы [QSGMaterial](http://qt-project.org/doc/qt-5.1/qtquick/qsgmaterial.html) and [QSGMaterialShader](http://qt-project.org/doc/qt-5.1/qtquick/qsgmaterialshader.html) требуют некоторого шаблонного кода для создания функционирующего материала, то данный класс пытается спрятать некоторые из них при помощи использования шаблонов. *Есть описание некоторых важных функций. Есть также информация про некоторый макрос, но я пока не понимаю данной информации. Есть некоторый пример кода, который также пока мне не очень понятен. Также есть некоторый макрос для объявления материла для некоторого состояния шейдера.*

#### QSGSimpleMaterial

<http://qt-project.org/doc/qt-5.1/qtquick/qsgsimplematerial.html#details>

QSGSimpleMaterial класс является шаблонным классом, используемым для сохранения состояния, используемого QSGSimpleMateralShader. Состояние шейдера доступно через шаблонную функцию state().

#### QSGOpaqueTextureMaterial

<http://qt-project.org/doc/qt-5.1/qtquick/qsgopaquetexturematerial.html#details>

класс QSGOpaqueTextureMaterial обеспечивает удобный способ рисования текстурированной геометрии в графе. Непрозрачный будет заполнять каждый пиксел геометрии при помощи поддерживаемой текстуры. Материал не учитывает непрозрачность [QSGMaterialShader::RenderState](http://qt-project.org/doc/qt-5.1/qtquick/qsgmaterialshader-renderstate.html), так что непрозрачные узлы в родительской цепочке узлов, использующих данный материал, не подвергаются воздействию. Прежде использования в графе материал должен содержать набор текстур.

#### QSGTextureMaterial

<http://qt-project.org/doc/qt-5.1/qtquick/qsgtexturematerial.html#details>

QSGTextureMaterial класс обеспечивает удобный способ рисования текстурированной геометрии в графе. Он будет заполнять каждый пиксел в геометрии при помощи обеспечиваемой текстуры. *Затем перечисляются некоторые требования к геометрии. Приводятся некоторые важные функции.* Также должен быть набор текстур перед использованием в графе данного класса.

#### QSGVertexColorMaterial

<http://qt-project.org/doc/qt-5.1/qtquick/qsgvertexcolormaterial.html#details>

QSGVertexColorMaterial класс обеспечивает удобный способ рисования цветной геометрии узла в графе. Он даёт каждому узлу в графе цвет. Пикселы между вершинами будут линейно интерполированы. Цвета могут содержать прозрачность. *Также перечислены требования к геометрии для использования данного класса, и приведена функция для получения данных атрибутов.*

#### QSGSimpleRectNode

<http://qt-project.org/doc/qt-5.1/qtquick/qsgsimplerectnode.html#details>

класс QSGSimpleRectNode является удобным классом для рисования заполненных сплошным цветом прямоугольников с использованием графа.

#### QSGSimpleTextureNode

<http://qt-project.org/doc/qt-5.1/qtquick/qsgsimpletexturenode.html#details>

класс QSGSimpleTextureNode обеспечивается для удобства лёгкого рисования текстурированного содержания с использованием qml scene graph. В классе должна быть некоторая текстура прежде его добавления в граф.

#### QSGGeometry::Attribute

<http://qt-project.org/doc/qt-5.1/qtquick/qsggeometry-attribute.html#details>

QSGGeometry::Attribute описывает атрибут одиночного узла в [QSGGeometry](http://qt-project.org/doc/qt-5.1/qtquick/qsggeometry.html). Данная структура описывает позицию регистра атрибут, размер attribute tuple и тип атрибута. Он также содержит подсказку для рисовальщика, если этот атрибут является атрибутом, описывающим позицию. Рисовальщик графа декораций может использовать данную информацию для выполнения оптимизаций. Также он содержит определённое количество бит, которое зарезервировано для будущего использования.

#### QSGGeometry::AttributeSet

<http://qt-project.org/doc/qt-5.1/qtquick/qsggeometry-attributeset.html#details>

QSGGeometry::AttributeSet описывает, как узлы в [QSGGeometry](http://qt-project.org/doc/qt-5.1/qtquick/qsggeometry.html) строятся.

#### QSGGeometry::ColoredPoint2D

<http://qt-project.org/doc/qt-5.1/qtquick/qsggeometry-coloredpoint2d.html#details>

QSGGeometry::ColoredPoint2D структура является удобной структурой для доступа к двумерным точкам с цветом.

#### QSGGeometry::Point2D

<http://qt-project.org/doc/qt-5.1/qtquick/qsggeometry-point2d.html#details>

QSGGeometry::Point2D структура является удобной структурой для доступа к двумерным точкам.

#### QSGBasicGeometryNode

<http://qt-project.org/doc/qt-5.1/qtquick/qsgbasicgeometrynode.html#details>

QSGBasicGeometryNode класс служит как базовый класс для основанных на геометрии узлов. Он не используется сам по себе.

#### QSGDynamicTexture

<http://qt-project.org/doc/qt-5.1/qtquick/qsgdynamictexture.html#details>

класс QSGDynamicTexture служит как базовый класс для динамически изменяющихся текстур, таких как содержание, которое рисуется в FBO.

#### QSGGeometry

<http://qt-project.org/doc/qt-5.1/qtquick/qsggeometry.html#details>

QSGGeometry класс обеспечивает низкоуровневое хранение графических примитивов в Qt Quick графе декораций. Он сохраняет геометрию примитивов, нарисованных при помощи графа. Он содержит данные о вершинах и об индексах (необязательные). Можно установить моду рисования, которая напрямую отображается в моду OpenGL рисования. Есть различные функции для получения тех или иных атрибутов вершин, а также перечисление данных атрибутов. *Есть пример кода, пока мне не очень понятный.* Данный класс является буфером программного обеспечения, а также сторона клиента в терминах OpenGL рисования. *В конце написано некоторое предложение, которое я пока не понял.*

#### QSGGeometry::TexturedPoint2D

<http://qt-project.org/doc/qt-5.1/qtquick/qsggeometry-texturedpoint2d.html#details>

QSGGeometry::TexturedPoint2D структура является удобной структурой для доступа к двумерным точкам с координатами текстуры.

## *Программный интерфейс КМЛ для С++*

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-module.html>

### ВАЖНЫЕ С++ КЛАССЫ, КОТОРЫЕ ОБЕСПЕЧИВАЕТ QML МОДУЛЬ

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-cppclasses-topic.html>

qml модуль обеспечивает С++ классы, которые применяют qml каркас. Клиенты могут использовать эти классы для взаимодействия с qml во время выполнения, а также для инстанцирования иерархий объектов из qml документа. Типичное приложение qml с С++ точками входа будет инстанцировать [QQmlEngine](http://qt-project.org/doc/qt-5.1/qtqml/qqmlengine.html), а затем использовать [QQmlComponent](http://qt-project.org/doc/qt-5.1/qtqml/qqmlcomponent.html) для загрузки qml документа. Движок обеспечивает [QQmlContext](http://qt-project.org/doc/qt-5.1/qtqml/qqmlcontext.html) по умолчанию, который будет контекстом оценки верхнего уровня, который будет использоваться для оценки функций и выражений, определённых в qml документе. Иерархия объектов, определённая в QML документе, будет инстанцирована при помощи вызова функции [create()](http://qt-project.org/doc/qt-5.1/qtqml/qqmlcomponent.html#create) класса [QQmlComponent](http://qt-project.org/doc/qt-5.1/qtqml/qqmlcomponent.html), в предположении, что во время чтения документа не произошло никаких ошибок. Клиент может желать модифицировать [QQmlContext](http://qt-project.org/doc/qt-5.1/qtqml/qqmlcontext.html), обеспечиваемый движком, чтобы вносить новые свойства или объекты в контекст. Они могут вызвать функцию [QQmlEngine::rootContext](http://qt-project.org/doc/qt-5.1/qtqml/qqmlengine.html#rootContext)(), чтобы тогда получить доступ к контексту высокого уровня.

После инстанцирования объекта клиент обычно передаёт управление циклу обработки событий приложения, так что события ввода пользователя могут быть доставлены и обработаны приложением. Заметьте, что Qt Quick модуль предоставляет удобный класс, [QQuickView](http://qt-project.org/doc/qt-5.1/qtquick/qquickview.html), который обеспечивает qml время выполенния и визуальное окно для отображения qml приложения. [QQmlEngine](http://qt-project.org/doc/qt-5.1/qtqml/qqmlengine.html) класс обеспечивает движок, который может управлять иерархией объектов, которая определена в qml документе. Он предоставляет корневой контекст qml, в котором оцениваются выражения. Данный класс позволяет настраивать Глобальные настройки, которые применяются ко всем объектам, которыми он управляет. Например, QNetworkAccessManager может быть использован для коммуникации по сети, а путь файла может быть использован для постоянного хранения.

[QQmlContext](http://qt-project.org/doc/qt-5.1/qtqml/qqmlcontext.html) класс обеспечивает контекст для инстанцирования объекта и оценки выражений. Все объекты инстанцируются в частном контексте, а все выражения, которые оцениваются, также оцениваются в некотором частном контексте. Этот контекст определяет, как символы разрешаются и таким образом, какие значения выражения следует обрабатывать. Динамическое инстанцирование объектов и динамическая оценка выражений также являются ключевыми понятиями в qml. [QQmlComponent](http://qt-project.org/doc/qt-5.1/qtqml/qqmlcomponent.html) используется для динамически инстанцируемых объектов. Произвольные выражения могут быть рассчитаны в С++ при помощи класса [QQmlExpression](http://qt-project.org/doc/qt-5.1/qtqml/qqmlexpression.html), и такие выражения могут напрямую взаимодействовать с qml контекстом.

[QQmlComponent](http://qt-project.org/doc/qt-5.1/qtqml/qqmlcomponent.html) класс может быть использован для загрузки qml документов. Он требует [QQmlEngine](http://qt-project.org/doc/qt-5.1/qtqml/qqmlengine.html), чтобы инстанцировать иерархию объектов, определённую в qml документе. [QQmlExpression](http://qt-project.org/doc/qt-5.1/qtqml/qqmlexpression.html) предоставляет способ клиентам оценить java script выражения из С++, используя частный контекст оценки qml. Это позволяет клиентам получать доступ к qml объектам с помощью идентификатора. Результат оценки возвращается как [QVariant](http://qt-project.org/doc/qt-5.1/qtcore/qvariant.html), а правила преобразования определены qml движком.

### ИНТЕГРИРОВАНИЕ QML И С++

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-cppintegration-topic.html>

qml разработан так, что легко расширяется при помощи С++. Это возможно благодаря интеграции qml движка и мета-объектной системы qt. С++ объекты и значения могут быть включены напрямую в контекст (область видимости) загружаемых qml объектов с использованием контекстных свойств и контекстного объекта. Это достигается через [QQmlContext](http://qt-project.org/doc/qt-5.1/qtqml/qqmlcontext.html) класс, обеспечиваемый модулем qml, который показывает данные контексту qml компонента, позволяя вставлять данные из С++ в qml.

### ПОКАЗ АТРИБУТОВ С++ ТИПАМ QML

[*http://qt-project.org/doc/qt-5.1/qtqml/qtqml-cppintegration-exposecppattributes.html*](http://qt-project.org/doc/qt-5.1/qtqml/qtqml-cppintegration-exposecppattributes.html)

qml может быть легко расширен при помощи функциональности, доступной из С++ кода. Qml имеет способности анализировать экземпляры QObject через мета-объектную систему. Это значит, что любой qml код может иметь доступ к следующим членам экземпляров данного класса:

* свойства,
* методы (Q\_INVOKABLE),
* сигналы,
* перечисления (Q\_ENUMS).

Если необходимо, чтобы данный класс был использован в качестве отдельного типа в методе, или свойстве, то его следует зарегистрировать. *Есть ссылка на хорошие примеры использования С++ и qml совместно.*

Любые данные, перенесённые из С++ в qml, должны быть типа, поддерживаемого qml движком. Некоторые типы поддерживаются по умолчанию, а некоторые следует регистрировать. Также применяются некоторые правила владения при переносе данных.

* Любое свойство, которое можно записать, должно иметь сигнал уведомления, чтобы быть совместимым с qml. *Есть рекомендация относительно имени уведомления. Очень полезная (<property>Changed).*
* Если свойство должно быть фиксированным, что следует использовать CONSTANT.
* Свойства типа объекта возможно добавлять в qml, но только если объект был ранее уже зарегистрирован.
* Также можно передавать qml список свойств, но для этого следует применять класс [QQmlListProperty](http://qt-project.org/doc/qt-5.1/qtqml/qqmllistproperty.html). *Есть пример кода.*
* Любое свойство только для чтения объектного типа доступно из qml кода как группированное свойство. Оно может быть использовано для показа группы связанных свойств, которые описывают набор атрибутов для типа. *Показан очень понятный пример кода.*

Для qml доступны методы С++, если они объявлены при помощи макроса Q\_INVOKABLE() или если они являются публичными слотами. Если С++ метод имеет параметр типа QObject\*, то значение параметра может быть передано из qml с использованием идентификатора объекта или значения var java script. Qml поддерживает вызов перегруженных С++ функций.

* Значения, возвращаемые из С++ методов, преобразуются в java script значения в соответствии с числом и типом аргументов, которые обеспечиваются.
* Любой публичный сигнал из С++, доступен в qml коде. qml движок автоматически создаёт обработчик сигнала для любого сигнала производного от QObject класса, который используется из qml. *Есть пример кода.*
* Классы могут иметь несколько сигналов с одинаковым именем, но только последний из них доступен qml. Сигналы с одинаковым именем, но разными параметрами, неотличимы друг от друга.

### НАПИСАНИЕ РАСШИРЕНИЙ QML ПРИ ПОМОЩИ QT

<http://qt-project.org/doc/qt-5.1/qtqml/qml-extending-tutorial-index.html>

данное руководство показывает, как писать расширения qml с использованием С++.

#### СОЗДАНИЕ НОВОГО ТИПА QML

<http://qt-project.org/doc/qt-5.1/qtqml/tutorials-extending-chapter1-basics.html>

часто бывает необходимо создать новый частный тип qml, кроме тех, которые обеспечивает Qt Quick. *Создаётся класс, который наследует какой-либо из Qt Quick С++ классов, в котором декларируются некоторые свойства. Затем он используется в qml файле, который импортирует файл с кодом. Затем в функции main перед использованием данный тип регистрируется при помощи специальной функции.*

#### СОЕДИНЕНИЕ С С++ МЕТОДАМИ И СИГНАЛАМИ ИЗ QML

[*http://qt-project.org/doc/qt-5.1/qtqml/tutorials-extending-chapter2-methods.html*](http://qt-project.org/doc/qt-5.1/qtqml/tutorials-extending-chapter2-methods.html)

*понятный пример использования методов и сигналов С++ из qml.*

#### ДОБАВЛЕНИЕ СВЯЗЫВАНИЯ СВОЙСТВ

<http://qt-project.org/doc/qt-5.1/qtqml/tutorials-extending-chapter3-bindings.html>

связывание свойств является мощным механизмом qml, который позволяет значениям различных типов быть автоматически синхронизированными. *В данном случае в свойстве С++ класса создаётся уведомление. В классе для установки цвета проверяется, действительно ли изменился цвет, а затем испускается сигнал о том, что цвет изменился.*

#### ИСПОЛЬЗОВАНИЕ ТИПОВ ЧАСТНЫХ СВОЙСТВ

<http://qt-project.org/doc/qt-5.1/qtqml/tutorials-extending-chapter4-custompropertytypes.html>

если мы хотим создать свойство частного типа, то тип этого свойства также должен быть зарегистрирован qml движком, а вся остальная процедура работает как и прежде.

#### ИСПОЛЬЗОВАНИЕ СВОЙСТВ ТИПА СПИСКОВ

<http://qt-project.org/doc/qt-5.1/qtqml/tutorials-extending-chapter5-listproperties.html>

что касается списка свойств, то это я уже читал ранее. *Здесь непонятно пока, как добавлять элементы в список.*

#### НАПИСАНИЕ ПЛАГИНА РАСШИРЕНИЯ QML

<http://qt-project.org/doc/qt-5.1/qtqml/tutorials-extending-chapter6-plugins.html>

1. одним способом использования qml расширения является создание класса С++ и его регистрация в качестве типа qml, а затем использования в qml файле и использование данного qml файла в [QQuickView](http://qt-project.org/doc/qt-5.1/qtquick/qquickview.html).
2. Вторым способом является создание плагина, который был бы доступен qml движку. Тогда частные типы могут быть зарегистрированы в собственном пространстве имён, чтобы их можно было использовать и за рамками данного приложения.

*Для создания плагина создаётся класс, являющийся наследником некоторого заданного класса. Затем оформляется некоторым определённым образом файл проекта. Затем в директорию плагина добавляется файл* [qmldir](http://qt-project.org/doc/qt-5.1/qtqml/qtqml-modules-qmldir.html), который анализируется qml движком при загрузке плагина. Затем для загрузки плагина используется средство [qmlscene tool](http://qt-project.org/doc/qt-5.1/qtdoc/qtquick-qmlscene.html).

### ОПРЕДЕЛЕНИЕ ТИПОВ QML ИЗ С++

[*http://qt-project.org/doc/qt-5.1/qtqml/qtqml-cppintegration-definetypes.html*](http://qt-project.org/doc/qt-5.1/qtqml/qtqml-cppintegration-definetypes.html)

при расширении qml с помощью С++ кода, С++ класс может быть зарегистрирован при помощи системы типов qml. Кроме всего прочего qml предоставляет возможности использования его специфических свойств, таких как присоединённые свойства и свойства по умолчанию в С++. *Далее показано, как регистрировать инстанцированный объектный тип. Иногда регистрируемый класс не должен быть инстанцируемым (перечислены случаи).*

Синглетонный тип предоставляет свойства, сигналы и методы, которые доступны без ручного инстанцирования экземпляра объекта. Это удобный способ предоставить функциональность глобальных значений свойств. Эти типы не имеют связанного с ними qml контекста, так как они разделяются между всеми контекстами. Данные экземпляры принадлежат исключительно движку qml.

Многие функции регистрации типов qml требуют задания версий. Версии позволяют добавлять новые свойства и методы в типы, оставляя их совместимыми с предыдущими версиями. *Показан ясный пример кода создания новой версии типа. Теперь понятно, что такое REVISION.* Это позволяет создавать изменения в приложении без разрушения текущего приложения. Но при этом qml разработчикам следует запоминать все документы, которые были изменены между малыми версиями. Есть функция для создания версии неинстанцируемого типа.

В qml есть присоединённые свойства и обработчики сигнала, которые присоединяются к объекту и обеспечиваются присоединённым типом. Обычные свойства объекта обеспечиваются самим объектом. *Есть пример кода.* Из С++ можно обеспечить классы для присоединяемого типа и присоединяющегося типа. В присоединяющем типе должна быть реализация функции qmlAttachedProperties(), которая должна возвращать экземпляр класса присоединяемого типа, а в качестве аргумента данной функции передаётся объект, представляющий тип, к которому присоединяют. Присоединяемый объект не уничтожается, пока существует тот, к которому его присоединили. Также в присоединяющем типе следует вызывать специальный макрос, который показывает, что тип является присоединяющим. [QML\_DECLARE\_TYPEINFO](http://qt-project.org/doc/qt-5.1/qtqml/qqmlengine.html#QML_DECLARE_TYPEINFO)(). *Есть прекрасный пример кода, который демонстрирует данный механизм.*

Также С++ может запрашивать с помощью специальной функции объект присоединяемых свойств.

Модификатор свойства – это специальный qml тип объекта. Он влияет на свойство, к которому применяется. Есть два разных вида модификаторов свойств.

* Перехватчики значений свойств
* и источники значений свойств.

Первый используется для фильтрации или изменения значений, когда они записываются в свойство. В настоящий момент такой тип существует только как Behavior. Второй тип используется для обновления значения свойства в определённое время. Клиенты могут определить собственные типы источников значений свойств. *Затем показан пример кода с синтаксисом модификатора свойств. Клиенты не могут создавать собственные перехватчики свойств.*

Например, источниками значений свойств являются различные типы анимации свойств. Для создания источника свойства следует создать подкласс класса [QQmlPropertyValueSource](http://qt-project.org/doc/qt-5.1/qtqml/qqmlpropertyvaluesource.html), который реализует функциональность по присваиванию различных значений свойству в различные моменты времени. *Приводится хороший пример кода. а также описано, как движок обрабатывает источник значений свойств.* Также тип источника рассматривается в qml как обычный тип, чтобы его можно было использовать в других контекстах, нежели источник значений свойства. Любой класс QObject, который регистрируется как инстанцируемый тип, может иметь свойство по умолчанию для данного типа. Свойство по умолчанию – это свойство, к которому автоматически присваиваются дети объекта, если они не присваиваются некоторому определённому свойству. Свойство по умолчанию устанавливается при помощи макроса [Q\_CLASSINFO](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html#Q_CLASSINFO)() для класса с определённым "DefaultProperty" значением. *Есть хороший пример кода, который демонстрирует данное свойство.*

При построении интерфейса пользователя с использованием Qt Quick модуля, qml объекты, которые рисуются визуально, должны наследовать тип Item, так как в QtQuick это базовый класс для всех объектов. Он реализуется классом С++ [QQuickItem](http://qt-project.org/doc/qt-5.1/qtquick/qquickitem.html), который также обеспечивается Qt Quick модулем. Этот класс можно специализировать для добавления визуального типа, который можно интегрировать в qml интерфейса пользователя.

Для некоторых qml объектных типов может быть предпочтительно запаздывать с инициализацией некоторых данных, пока объект не будет создан и все его свойства не будут установлены. Например, в случае, если инициализация дорогая, или пока инициализация не может произойти, пока не инициализированы значения свойств. Для этих целей qml обеспечивает класс [QQmlParserStatus](http://qt-project.org/doc/qt-5.1/qtqml/qqmlparserstatus.html), который следует специализировать. Он содержит несколько методов, которые вызываются на определённых стадиях инстанцирования компонента. *Есть пример кода, как использовать данный класс.*

### ВСТРОЕННЫЕ С++ ОБЪЕКТЫ В QML ПРИ ПОМОЩИ СВОЙСТВ КОНТЕКСТА

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-cppintegration-contextproperties.html>

при загрузке qml объекта в С++ приложение может быть полезно напрямую вставить некоторые С++ данные, которые могут быть использованы из qml кода. это делает возможным, например, запросить С++ метод на вставленный объект, или использовать экземпляр С++ объекта в качестве модели данных для qml изображений. Возможность вставки С++ данных в qml объект делается возможной при помощи использования [QQmlContext](http://qt-project.org/doc/qt-5.1/qtqml/qqmlcontext.html) класса. Этот класс показывает данные в контекст qml объекта, так что на данные можно сослаться напрямую из области видимости qml кода. *Есть простой пример использования данного класса.*

Использование контекста не при инициализации приложения может привести к серьёзному упадку производительности, так как при этом переоцениваются всё выражения. Контекстные свойства могут иметь значения [QVariant](http://qt-project.org/doc/qt-5.1/qtcore/qvariant.html) или [QObject](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html)\*. Частные объекты могут быть напрямую изменены в qml. *Есть пример использования объекта из qml.*

Если qml хочет получать сигналы от контекстного объекта, то следует использовать тип Connections. *Есть пример кода.* Контекст может быть полезен для использования основанных на С++ моделей в qml представлении.

### ИНТЕГРИРОВАНИЕ С QML ОБЪЕКТАМИ ИЗ С++

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-cppintegration-interactqmlfromcpp.html>

Все типы qml являются [QObject](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html)-derived типами. Это означает, что движок qml использует qt мета-объектную систему для динамического инстанцирования любого объектного типа qml и проверяет созданные объекты.

Qml компонент может быть загружен в С++ при помощи классов [QQmlComponent](http://qt-project.org/doc/qt-5.1/qtqml/qqmlcomponent.html) или [QQuickView](http://qt-project.org/doc/qt-5.1/qtquick/qquickview.html). Первый загружает qml документ как объект, а второй является оконным подклассом, поэтому загружаемый объект будет также нарисован. Этот класс используется для интегрирования отображаемого документа qml в интерфейс пользователя. *Есть примеры использования каждого их этих классов. Также показано затем, как вызывать методы qml объекта. Затем показано, как получать доступ к дочерним объектам.*

Авторы рекомендуют не использовать С++ для доступа к глубоким дочкам qml объекта, так как это разрушает основной принцип qml: отделить логику приложения С++ от интерфейса пользователя.

*Затем показано, как получать доступ к свойствам объектов qml.*

Для модификации свойства объекта qml всегда используйте [QObject::setProperty](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html#setProperty)(), [QQmlProperty](http://qt-project.org/doc/qt-5.1/qtqml/qqmlproperty.html) или [QMetaProperty::write](http://qt-project.org/doc/qt-5.1/qtcore/qmetaproperty.html#write)(), чтобы движок осознал изменение свойства. *Есть пример плохого способа изменения свойства.*

Функция [QMetaObject::invokeMethod](http://qt-project.org/doc/qt-5.1/qtcore/qmetaobject.html#invokeMethod)() используется для запроса qml метода из С++. Есть пример кода. Qml сигналы автоматически доступны из С++ и могут быть связаны с использованием [QObject::connect](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html#connect)(). *Есть пример кода.*

Если параметром в qml является объект С++, то следует использовать тип var, а значение должно быть получено в С++ с использованием [QVariant](http://qt-project.org/doc/qt-5.1/qtcore/qvariant.html).

### ПРЕОБАРЗОВАНИЕ ДАННЫХ МЕЖДУ QML И С++

<http://qt-project.org/doc/qt-5.1/qtqml/qtqml-cppintegration-data.html>

когда происходит обмер данными между qml и С++, они преобразуются qml движком, чтобы иметь корректные типы данных, какие требуются для использования в qml или С++. Поэтому требуется, чтобы данные для обмена были такого типа, который распознаётся движком. Qml движок предоставляет встроенную поддержку для большого числа типов данных qt С++. Также можно зарегистрировать при помощи системы qml типов частные С++ классы, чтобы они могли быть доступны как типы в qml.

Когда данные переносятся из qml в С++, владение данными всегда остаётся у С++. Исключением из этого правила является тот случай, когда QObject возвращается из метода С++ явно: в этом случае qml движок предполагает владение объектом, если только владение объектом не было явно установлено, чтобы оставаться с С++ при помощи [QQmlEngine::setObjectOwnership](http://qt-project.org/doc/qt-5.1/qtqml/qqmlengine.html#setObjectOwnership)() с заданным [QQmlEngine::CppOwnership](http://qt-project.org/doc/qt-5.1/qtqml/qqmlengine.html#ObjectOwnership-enum). Вдобавок qml движок принимает во внимание нормальную семантику отношений наследования QObject qt С++ объектов и никогда не возьмёт владение экземпляра QObject, который уже имеет родителя. *Далее перечислены типы данных, которые распознаёт qml и автоматически конвертирует. Некоторые классы (перечислены) доступны только тогда, когда Qt Quick модуль включён. Также в данной части показана специальная функция для связи типов из С++ и qml.* Наследник QObject может быть использован как тип для обмена данными между qml и С++, обеспечивающий то, чтобы класс был зарегистрирован при помощи системы типов qml. Движок разрешает регистрацию как инстанцируемых, так и неинстанцируемых типов.

Qml движок имеет встроенную поддержку для преобразования некоторого числа qt типов в соответствующие java script типы, и наоборот, при переносе данных между qml и С++. При этом qml немного модифицирует родные прототипы java script объекта, включая в их число строку, дату и число, чтобы обеспечить дополнительные возможности. Qml движок обеспечивает автоматическое преобразование типов между QVariantList и java script массивами и между QVariantMap и java script объектами. Также значения могут быть созданы как объекты java script и преобразованы в соответствующие объекты С++. Для преобразования QDateTime и java script объекта Date *есть пример кода.*

Некоторые контейнерные типы С++ поддерживаются прозрачно qml как java script массивы. *Далее они перечислены.* Но между данными типами и java script типами есть некоторые семантические отличия, которые вытекают из специфики С++. Например, удаление элемента из Array приведёт к замещению значения значением, полученным из конструктора по умолчанию, а не неопределённым значением. Также С++ контейнеры поддерживают некоторые индексы, в результате чего при задании индекса более максимального целочисленного значения, доступ будет утерян. *Далее перечислены значения по умолчанию для каждого стандартного контейнерного типа в qml.*

Для использования частного перечисления как типа данных, его класс должен быть зарегистрирован и перечисление должно быть также объявлено при помощи макроса Q\_ENUMS(), чтобы зарегистрировать его в мета-объектной системе qt. *Есть пример кода использования.* Имена перечислений должны начинаться с большой буквы, чтобы быть использованными в qml. Чтобы использовать перечисление в качестве параметра сигнала или метода в qml, следует обеспечить, чтобы они все были зарегистрированы в мета-объектной системе qt. Или чтобы перечисление было бы одним из типов, объявленных в [Qt Namespace](http://qt-project.org/doc/qt-5.1/qtqml/collision-qt.html).

Также С++ сигнал может быть соединимым с qml функцией с использование [connect()](http://qt-project.org/doc/qt-5.1/qtqml/qtqml-syntax-signals.html#connecting-signals-to-methods-and-signals); а перечисление следует зарегистрировать при помощи [qRegisterMetaType](http://qt-project.org/doc/qt-5.1/qtcore/qmetatype.html#qRegisterMetaType)().

### С++ КЛАССЫ ДЛЯ QML

#### QJSEngine

<http://qt-project.org/doc/qt-5.1/qtqml/qjsengine.html>

QJSEngine класс обеспечивает среду для оценки java script кода. Есть функции для оценки java script сценария. *Есть пример кода.* Также можно возвращать Глобальный объект скриптового движка. Также класс предоставляет обрабатывать исключения сценария. При помощи класса можно создавать java script объект. *Есть примеры, демонстрирующие использование данного кода.*

#### QJSValue

<http://qt-project.org/doc/qt-5.1/qtqml/qjsvalue.html#details>

класс QJSValue действует как контейнер для qt/java script типов данных. *Далее перечислены типы, которые поддерживает данный класс. Также перечислены функции, которые следует использовать для создания тех или иных экземпляров данного класса. Есть методы для проверки того, является ли данное значение значением определённого типа.* Можно устанавливать свойства для объектов. *Есть пример кода.* Также можно вызывать функции объекта и т.д.

#### QJSValueIterator

<http://qt-project.org/doc/qt-5.1/qtqml/qjsvalueiterator.html#details>

класс QJSValueIterator используется в качестве java итератора для [QJSValue](http://qt-project.org/doc/qt-5.1/qtqml/qjsvalue.html). Конструктор данного класса принимает экземпляр последнего в качестве аргумента. Итератор используется для просмотра свойств объекта java script.

#### QQmlAbstractUrlInterceptor

<http://qt-project.org/doc/qt-5.1/qtqml/qqmlabstracturlinterceptor.html>

QQmlAbstractUrlInterceptor класс позволяет вам контролировать загрузку qml файлов. Данный класс дорабатывается разработчиками. Он позволяет изменять url перед их использованием qml движком.

#### QQmlApplicationEngine

<http://qt-project.org/doc/qt-5.1/qtqml/qqmlapplicationengine.html#details>

QQmlApplicationEngine класс обеспечивает удобный способ для загрузки приложения из одного qml файла. Он комбинирует классы [QQmlEngine](http://qt-project.org/doc/qt-5.1/qtqml/qqmlengine.html) и [QQmlComponent](http://qt-project.org/doc/qt-5.1/qtqml/qqmlcomponent.html), чтобы обеспечить удобный способ загрузки одного qml файла. Он также отображает некоторую центральную функциональность приложения qml, которую гибридное приложение С++/qml обычно контролирует из С++. *Есть пример кода.* В будущем данный класс будет иметь выборщик файлов. Поэтому сейчас для будущей совместимости не используйте символ «+» в структуре ваших qml файлов.

#### QQmlComponent

<http://qt-project.org/doc/qt-5.1/qtqml/qqmlcomponent.html#details>

QQmlComponent класс инкапсулирует определение qml компонента. *Показан хороший пример кода использования данного класса. Есть некоторые замечания по поводу того, если qml компонент использует ресурсы интернета.*

#### QQmlContext

[*http://qt-project.org/doc/qt-5.1/qtqml/qqmlcontext.html#details*](http://qt-project.org/doc/qt-5.1/qtqml/qqmlcontext.html#details)

класс QQmlContext определяет контекст внутри qml движка. Контексты разрешают показывать данные qml компонентам, инстанцированным qml движком. *Есть пример использования данного класса.* Также важно заметить, что именно Qt Creator должен следить за удалением контекста, который он создаёт. *Есть пример использования контекста, но пока я слабо представляю его основное назначение.* Контексты образуют иерархию. *Есть пример кода, демонстрирующий использование данного обстоятельства.* Установка объекта контекста или добавление новых свойств контекста после того, как объект уже был создан в данном контексте, является дорогой операцией. Поэтому, когда это возможно, вам следует завершать установку контекста перед использованием его, чтобы создавать объекты. *Пока что понятие контекст во многих смыслах его употребления, в том числе и в данном, мне не очень понятно.*

#### QQmlEngine

<http://qt-project.org/doc/qt-5.1/qtqml/qqmlengine.html#details>

класс QQmlEngine обеспечивает среду для инстанцирования qml компонентов. Каждый qml компонент инстанцируется в [QQmlContext](http://qt-project.org/doc/qt-5.1/qtqml/qqmlcontext.html). Последний является важным для передачи данных qml компонентам. В qml контексты организуются в иерархию и эта иерархия управляется движком qml. *Есть пример использования данного класса.*

#### QQmlError

<http://qt-project.org/doc/qt-5.1/qtqml/qqmlerror.html#details>

класс QQmlError инкапсулирует qml ошибку. Он включает текстовое описание ошибки, также как и место положения информации (файл, линию и строку). Ошибки можно выводить на консоль. *Есть пример использования кода. Пока не разобрался с техническими вопросами реализации данного класса.*

#### QQmlExpression

<http://qt-project.org/doc/qt-5.1/qtqml/qqmlexpression.html#details>

класс QQmlExpression оценивает java script в контексте qml. *Есть понятный пример использования данного класса.*

#### QQmlExtensionPlugin

<http://qt-project.org/doc/qt-5.1/qtqml/qqmlextensionplugin.html#details>

класс QQmlExtensionPlugin обеспечивает абстрактное основание для частных плагинов расширения qml. Это интерфейс плагина, который делает возможным создание qml расширений, которые можно загрузить динамически в qml приложения. Это позволяет делать доступными движку частные qml типы. *Затем описана процедура создания qml плагина. Есть некоторое замечание по поводу корневого контекста для библиотечных плагинов. Есть замечательный пример создания плагина.*

#### QQmlImageProviderBase

<http://qt-project.org/doc/qt-5.1/qtqml/qqmlimageproviderbase.html#details>

класс QQmlImageProviderBase используется для регистрации обеспечителя изображений в qml движке. Они должны быть зарегистрированы в qml движке. *Пока не понимаю предназначение данного класса.*

#### QQmlIncubationController

<http://qt-project.org/doc/qt-5.1/qtqml/qqmlincubationcontroller.html#details>

QQmlIncubationController экземпляры управляют прогрессом QQmlIncubators. Класс позволяет управлять, когда происходит создание объектов. Это важно, чтобы не было замерзания приложения. *Есть довольно понятный пример использования данного класса.*

#### QQmlListProperty

<http://qt-project.org/doc/qt-5.1/qtqml/qqmllistproperty.html#details>

класс QQmlListProperty позволяет приложениям показывать подобные спискам свойства qml. *Далее описан данный класс, но пока, без конкретных примеров, мне тяжело его понять. Также там указаны некоторые тонкости его использования.*

#### QQmlIncubator

<http://qt-project.org/doc/qt-5.1/qtqml/qqmlincubator.html#details>

QQmlIncubator класс позволяет создавать qml объекты асинхронно. Создание qml объектов может занимать значительное время. При использовании [QQmlComponent::create](http://qt-project.org/doc/qt-5.1/qtqml/qqmlcomponent.html#create)() объект создаётся синхронно. Использование данного класса предоставляет больше контроля процесса создания объекта. Например, можно создавать объекты асинхронно, пока приложение работает вхолостую. *Есть пример создания данного класса.* Асинхронные инкубаторы управляются [QQmlIncubationController](http://qt-project.org/doc/qt-5.1/qtqml/qqmlincubationcontroller.html), которые устанавливаются для [QQmlEngine](http://qt-project.org/doc/qt-5.1/qtqml/qqmlengine.html), который позволяет приложению знать, когда приложение работает в холостую, а когда инкубационные объекты следует обработать. *Затем перечислены три инкубационные моды, поддерживаемые данным классом.*

#### QQmlListReference

<http://qt-project.org/doc/qt-5.1/qtqml/qqmllistreference.html#details>

QQmlListReference класс позволяет манипулировать свойствами [QQmlListProperty](http://qt-project.org/doc/qt-5.1/qtqml/qqmllistproperty.html). Данный класс позовляет С++ программам считывать и присваивать значения qml спискам простым и безопасным с точки зрения типа способом. Не все qml свойства поддерживают все операции. Набор методов позволяет узнать, поддерживает ли данное свойство данный метод или нет.

#### QQmlNetworkAccessManagerFactory

<http://qt-project.org/doc/qt-5.1/qtqml/qqmlnetworkaccessmanagerfactory.html#details>

класс QQmlNetworkAccessManagerFactory создаёт QNetworkAccessManager для qml движка. Движок использует последний для доступа к сети. При помощи данной фабрики можно создавать частные QNetworkAccessManager с заданными кэшем, прокси и куки. *Затем сказано, как использовать данный класс.* Заметьте, что qml может управлять сетью из разных потоков, а поэтому метод create() должен быть многопоточным. Также следует быть осторожным с отправлением сигналов из данной функции, так как они могут быть связаны с обработчиками в других потоках. *Есть некоторые замечания по этому поводу.*

#### QQmlParserStatus

<http://qt-project.org/doc/qt-5.1/qtqml/qqmlparserstatus.html#details>

класс QQmlParserStatus обеспечивает обновления для qml состояния анализа. Данный класс предоставляет механизм для классов, инстанцированных при помощи [QQmlEngine](http://qt-project.org/doc/qt-5.1/qtqml/qqmlengine.html), для получения уведомления о ключевых моментах их создания. Он часто используется с целью оптимизации. *Есть хороший пример важности данного класса.* Методы данного класса вызываются, только когда класс инстанцирован qml движком. *Есть некоторые непонятные замечания по поводу данного класса. Есть пример кода использования данного класса. Его нужно будет понять более подробно. Его было бы хорошо поисследовать экспериментально.*

#### QQmlProperty

<http://qt-project.org/doc/qt-5.1/qtqml/qqmlproperty.html#details>

класс QQmlProperty абстрагирует доступ к свойствам объектов, созданных в qml. Этот класс является более удобным для qml свойств, нежели [QMetaProperty](http://qt-project.org/doc/qt-5.1/qtcore/qmetaproperty.html). *Есть хороший пример использования данного класса.*

#### QQmlPropertyMap

<http://qt-project.org/doc/qt-5.1/qtqml/qqmlpropertymap.html#details>

класс QQmlPropertyMap позволяет вам установить пары значений значение-ключ, которые могут использованы при qml связывании. Этот класс предоставляет удобный способ показа данных домена в слое интерфейса пользователя. *Есть пример использования кода, который пока мне непонятен. Было бы интересно поэкспериментировать с данным классом.*

#### QQmlPropertyValueSource

<http://qt-project.org/doc/qt-5.1/qtqml/qqmlpropertyvaluesource.html#details>

класс QQmlPropertyValueSource является интерфейсным для источников значения свойства, таких как анимации или связывание. *Далее идёт некоторая ссылка на подробное изучение данного класса.*

#### QQmlScriptString

<http://qt-project.org/doc/qt-5.1/qtqml/qqmlscriptstring.html#details>

класс QQmlScriptString инкапсулирует скрипт и его содержание. Он используется для создания свойств [QObject](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html), которые допускают присваивание скрипта из qml. *Есть примеры использования данного класса, но я также пока его не понимаю.*

### ТОЧКИ РАСШИРЕНИЯ С++, ОБЕСПЕЧИВАЕМЫЕ QT QUICK

<http://qt-project.org/doc/qt-5.1/qtquick/qtquick-cppextensionpoints.html>

Qt Quick модуль дополнительно обеспечивает несколько расширений и точек интеграции для разработчиков С++, специфических для этого модуля. В частности, он позволяет С++ разработчикам создавать и регистрировать частные производные от [QQuickItem](http://qt-project.org/doc/qt-5.1/qtquick/qquickitem.html) типы, которые могут быть нарисованы при помощи Qt Quick. Он также обеспечивает несколько связанных с графом декораций классов, которые позволяют разработчикам определить их собственные примитивы рисования. Для того чтобы создать через С++ частный визуальный тип, следует специализировать класс [QQuickItem](http://qt-project.org/doc/qt-5.1/qtquick/qquickitem.html). Использование графа декораций позволяет знать набор графических примитивов ещё до начала рисования.

Если необходимо загружать изображения из основанных на С++ процессах, то можно применить класс [QQuickImageProvider](http://qt-project.org/doc/qt-5.1/qtquick/qquickimageprovider.html), который обеспечивает поддержку для загрузки растров и помещает запросы изображения в отдельный поток для qml приложений. Любое qml приложение, которое требует изображение через специальную url схему изображений, будет направлен к подходящему провайдеру изображений, чтобы загрузить изображение. *Рассмотрим классы С++ для Qt Quick.*

#### QQuickImageProvider

<http://qt-project.org/doc/qt-5.1/qtquick/qquickimageprovider.html#details>

класс QQuickImageProvider обеспечивает интерфейс для поддержки растров и запросов изображений в потоке в qml. Он используется для продвинутой загрузки растров в qml приложениях. Он позволяет загружать изображения с использованием QPixmaps вместо действительных файлов изображений, а также асинхронную загрузку изображений в отдельном потоке. *Далее показано, как указать то, что изображение следует загрузить при помощи провайдера изображений. Далее показан пример загрузки изображения с использованием провайдера. Понятный пример. Наследование класса, переопределение некоторой функции и регистрация его в qml движке. Далее описан механизм асинхронной загрузки, и указаны некоторые ограничения применения данной особенности для целевых платформ.* Изображения, которые возвращаются этим классом, автоматически

кэшируются, так же как и изображения, загруженные при помощи qml движка.

#### QQuickItem

<http://qt-project.org/doc/qt-5.1/qtquick/qquickitem.html#details>

QQuickItem класс обеспечивает наиболее базовый из всех визуальных элементов Qt Quick. Все визуальные элементы в QtQuick наследуют от данного класса. Специализируя данный класс, можно создать свои собственные элементы в qt. Все визуальные элементы qml рисуются с использованием scene graph, низкоуровневого, высокопроизводительного стека рисования, близко связанного с OpenGL. Для подклассов данного класса возможно добавить некоторое своё частное содержание в граф при помощи установки [QQuickItem::ItemHasContents](http://qt-project.org/doc/qt-5.1/qtquick/qquickitem.html#Flag-enum) флага и переопределения [QQuickItem::updatePaintNode](http://qt-project.org/doc/qt-5.1/qtquick/qquickitem.html#updatePaintNode)() функции.

Критично, что OpenGL операции и взаимодействие с графом случается исключительно в потоке рисования, прямо во время вызова функции [updatePaintNode](http://qt-project.org/doc/qt-5.1/qtquick/qquickitem.html#updatePaintNode)(). Наиболее лучшим правилом является использование только классов с префиксом QSG внутри функции [QQuickItem::updatePaintNode](http://qt-project.org/doc/qt-5.1/qtquick/qquickitem.html#updatePaintNode)().

[QQuickPaintedItem](http://qt-project.org/doc/qt-5.1/qtquick/qquickpainteditem.html) позволяет рисовать с использованием [QPainter](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html). Использование графа всегда намного быстрее.

#### QQuickWindow

<http://qt-project.org/doc/qt-5.1/qtquick/qquickwindow.html#details>

класс QQuickWindow обеспечивает окно для отображения графики в qml. Класс обеспечивает управление графическими декорациями, необходимое для взаимодействия с и отображения декораций класса QQuickItem. Окно всегда имеет единственный невидимый корневой элемент. Класс использует граф декораций для рисования. Граф рисования живёт независимо и отдельно от qml графа, а поэтому может удаляться без влияния на последний. Возможно интегрировать напрямую OpenGL вызовы в данный класс с использованием такого же OpenGL контекста, что и граф. *Как это делается, указано в данной части. Далее описано, что происходит с графом, если мы прячем окно. Оказывается, граф можно тогда удалять, что приведёт к испусканию некоторого сигнала.* Qml будет пытаться кэшировать изображения и узлы графа, чтобы улучшить производительность, но в некоторых сценариях с малой памятью, можно потребовать агрессивно освобождать данные ресурсы. *Далее указаны функции для этого.*

#### QQuickTextDocument

<http://qt-project.org/doc/qt-5.1/qtquick/qquicktextdocument.html#details>

класс QQuickTextDocument обеспечивает доступ к [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html) из QQuickTextEdit. [QTextDocument](http://qt-project.org/doc/qt-5.1/qtgui/qtextdocument.html) используется внутри Qt Quick элементов, чтобы обеспечить манипуляцию текстовых примитивов. Вам не разрешается выполнять любую модификацию внутреннего состояния документа. если вы сделаете так, элемент может остановиться или обрушиться.

#### QQuickTextureFactory

<http://qt-project.org/doc/qt-5.1/qtquick/qquicktexturefactory.html#details>

QQuickTextureFactory класс обеспечивает интерфейс для загрузки частных текстур из qml. Целью данного класса является обеспечение указателя места для данных изображения, которые могут быть преобразованы в OpenGL текстуру. Создание текстуры напрямую невозможно, так как редко OpenGL контекст доступен в потоке, который ответственен за загрузку данных изображения.

#### QQuickView

<http://qt-project.org/doc/qt-5.1/qtquick/qquickview.html#details>

QQuickView класс обеспечивает окно для отображения Qt Quick интерфейса пользователя. Это удобный подкласс [QQuickWindow](http://qt-project.org/doc/qt-5.1/qtquick/qquickwindow.html), который будет автоматически загружать и отображать qml декорацию, когда дан url Главного файла исходного кода. Иначе вы можете инстанцировать ваши собственные объекты при помощи [QQmlComponent](http://qt-project.org/doc/qt-5.1/qtqml/qqmlcomponent.html) и поместить их в [QQuickWindow](http://qt-project.org/doc/qt-5.1/qtquick/qquickwindow.html). *Есть пример кода. Показано, как получать ошибки от данного файла. Также данный класс управляет увеличением размеров представления и корневого объекта. указаны некоторые функции.*

#### QQuickItem::ItemChangeData

<http://qt-project.org/doc/qt-5.1/qtquick/qquickitem-itemchangedata.html#details>

QQuickItem::ItemChangeData добавляет дополнительную информация для функции [QQuickItem::itemChange](http://qt-project.org/doc/qt-5.1/qtquick/qquickitem.html#itemChange)().

# ВИДЖЕТЫ

<http://qt-project.org/doc/qt-5.1/qtwidgets/qtwidgets-index.html>

qt виджет модуль обеспечивает набор графических элементов для создания классических десктопных интерфейсов пользователя. *Показан импорт модуля.*

Виджеты – это первичные элементы для создания интерфейсов пользователя в qt. Виджеты могут отображать данные и информацию о статусе, получать ввод пользователя, и обеспечивать контейнер для других виджетов, которые следует группировать вместе. Виджет, который не внедрён в родительский виджет, называется окном.

[QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html) класс обеспечивает базовую возможность рисования на экране и обработки ввода пользователя. Все графические элементы, которые обеспечивает qt, являются подклассами данного класса, или используются вместе с подклассом данного класса. Создание частных виджетов делается при помощи специализации [QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html) или другого подкласса, и переопределения виртуальных обработчиков событий. Стили рисуют от имени виджетов и инкапсулируют вид и ощущение интерфейса пользователя. Встроенные в qt виджеты используют класс [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html) для выполнения практически всего своего рисования, гарантируя, что они выглядят в точности похожими на эквивалентные нативные виджеты.

[Qt Style Sheets](http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet.html) является мощным механизмом, который позволяет вам настраивать внешность виджетов вдобавок к тому, что уже возможно при помощи специализации QStyle.

Макеты – это элегантный и гибкий способ автоматической организации дочерних виджетов внутри их контейнера. Каждый виджет сообщает его требования к размерам макету через свойства [sizeHint](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#sizeHint-prop) and [sizePolicy](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#sizePolicy-prop), а макет распространяет доступное пространство в соответствии с этим.

Архитектура модель/представление обеспечивает классы для управления способом, которым данные представляются пользователю. Приложения, основанные на данных, которые используют списки и таблицы, структурированы, чтобы отделить данные и представление, используя модели, представления и делегаты.

[Graphics View Framework](http://qt-project.org/doc/qt-5.1/qtwidgets/graphicsview.html) предназначен для управления и взаимодействия с большим числом самостоятельно сделанных двумерных графических элементов, и для виджетов для визуализации этих элементов, с поддержкой масштабирования и вращения.

## ИНТЕРФЕЙСЫ ПОЛЬЗОВАТЕЛЯ

<http://qt-project.org/doc/qt-5.1/qtdoc/topics-ui.html>

qt имеет несколько технологий для создания интерфейсов пользователя. Так как возможно перемешивать и сопоставлять эти разные технологии, если необходимо, один подход всегда является лучшим подходящим решением для некоторого определённого типа интерфейса пользователя, чем другие. [Qt Creator](http://qt-project.org/doc/qt-5.1/qtdoc/qtexamplesandtutorials.html#qt-creator) является хорошим примером приложения, которое совмещает разные технологии интерфейса пользователя. Действительно, он использует все три разные подхода, которые описываются ниже. Qt Creator использует традиционный [Qt Widgets](http://qt-project.org/doc/qt-5.1/qtwidgets/qtwidgets-index.html), такие как меню и диалоги, как базу для интерфейса пользователя. [Qt Quick](http://qt-project.org/doc/qt-5.1/qtquick/qtquick-index.html) используется для отображения окна приветствия. [Qt WebKit](http://qt-project.org/doc/qt-5.1/qtwebkit/qtwebkit-index.html) используется для представления qt документации со ссылками. Следующие секции обеспечивают ясное введение к доступным технологиям для создания интерфейсов пользователя и таблица сравнения для помощи в выборе лучшего подходящего решения.

Qml позволяет пользователям строить интерфейсы пользователя декларативным способом. Интерфейсы пользователя особенно предпочтительно относятся к простоте соединения qml объектов и настройкам их с использованием связывания свойств.

Qt Quick модуль обеспечивает qml типы для создания интерфейса пользователя, например, визуальную канву с её собственной координатной системой и движком рисования. Анимация и эффекты перехода являются первостепенными понятиями в Qt Quick и визуальные эффекты могут быть дополнены через специальные компоненты для эффектов частиц и шейдеров.

Qt обеспечивает [WebKit](http://www.webkit.org/)-based движок с поддержкой для широкого диапазона стандартных веб-технологий, таких как HTML, CSS и JavaScript, которые делают возможным внедрять содержание из веб в ваше приложение. [Qt WebKit](http://qt-project.org/doc/qt-5.1/qtwebkit/qtwebkit-index.html) отображает веб контент через [WebView](http://qt-project.org/doc/qt-5.1/qtwebkit/qml-qtwebkit3-webview.html) QML type, тогда как [Qt WebKit Widgets](http://qt-project.org/doc/qt-5.1/qtwebkit/qtwebkitwidgets-index.html) использует С++ программный интерфейс для рисования и взаимодействия с веб контентом. Главное отличие между двумя программными интерфейсами – это то, что [WebView](http://qt-project.org/doc/qt-5.1/qtwebkit/qml-qtwebkit3-webview.html) использует многопроцессорную архитектуру, [WebKit2](http://trac.webkit.org/wiki/WebKit2), вместе с отдельным процессом рисования веб. С++ API использует предыдущую версию WebKit, которая имеет однопроцессную архитектуру.

Заметьте, что выбор подходящей технологии для построения интерфейса пользователя не всегда является простым, но может зависеть от нескольких критериев, таких как другие существующие технологии, используемые в проекте или даже ограничения целевой платформы. В некоторой степени многие элементы в следующей таблице могут быть реализованы с использованием любой из трёх доступных технологий. Следующая таблица просто преследует цель помочь выбрать лучший подходящий инструмент для работы.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Qt Quick** | **Qt Widgets** | **Qt WebKit** | **Comments** |
| Используемые языки | QML/JS | C++ | HTML/CSS/JS |  |
| Нативный вид | Х | Х |  | Виджеты и QtQuick элементы управления хорошо интегрируются в нижележащую платформу (Windows, Linux, Mas OS) |
| Частный вид | X |  | (Х) | Виджеты обеспечивают средства для настраивания стилей, но QtQuick является лучшим выбором для интерфейсов пользователя, которые не преследуют целью нативный вид окон. |
| Динамически анимируемые графические интерфейсы | X |  |  | Виджеты не очень подходят для анимации, а Qt WebKit CSS3 анимации не такие мощные, как анимации QtQuick. Qt Quick обеспечивает очень простой способ создания анимаций. |
| Сенсорный экран | X |  |  | *Не важно сейчас для нас* |
| Стандартные промышленные виджеты |  | X |  | Виджеты обеспечивают все колокола и свистки, разработанные за два десятилетия, необходимые для построения стандартных промышленных типов приложений. WebKit виджеты обеспечивают виджеты и дополнительные классы для рисования и взаимодействуют с веб содержанием. |
| Модель/представление программирование | (X) | X |  | Некоторые мвс понятия существуют в WebKit, QtQuick обеспечивает удобные представления, но виджеты обеспечивают более удобные и полные возможности для этих целей. |
| Быстрая разработка интерфейса пользователя | X |  | (X) | Благодаря его превосходной производительности, QtQuick является прекрасным выбором для быстрого прототипирования и разработки интерфейсов пользователя. |
| HW ускоренная графика | X |  |  | Виджеты обеспечивают [QGLWidget](http://qt-project.org/doc/qt-5.1/qtopengl/qglwidget.html) для рисования OpenGL графики, а WebKit поддерживает WebGL, но доказано, что OpenGL ES 2.0 или OpenGL 2.0 основанный [Qt Quick Scene Graph](http://qt-project.org/doc/qt-5.1/qtquick/qtquick-visualcanvas-scenegraph.html), обеспечивает лучшую производительность для интерфейсов пользователя при интеграции с OpenGL контентом. |
| Графические эффекты | X |  |  | Система частиц и эффектов шейдеров, доступных в Qt Quick, являются более гибкими по сравнению с CSS эффектами и шейдерами, доступными в WebKit. Виджеты предоставляют очень мало возможностей в данной области. |
| Обработка форматированного текста | X | X |  | Виджеты в настоящий момент обеспечивают наиболее комплексную основу для реализации текстовых редакторов. Данные классы также могут быть использованы в Qt Quick, но при этом они требуют некоторую С++ реализацию. |
| Существующий веб контент |  |  | Х | И QtQuick, и виджеты обеспечивают компоненты для представления простого форматированного текста, но WebKit является правильным выбором для представления полномасштабного веб-контента. |

Qt предоставляет превосходную поддержку для перевода на локальные языки.

## ВИДЖЕТЫ ОКОН И ДИАЛОГОВ

<http://qt-project.org/doc/qt-5.1/qtwidgets/application-windows.html>

окном называется виджет, который не внедрён в другой виджет. В qt наиболее общими типами окон являются классы [QMainWindow](http://qt-project.org/doc/qt-5.1/qtwidgets/qmainwindow.html), различные подклассы [QDialog](http://qt-project.org/doc/qt-5.1/qtwidgets/qdialog.html). В приложениях окна обеспечивают пространство экрана, на котором строится пользовательский интерфейс. Окна отделяют приложения визуально одно от другого и обычно обеспечивают украшение окна, которое позволяет пользователю изменять размер или положение приложений в соответствии с его предпочтениями. Окна обычно интегрированы в окружение рабочего стола и в некоторой степени управляются системой управления окон, которую обеспечивает окружение рабочего стола. Например, выбранные окна приложения представляются на панели задач. [QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html), который не имеет родителя, станет окном и будет на большинстве платформ перечислен в панели задач окружения рабочего стола. *Обычно хочется, чтобы только одно окно в приложении стало первичным окном.* Вдобавок, виджет, который имеет родителя, также может быть сделан окном при помощи установки флага [Qt::Window](http://qt-project.org/doc/qt-5.1/qtcore/qt.html#WindowType-enum). В зависимости от системы управления окон такие вторичные окна обычно находятся сверху окон их родителей и не имеют панели задач сами по себе.

[QMainWindow](http://qt-project.org/doc/qt-5.1/qtwidgets/qmainwindow.html) класс разработан, чтобы быть окном. Он имеет некоторые средства, которые нежелательны для виджета. [QMainWindow](http://qt-project.org/doc/qt-5.1/qtwidgets/qmainwindow.html) имеет свой собственный макет, в который вы можете добавить

* панель инструментов,
* меню,
* закрепляемые виджеты и
* строку состояния.

Диалоговые окна используются как вторичные окна, которые представляются пользователю вместе с настройками и выборами. Также qt содержит некоторое количество стандартных диалогов, которые можно использовать для стандартных задач. Диалоги и окна лучше создавать при помощи дизайнера.

[QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html) предоставляет несколько функций, которые имеют дело с геометрией окна. Некоторые из них работают с чистой клиентской областью, другие включают фрейм окна. *Есть хорошая иллюстрация, которая демонстрирует отличия данных типов функций.*

ВАЖНО!!! *В конце данной части сообщается о проблемах окон в оконной системе X11. Это критично и важно для меня.*

## РУКОВОДСТВО ПО ВИДЖЕТАМ

<http://qt-project.org/doc/qt-5.1/qtwidgets/widgets-tutorial.html>

виджеты – это базовые строительные блоки для графического интерфейса пользователя, построенного при помощи qt. Каждый графический компонент является виджетом, который размещается где-то внутри окна интерфейса пользователя, или отображается как независимое окно.

[QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html) не является абстрактным классом. Он может быть использован как контейнер для других виджетов, а также он может быть специализирован с небольшим усилием для создания нового виджета. Также как и [QObject](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html), данный класс может быть создан при помощи родительского объекта, чтобы показать владельца, который гарантирует, что объекты будут удалены, когда они более не используются. Что касается виджетов, то это правило имеет некоторое дополнительное значение: дочерние виджеты размещаются внутри области их родительского виджета. Это означает, что при удалении родительского виджета, также удаляются и его дочерние виджеты.

Многие из GUI примеров, обеспечиваемых при помощи qt, следуют некоторому типовому решению, в соответствии с которым в проекте имеется файл main.cpp, который содержит стандартный код для инициализации приложения, плюс некоторое количество других исходных/заголовочных файлов, которые содержат логику приложения и некоторые

частные компоненты графического интерфейса пользователя.

## ГЛАВНОЕ ОКНО ПРИЛОЖЕНИЯ

<http://qt-project.org/doc/qt-5.1/qtwidgets/mainwindow.html>

Qt предоставляет следующие классы для управления Главными окнами и связанными с ними компонентами интерфейса пользователя.

* [QMainWindow](http://qt-project.org/doc/qt-5.1/qtwidgets/qmainwindow.html) – центральный класс, вокруг которого строятся приложения.
* [QDockWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qdockwidget.html) обеспечивает виджет, который может быть использован, чтобы создать панели инструментов или вспомогательные окна. Встроенные виджеты следят за своими собственными свойствами, а также они могут двигаться, закрываться или плавать как внешние окна.
* [QToolBar](http://qt-project.org/doc/qt-5.1/qtwidgets/qtoolbar.html) обеспечивает панель инструментов, которую можно удерживать на нескольких связанных с действиями виджетов, таких как кнопки, всплывающие меню, выпадающие списки, а также счётчики. Акцент в qt на модель унифицированных действий означает, что панели инструментов хорошо кооперируются вместе с меню и горячими клавишами.

*Далее показан пример кода, исходя из которого можно проследить создание Главного окна приложения.*

#### QAction

<http://qt-project.org/doc/qt-5.1/qtwidgets/qaction.html#details>

QAction класс обеспечивает абстрактный интерфейс действий пользователя, которые могут быть вставлены в виджеты. В приложениях многие команды могут быть запрошены через меню, кнопки на панели инструментов или горячие клавиши клавиатуры. Так как пользователь ожидает, что каждая команда будет использована одинаковым способом, независимо от используемого интерфейса пользователя, полезно представить каждую команду через действие. Действия могут быть созданы как независимые объекты, но они также могут быть созданы и внутри меню. QAction может содержать иконку, текст меню, горячую клавишу, текст статуса, подсказки. Действия следует добавлять в виджет перед их использованием. *Далее приводится пример кода использования данного класса.*

Рекомендуется использовать действия как дочерние классы окна, в котором они используются. В большинстве случаев они должны быть детьми Главного окна приложения.

#### QActionGroup

<http://qt-project.org/doc/qt-5.1/qtwidgets/qactiongroup.html#details>

QActionGroup класс группирует действия вместе. *Далее показан пример кода использования данного класса, а также перечислены важные функции для его использования.*

#### QWidgetAction

<http://qt-project.org/doc/qt-5.1/qtwidgets/qwidgetaction.html#details>

QWidgetAction класс расширяет [QAction](http://qt-project.org/doc/qt-5.1/qtwidgets/qaction.html) при помощи интерфейса для вставки частных виджетов в основанные на действиях контейнерах, таких как панели инструментов. Большинство действий в приложениях представляются в виде элементов меню или кнопок в панели инструментов. Однако иногда необходимы более сложные виджеты. Далее приводятся различные аспекты использования данного класса.

#### QDockWidget

[*http://qt-project.org/doc/qt-5.1/qtwidgets/qdockwidget.html#details*](http://qt-project.org/doc/qt-5.1/qtwidgets/qdockwidget.html#details)

QDockWidget класс обеспечивает виджет, который может быть закреплён в [QMainWindow](http://qt-project.org/doc/qt-5.1/qtwidgets/qmainwindow.html) или плавать как окно высокого уровня на рабочем столе. Данная концепция также известна как палитра инструментов или окна утилит. Это вторичные окна, помещаемые в специальной области окна вокруг центрального виджета. QDockWidget состоит из заголовка и области содержания. Он действует как обёртка для его дочерних виджетов. *Также в данной части описаны некоторые аспекты использования данного класса.*

#### QMainWindow

<http://qt-project.org/doc/qt-5.1/qtwidgets/qmainwindow.html#details>

QMainWindow класс обеспечивает Главное окно приложения. *Далее показана замечательная схема устройства данного окна.* Создание Главного окна без центрального виджета не поддерживается. Главное окно может иметь как одиночный так и множественный интерфейс документа. последние приложения создаются в qt при помощи использования класса [QMdiArea](http://qt-project.org/doc/qt-5.1/qtwidgets/qmdiarea.html) в качестве центрального виджета. *Далее приводятся примеры создания меню, панели инструментов, доковых виджетов, панелей задач.* Также есть специальная функция для сохранения состояния Главного окна приложения. Это положения и размеры панелей инструментов и доковых виджетов, которые сохраняются.

#### QMdiArea

<http://qt-project.org/doc/qt-5.1/qtwidgets/qmdiarea.html#details>

QMdiArea класс обеспечивает область, в которой мди окна отображаются. *Написан код добавления данной области в окно. Далее разговор ведётся о некоторых функциях, обеспечиваемых данным классом. Следует изучит данный класс экспериментально, чтобы вникнуть в его суть.*

#### QMdiSubWindow

<http://qt-project.org/doc/qt-5.1/qtwidgets/qmdisubwindow.html#details>

QMdiSubWindow класс обеспечивает дочернее окна для класса [QMdiArea](http://qt-project.org/doc/qt-5.1/qtwidgets/qmdiarea.html). Экземпляр данного класса состоит из заглавия вместе с декорациями окна, а также внутреннего виджета, а также фрейма окна и захвата для изменения размера. *Далее описаны некоторые аспекты использования и управления данными окнами.*

#### QMenu

<http://qt-project.org/doc/qt-5.1/qtwidgets/qmenu.html#details>

QMenu класс обеспечивает виджет меню для использования в меню, контекстных меню, а также других всплывающих меню. Меню состоит из списка действий. Есть четыре вида действий: разделители, действия, которые разрешают подменю, виджеты, а также действия для выполнения действия. При вставке действия вы обычно определяете слот и получателя. Также данный класс может быть использован для создания отрывных меню: окон, в которых содержится точная копия меню. Чтобы сделать меню видимым на экране, следует использовать функции [exec](http://qt-project.org/doc/qt-5.1/qtwidgets/qmenu.html#exec)() или [popup](http://qt-project.org/doc/qt-5.1/qtwidgets/qmenu.html#popup)() вместо [show](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#show)(). *Далее представлены некоторые замечания касательно использования меню на некоторых платформах.*

#### QSizeGrip

<http://qt-project.org/doc/qt-5.1/qtwidgets/qsizegrip.html#details>

QSizeGrip класс обеспечивает изменение размера для окон верхнего уровня. *Есть некоторые нюансы использования данного средства на X11.* Далее рассказывается, как использовать данный класс. *Его также следует изучить экспериментально, так как он может быть очень полезен.*

#### QStatusBar

<http://qt-project.org/doc/qt-5.1/qtwidgets/qstatusbar.html#details>

класс QStatusBar обеспечивает горизонтальную строку, доступную для представления информации о статусе. Каждый индикатор статуса может подпадать в одну из трёх категорий:

* Временный – явно заполняет большинство строк состояния. Используется для объяснения подсказок или вводов меню.
* Нормальный – заполняет часть строки состояния и может быть спрятан временными индикаторами статуса.
* Постоянный – никогда не прячется. Используется для указания на важную моду.

Данный класс позволяет вам отображать все три данных типа индикаторов. *Далее довольно хорошо описаны основные аспекты использования данного класса.*

#### QToolBar

<http://qt-project.org/doc/qt-5.1/qtwidgets/qtoolbar.html#details>

класс QToolBar обеспечивает передвигаемую панель, которая содержит набор элементов управления.

## ДИАЛОГОВЫЕ ОКНА

[*http://qt-project.org/doc/qt-5.1/qtwidgets/dialogs.html*](http://qt-project.org/doc/qt-5.1/qtwidgets/dialogs.html)

диалоги могут быть модальными, когда от пользователя требуется обеспечить необходимую информацию перед тем, как работа в Главном окне может быть продолжена, или немодальными. Немодальные диалоги не мешают пользователю взаимодействовать с любым другим окном в приложении. *Далее предоставляется список готовых диалогов qt.* Частные диалоги могут быть легко созданы при помощи собирания обычных виджетов в [QDialog](http://qt-project.org/doc/qt-5.1/qtwidgets/qdialog.html). Эти классы специфически разработаны для построения частных диалогов: *далее они приводятся в отдельной таблице. Рассмотрим все приведённые в данной части классы.*

#### QColorDialog

<http://qt-project.org/doc/qt-5.1/qtwidgets/qcolordialog.html#details>

класс QColorDialog обеспечивает виджет диалог для задания цветов. *Далее приводится описание данного класса.*

#### QFileDialog

<http://qt-project.org/doc/qt-5.1/qtwidgets/qfiledialog.html#details>

класс QFileDialog обеспечивает диалог, который позволяет пользователям выбирать файлы или директории. *Показан хороший способ использования данных диалогов. Также показано, как создавать множественные фильтры. Данный класс понятен. Если понадобится, то я его изучу.*

#### QFontDialog

<http://qt-project.org/doc/qt-5.1/qtwidgets/qfontdialog.html#details>

QFontDialog класс обеспечивает диалог для выбора шрифта. *Есть пояснение, есть иллюстрация и несколько примеров кода.*

#### QInputDialog

<http://qt-project.org/doc/qt-5.1/qtwidgets/qinputdialog.html#details>

QInputDialog класс обеспечивает простой удобный диалог для получения единственного значения от пользователя. *Есть примеры кода и иллюстрации.*

#### QMessageBox

<http://qt-project.org/doc/qt-5.1/qtwidgets/qmessagebox.html#details>

QMessageBox класс обеспечивает модальный диалог для информирования пользователя или для запроса пользователя и получения ответа. *Вот этот класс я изучу, так как он очень полезен для меня.* Для использования данного класса предоставляются два вида программных интерфейсов: основанный на свойствах (рекомендуется) и статические функции.

Для использования интерфейса, основанного на свойствах, создайте диалог, установите его свойства и вызовите функцию [exec](http://qt-project.org/doc/qt-5.1/qtwidgets/qmessagebox.html#exec)(), чтобы показать сообщение. Пользователя также можно спросить, что ему делать на счёт этого. Кнопки устанавливаются при помощи комбинирования значений из [StandardButtons](http://qt-project.org/doc/qt-5.1/qtwidgets/qmessagebox.html#StandardButton-enum) с использованием оператора ИЛИ. Порядок отображения для кнопок является зависимым от платформы. Также можно установить кнопку о показе деталей касательно данного диалога. *Есть примеры кода использования данного диалога.* Детальный текст всегда интерпретируется как простой текст. Главный текст и информативный текст могут быть как простым текстом, так и форматированным. Данные строки интерпретируются согласно с установками свойства [text format](http://qt-project.org/doc/qt-5.1/qtwidgets/qmessagebox.html#textFormat-prop). *Есть также некоторый редкий случай, который стоит принимать во внимание.* Также можно для сообщений выбрать четыре типа иконок. *Показан пример использования статических функций. Можно создавать свои собственные кнопки. Показан пример кода.*

*В конце сказано про кнопку по умолчанию и кнопку выхода. Зачем они нужны, мне пока не совсем понятно.*

#### QProgressDialog

<http://qt-project.org/doc/qt-5.1/qtwidgets/qprogressdialog.html#details>

QProgressDialog класс обеспечивает обратную связь с прогрессом медленных операций. Данный класс самостоятельно оценивает продолжительность операции и показывает себя, если она оказывается ниже чем значение от функции [minimumDuration](http://qt-project.org/doc/qt-5.1/qtwidgets/qprogressdialog.html#minimumDuration-prop)(). Количество шагов может быть выбрано произвольно. Это может быть количество байтов или количество файлов, или количество итераций. Диалог автоматически себя устанавливает и прячет при окончании операции. Но если вы установите новый максимум, то диалог не закроется. Есть два способа использования данного диалога: модальный и немодальный. Модальный диалог проще для программиста. *Есть хороший пример использования данного типа диалога.* Немодальный диалог подходит для операций, которые имеют место на фоне, когда пользователь способен взаимодействовать с приложением. Такие операции типично основаны на таймере, сокет уведомителе или url операторе, или выполняются в отдельном потоке. Альтернативой данного диалога является [QProgressBar](http://qt-project.org/doc/qt-5.1/qtwidgets/qprogressbar.html) в строке состояния вашего Главного окна. *Далее приведён пример использования немодального диалога.* Также в данном диалоге предусмотрены различные функции для настройки вида данного диалога.

#### QDialog

<http://qt-project.org/doc/qt-5.1/qtwidgets/qdialog.html#details>

QDialog класс является основным классом для окон диалога. Диалоговое окно является окном верхнего уровня, которое обычно используется для выполнения некоторых коротких задач и ясных коммуникаций с пользователем. Диалоги могут быть как модальными так и немодальными. Они могут обеспечить возвращаемое значение, а также кнопки по умолчанию. Также они могут иметь захват для изменения размера. *Этот класс отличается от всех остальных классов qt тем, что он находится сверху от своего родителя.*

Модальным диалогом является диалог, который блокирует ввод в любые другие видимые окна в том же приложении. Диалоги могут быть модальными по отношению к приложению или по отношению к окну. Наиболее общим способом отображения модального диалога является функция [exec](http://qt-project.org/doc/qt-5.1/qtwidgets/qdialog.html#exec)(), которая возвращает некоторое значение после завершения работы диалога. Либо же можно установить флаг модальности и показать диалог. Немодальный диалог – это диалог который работает независимо от других окон в данном приложении. Данные диалоги показываются при помощи функции show(). *Также показывается, как открывать диалоги в одной и той же позиции.* Кнопка по умолчанию диалога – это кнопка, которую нажимает пользователь при нажатии кнопки Enter. Если пользователь нажимает кнопку отмены в диалоге, то вызывается функция [QDialog::reject](http://qt-project.org/doc/qt-5.1/qtwidgets/qdialog.html#reject)().

Расширяемость – это способность показывать диалог двумя способами: только с наиболее основными опциями и со всеми опциями. *Есть ссылка на пример такого диалога.* Модальные диалоги часто используются в ситуациях, когда необходимо возвращать некоторое значение. *Показано как переопределить поведение вашего диалога при его закрытии. В конце приведены примеры для написания кода для модального и немодального диалога.*

#### QDialogButtonBox

<http://qt-project.org/doc/qt-5.1/qtwidgets/qdialogbuttonbox.html#details>

QDialogButtonBox класс является виджетом, который представляет кнопки в макете, которые подходят для текущего стиля виджета. Большинство кнопок диалога следуют определённым ролям:

* Принятие или отмена диалога.
* Запрашивание помощи.
* Выполнение действия в самом диалоге.

*Далее идут примеры, которые замечательно иллюстрируют предназначение данного класса. Он задаёт конфигурации кнопок в диалоге в специфическом для данной платформы стиле.* В данный ящик можно добавлять как свои кнопки, так там имеются и уже существующие кнопки.

## ДЕСКТОП ИНТЕГРАЦИЯ

<http://qt-project.org/doc/qt-5.1/qtdoc/desktop-integration.html>

приложения qt хорошо себя ведут в окружении рабочего стола, но некоторые интеграции требуют дополнительных, иногда специфических для платформы методов. *далее приводятся удобные классы для данных целей.*

Иногда необходимо также открывать внешние ресурсы и файлы с использованием внешних приложений. [QDesktopServices](http://qt-project.org/doc/qt-5.1/qtgui/qdesktopservices.html) обеспечивает интерфейс для служб, предоставляемых окружением рабочего стола пользователя.

Многие современные среды рабочего стола имеют доки или панели с системными лотками, в которых приложения могут устанавливать иконки. Иногда в этой иконке отображается статусная информация о приложении. Вдобавок многие приложения обеспечивают всплывающие меню, которые могут быть доступны из данных иконок. [QSystemTrayIcon](http://qt-project.org/doc/qt-5.1/qtwidgets/qsystemtrayicon.html) класс отображает всё об особенностях через интуитивный программный интерфейс в стиле qt, который может быть использован для рабочего стола всех платформ.

На системах, где отображается рабочий стол с использованием более одного экрана, определённые типы приложений могут нуждаться в получении информации о конфигурации рабочего пространства пользователя, чтобы гарантировать, что новые окна и диалоги откроются в подходящем месте. [QDesktopWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qdesktopwidget.html) класс может быть использован для проверки положений виджетов и уведомления приложений об изменениях в способе, которым рабочий стол делится в нескольких экранах. Это предоставляет приложениям возможность применять политики для размещения новых окон так, чтобы, например, они отвлекали пользователя, который работает с определённой задачей.

#### QDesktopServices

<http://qt-project.org/doc/qt-5.1/qtgui/qdesktopservices.html#details>

QDesktopServices класс обеспечивает методы для доступа к общим службам рабочего стола. Функция [openUrl](http://qt-project.org/doc/qt-5.1/qtgui/qdesktopservices.html#openUrl)() используется для открытия файлов, расположенных в произвольном url во внешних приложениях. Для url в локальной системе для открытия файла будет использовано соответствующее приложение. Для удалённых адресов будет использован браузер. Поведение функции [openUrl](http://qt-project.org/doc/qt-5.1/qtgui/qdesktopservices.html#openUrl)() может быть настроено для индивидуальных url схем, чтобы разрешить приложениям переписывать обработку по умолчанию некоторых типов url. Механизм диспетчеризации позволяет использовать только один частный обработчик для одной схемы url. Это устанавливается с использованием функции [setUrlHandler](http://qt-project.org/doc/qt-5.1/qtgui/qdesktopservices.html#setUrlHandler)(). Каждый обработчик применяется как слот, который допускает только один аргумент [QUrl](http://qt-project.org/doc/qt-5.1/qtcore/qurl.html). Данная система позволяет легко применять справочную систему. Справка может быть реализована при помощи меток и текстовых браузеров с использованием url **help://myapplication/mytopic** или при помощи регистрации обработчика становится возможно отображать справочный текст внутри приложения. *Есть хороший пример кода.*

#### QDesktopWidget

<http://qt-project.org/doc/qt-5.1/qtwidgets/qdesktopwidget.html#details>

QDesktopWidget класс обеспечивает доступ к информации об экране в системах с многими частями. В системах с более чем одной графической картой или монитором можно управлять физическим пространством, доступным либо через множественные рабочие столы, либо через один большой виртуальный рабочий стол. *Далее я не читал данную часть, так как пока данная часть мне не нужна.*

#### QSystemTrayIcon

<http://qt-project.org/doc/qt-5.1/qtwidgets/qsystemtrayicon.html#details>

QSystemTrayIcon класс обеспечивает иконку для приложения в системном лотке. *Есть хорошая иллюстрация того, что называется системным лотком. Далее перечислены платформы, на которых может использоваться данный класс. Далее речь идёт в данной части о некоторых функциях данного класса, а также о некоторых аспектах его использования.*

## ФОКУС ВВОДА КЛАВИАТУРЫ В ВИДЖЕТАХ

<http://qt-project.org/doc/qt-5.1/qtwidgets/focus.html>

qt виджеты обрабатывают фокус клавиатуры теми же способами, которые стали обычными в GUI. Основной вопрос состоит в том, что нажатия клавиш могут быть направлены к любому из окон на экране и к любому из нескольких виджетов, находящихся внутри того окна. Когда пользователь нажимает клавишу, то он ожидает, что она отправится в правильное место. А программное обеспечение должно удовлетворить данное ожидание. Поэтому система должна определить, какому приложению, какому окну и какому виджету следует отправить данное событие от клавиатуры. Существует несколько способов помещения фокуса ввода на виджет:

* Пользователь нажимает клавишу tab.
* Пользователь кликает на виджет.
* Пользователь нажимает на горячую клавишу.
* Пользователь использует колесо мыши.
* Пользователь передвигает фокус на окно, и приложение должно определить, виджет в данном окне.

Tab используется для обхода виджетов в окне в некотором направлении, а shift+tab - в обратном. Порядок, в котором нажатия tab передвигают от виджета к виджету, называется порядком табуляции. *Далее описаны методы, при помощи которых можно пропускать некоторые виджеты при их обходе.* В названии метки может быть горячая клавиша, которая передвигает фокус ввода к данному полю. *Далее описывается, как быть в тех случаях, когда необходимо в сам виджет вводить табуляцию.* Что касается использования кликов пользователя по нужному виджету, то это также мощный механизм, но есть одно очень важное исключение, которое заключается в том, что в некоторых случаях нежелательно забирать фокус ввода у тех приложений, у которых он был до этого. Авторы рекомендуют использовать данный тип перемещения только для виджетов, которые используют текстовый ввод. В qt только функция [QWidget::setFocusPolicy](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#focusPolicy-prop)() виляет на установку фокуса по клику пользователя. При нажатии горячих клавиш фокус может передаваться неявно, если инициируются некоторые диалоговые окна, или явно, при помощи [QLabel::setBuddy](http://qt-project.org/doc/qt-5.1/qtwidgets/qlabel.html#setBuddy)(), [QGroupBox](http://qt-project.org/doc/qt-5.1/qtwidgets/qgroupbox.html), and [QTabBar](http://qt-project.org/doc/qt-5.1/qtwidgets/qtabbar.html). Авторы рекомендуют поддержку горячих клавиш для всех виджетов, на которые пользователь может захотеть перепрыгнуть. Такой способ называется ускорителями фокуса. Что касается использования колёсика мыши, то здесь всё зависит от той платформы, на которой установлена система. *Некоторое описание отличий сообщается в данной части.*

Если окно уже было использовано, то при получении фокуса ввода, он устанавливается на тот виджет, который был использован в последний раз. Если же окно используется впервые, то, прежде чем показать окно, следует использовать функцию [QWidget::setFocus](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#setFocus)() для установки фокуса ввода.

*Теперь переходим к изучению довольно большой части данного модуля, которая посвящена стилям.*

## СТИЛИ И СТИЛИЗОВАННЫЕ ВИДЖЕТЫ

<http://qt-project.org/doc/qt-5.1/qtwidgets/style-reference.html>

стили рисуют от имени виджетов и инкапсулируют вид и ощущение графического интерфейса пользователя. [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html) класс является абстрактным классом, который инкапсулируют вид и ощущение графического интефрейса пользователя. Встроенные виджеты qt используют данный класс для выполнения практически всего рисования, гарантируя, что они будут выглядеть в точности как и эквивалентные виджеты. В qt есть встроенные стили. Частные стили создаются при помощи плагинов или при помощи создания экземпляра класса [QStyleFactory::create](http://qt-project.org/doc/qt-5.1/qtwidgets/qstylefactory.html#create)() и установки его при помощи функции [QApplication::setStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qapplication.html#setStyle)(). Для того чтобы настроить существующие стили, наследуйте [QProxyStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qproxystyle.html) и переопределите желаемые методы. [QCommonStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qcommonstyle.html) обеспечивает удобный базовый класс для всех реализаций частных стилей. Подход является таким же, как и в классе [QProxyStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qproxystyle.html). *Затем перечислены классы для настройки стиля приложения.*

API класса [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html) содержит

* функции для рисования виджетов,
* статические вспомогательные функции, чтобы делать общие и сложные задачи, и
* функции*,* чтобы делать различные вычисления, необходимые во время рисования.

Данный класс рисует графические элементы: элемент – это виджет, или часть виджета, например кнопка или фрейм окна или полоса прокрутки. Большинство функций рисования имеют четыре аргумента:

* Элемент перечисления, определяющий, какой графический элемент следует рисовать.
* [QStyleOption](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleoption.html), определяющий, как и где рисовать этот элемент.
* [QPainter](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html), который следует использовать для рисования элемента.
* [QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html), в котором происходит рисование.

Использование второго аргумента обеспечивает класс, который содержит информацию, необходимую для рисования. Благодаря этому классу возможно сделать виджеты, рисуемые при помощи QStyle без связывания с любым кодом для виджета. Это делает возможным использование функций рисования класса QStyle на любом устройстве рисования.

Элементы в представлениях элементов (модель/представление) рисуются при помощи делегатов в qt. Но заголовки представлений элементов всё ещё рисуются при помощи стилей. Делегат по умолчанию рисует элементы частично через текущий стиль; он рисует флажки индикаторы и рассчитывает ограничивающие прямоугольники для элементов, из которых состоит элемент. Если вы желаете добавить поддержку для других типов данных, нежели те, которые поддерживаются при помощи [QStyledItemDelegate](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleditemdelegate.html), вам следует реализовать частный делегат. Заметьте, что делегат должен быть установлен программно для каждого отдельного виджета.

Элемент стиля является графической частью GUI. Виджет состоит из иерархии элементов стилей. *В данной части есть хороший пример иллюстрации данной концепции.* Есть три типа элементов:

* примитивные,
* элементы управления и
* сложные элементы управления.

Элементы определяются соответственно перечислениями [ComplexControl](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html#ComplexControl-enum), [ControlElement](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html#ControlElement-enum), and [PrimitiveElement](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html#PrimitiveElement-enum). Каждый элемент имеет некоторый префикс: CC\_ for complex elements, CE\_ for control elements, and PE\_ for primitive elements. Описание данных элементов содержится в классе [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html). Примитивные элементы не могут существовать по отдельности, но они всегда являются частью более сложных конструкций. Элементы управления выполняют некоторое действие или отображают некоторую информацию для пользователя. В отличие от примитивных элементов они не пассивны, но выполняют некоторую функцию во взаимодействии с пользователем. Сложные элементы управления содержат некоторые под элементы управления. Они ведут себя по-разному в зависимости от того, где пользователь взаимодействует с ними. В дополнение к рисованию, стилю необходимо обеспечивать виджеты информацией о том, какой встроенный контроль был вызван нажатием мыши. Например, [QScrollBar](http://qt-project.org/doc/qt-5.1/qtwidgets/qscrollbar.html) необходимо знать, что нажал ли пользователь бегунок, бороздку ползунка или одну из кнопок.

Заметьте, что под элементы управления не являются теми самыми элементами, которые описывались в предыдущей секции. Вы не можете использовать стиль, чтобы рисовать вспомогательные элементы управления; стиль только рассчитывает ограничивающий прямоугольник, в котором должен быть нарисован под контроль.

В общем, сложные элементы используют элементы управления и примитивные элементы для рисования их под элементов управления, что является подходом, который часто используется во встроенных стилях qt и также в Java стиле. Элементы стилей и виджеты используют стиль для расчёта ограничивающих прямоугольников под элементов управления. Также для измерений во время рисования используется метрика пикселов. Доступные прямоугольники и метрики пикселов представляются тремя перечислениями в [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html): [SubElement](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html#SubElement-enum), [SubControl](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html#SubControl-enum), and [PixelMetric](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html#PixelMetric-enum).

Стиль также содержит набор подсказок стиля, которые представляются как значения в перечислении [StyleHint](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html#StyleHint-enum).

Стиль обычно имеет набор стандартных изображений (таких, как предупреждения, вопросы и изображения ошибок) для окон сообщения, файловых диалогов и т.д. для этого предназначено перечисление [StandardPixmap](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html#StandardPixmap-enum). Его значения представляют стандартные изображения. Qt виджеты используют их, так что когда вы реализуете частные стили, вам следует обеспечить изображения, используемые при помощи стиля, который применяется.

Также стиль рассчитывает пространство между виджетами и макетами. *Далее перечислены функции для контроля за макетами.* Подклассы класса [QStyleOption](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleoption.html) содержит всю информацию, необходимую для стиля отдельных элементов. Настройки стиля инстанцируются – обычно в стеке – и заполняются вызывающимися функциями [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html). В зависимости от того, что рисуется, стиль ожидает разное в разных классах настроек. Например, QStyle::PE\_FrameFocusRect элемент ожидает [QStyleOptionFocusRect](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleoptionfocusrect.html) аргумент, и возможно создать частный подкласс, который может использовать частный подкласс. Также данный класс содержит некоторые публичные переменные по причине производительности.

Виджеты могут быть в определённом количестве состояний, что определяется при помощи перечисления [State](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html#StateFlag-enum). Некоторые из флагов являются общими для всех виджетов, а некоторые имеют отдельные значения для некоторых виджетов. В частности, настройки стиля содержат палитру и ограничивающие прямоугольники виджетов, которые следует рисовать. Большинство виджетов имеют специализированные настройки стиля, которые содержат текст, иконку и размер иконки.

При переопределении функций класса [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html), которые имеют в качестве параметра [QStyleOption](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleoption.html), вам часто необходимо осуществлять приведение типа от [QStyleOption](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleoption.html) к его подклассу. По соображениям безопасности вам следует использовать функцию [qstyleoption\_cast](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleoption.html#qstyleoption_cast)(), чтобы гарантировать, что тип указателя является корректным. *Есть пример кода. Есть пример кода того, как использовать* [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html), чтобы рисовать прямоугольник из функции частного виджета paintEvent(). *Есть пример кода создания некоторого частного стиля.*

Класс стилей определяет три класса для рисования примитивов, элементов управления и сложных элементов управления: [drawPrimitive()](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html#drawPrimitive), [drawControl()](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html#drawControl), and [drawComplexControl()](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html#drawComplexControl). Данные функции принимают следующие параметры:

* значение перечисления элемента для рисования
* QStyleOption, который содержит информацию, необходимую для рисования элемента.
* QPainter, при помощи которого происходит рисование элемента.
* Указатель на QWidget, типично виджет, на котором рисуется элемент.

Не все виджеты передают данным функциям указатели на самих себя. Если настройки стиля не содержат информации, которая вам необходима, то вам следует проверить реализацию виджета, чтобы увидеть высылает ли он указатель на самих себя.

Также в классе стилей есть функции для помещения текста в некотором прямоугольнике, для помещения растра в некотором прямоугольнике и т.д. также есть ряд функций, которые используются функциями для рисования для некоторых расчётов. Они перечислены в данной части. [polish()](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html#polish) and [unpolish()](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html#unpolish) функции также очень полезны. Все виджеты посылаются функции polish() перед тем, как быть показанными, и к функции unpolish(), когда они прячутся. Вы можете использовать эти функции для установки атрибутов на виджеты или для производства другой работы, которая требуется вашим стилем.

Обычный подход при переопределении виртуальных функций [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html) – это делать работу на элементах, которые отличаются от реализации суперкласса; для всех остальных элементов вы может просто использовать реализацию суперкласса.

Каждый стиль обеспечивает цвет, то есть [QBrush](http://qt-project.org/doc/qt-5.1/qtgui/qbrush.html), - палитру, которую следует использовать для рисования виджетов. Существует только один набор цветов для состояний виджета:

* активен,
* неактивен,
* недействителен.

Каждый набор содержит определённый набор ролей цветов, которые можно найти при помощи запроса перечисления [QPalette::ColorRole](http://qt-project.org/doc/qt-5.1/qtgui/qpalette.html#ColorRole-enum). Стиль может создавать градиенты, и, вообще, именно он решает, как используются роли цветов. [QPalette](http://qt-project.org/doc/qt-5.1/qtgui/qpalette.html) сохраняет цвета для разных состояний виджета и для разных ролей цветов. *Далее кратко сказано, как получить палитру стиля, а также как установить свою собственную палитру стилей.* Рекомендуется не использовать свой код для определения цветов. Ни один qt виджет не делает этого.

Некоторые советы при реализации стилей:

1. При реализации стилей необходимо просмотреть код виджетов и код базового класса, а также его предков. Это из-за того, что виджеты используют стили по-разному.
2. Также не рекомендуется, чтобы стили изменяли предлагаемый размер виджетов при помощи [QStyle::sizeFromContents](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html#sizeFromContents)(), но предоставили реализации [QCommonStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qcommonstyle.html) обрабатывать его вместо этого.

Мы используем стиль, который напоминает вид java по умолчанию (ранее известный как Metal). *Везде в данной части далее будет использоваться именно java стиль. Далее описаны ограничения и отличия для данного стиля.*

Первым шагом в проектировании стиля является выбор базового класса. Проектируя стиль в одном классе, мы выигрываем в производительности. также в коде используются операторы switch, что создаёт большие функции, однако при этом код становится очень удобочитаемым.

*Далее рассказывается о тех ограничениях, которые существуют в qt при использовании java стиля. Я плохо понимаю, что такое java стиль и поэтому я не понимаю в достаточной мере данную часть.*

*Далее показан изумительный пример анализа кода встроенного класса qt. Далее показан пример кода, как рисовать в некотором виджете флажок. В данном примере видно, как флажки стилизуются в java стиле от момента, когда виджет получает запрос на рисование, до момента, стиль заканчивает рисование. В целом я разобрался с данным примером, но с ним следует разобраться ещё более подробно.* В данной секции мы проверим то, как большинство виджетов в qt стилизованы. Для всех виджетов предоставлены следующие пункты:

* Таблица с членами настроек стиля.
* Таблица с флагами состояния.
* Его дерево элементов.
* Изображение виджета, в котором элементы обозначены.

Дерево элементов содержит примитивы, элементы управления и сложные элементы управления. При помощи проделывания обхода дерева сверху вниз вы получаете последовательность, в которой следует нарисовать элементы. В узлах мы пишем прямоугольники под элементов, под элементы управления, и метрику пикселов, которые следует рассматривать во время рисования элемента узла. Наш подход к стилизации концентрирует внимание на рисовании виджетов. Расчёты прямоугольников под элементов, под элементов управления, и метрики пикселов, использованных во время рисования, только перечисляются как содержания в деревьях элементов.

*Далее упоминаются некоторые рекомендации, если у вас возникают при изучении стилей некоторые сомнения.*

Некоторые состояния и переменные являются общими для всех виджетов. Они устанавливаются при помощи функции [QStyleOption::initFrom](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleoption.html#initFrom)(). *Далее в таблице перечислены некоторые общие свойства виджетов.*

*В общем смысл стилизации таков: каждый элемент представляется как совокупность некоторых примитивов и состояний. Примитивы рисуются с учётом настроек стиля. Причём для каждого состояния примитив может рисоваться по-разному.*

#### QCursor

<http://qt-project.org/doc/qt-5.1/qtgui/qcursor.html#details>

QCursor класс обеспечивает курсор мыши с произвольной формой. Этот класс используется для создания курсора мыши, которые связаны с некоторым виджетом, а также для того, чтобы получать и устанавливать позицию курсора мыши. Вы можете создавать свои собственные курсоры, основанные на [QBitmap](http://qt-project.org/doc/qt-5.1/qtgui/qbitmap.html), масках и горячих точках.*Далее в данной части описаны некоторые аспекты использования данного класса.*

#### QPalette

<http://qt-project.org/doc/qt-5.1/qtgui/qpalette.html#details>

QPalette класс содержит группы цветов для каждого состояния виджета. Палитра состоит из трёх групп цветов: активный, недействительный и неактивный. Все виджеты в qt содержат палитру и используют её для собственного рисования. Это позволяет делать интерфейс пользователя более настраиваемым. При создании виджета рекомендуется использовать цвета именно в палитре, а не кодировать их напрямую. Рекомендуется использовать палитру по умолчанию (returned by [QGuiApplication::palette](http://qt-project.org/doc/qt-5.1/qtgui/qguiapplication.html#palette)()) и изменять её по необходимости. Именно так и делают виджеты qt. Некоторые стили не используют палитру. Указаны системы Windows XP, Windows Vista, Mac OS X.

#### QCommonStyle

<http://qt-project.org/doc/qt-5.1/qtwidgets/qcommonstyle.html#details>

QCommonStyle класс инкапсулирует общий вид и ощущение графического интерфейса. Этот абстрактный класс реализует некоторый вид виджета, который является общим для всех стилей, обеспечиваемых и являющихся частью qt.

#### QStyleFactory

<http://qt-project.org/doc/qt-5.1/qtwidgets/qstylefactory.html#details>

QStyleFactory класс создаёт объекты [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html), которые являются абстрактными базовыми классами, которые инкапсулируют вид и ощущение GUI. Стили являются либо встроенными либо динамически загружаемыми из плагина стилей ([QStylePlugin](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleplugin.html)).

#### QStyleOption

<http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleoption.html#details>

QStyleOption класс сохраняет параметры, используемые функциями [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html). Данный класс и его подклассы содержат всю информацию, которую функции [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html) получают, чтобы рисовать графический элемент. Обычно данный объект создаётся в стеке при вызове функции класса QStyle. *Есть хороший пример кода, который показывает, как рисуется кнопка.* При переопределении функции [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html), которая принимает параметр QStyleOption, вам часто нужно выполнять приведение типов от QStyleOption к некоторому подклассу. По соображениям безопасности для этого лучше использовать функцию [qstyleoption\_cast](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleoption.html#qstyleoption_cast)(), чтобы гарантировать, что тип указателя является корректным. *Есть хороший демонстрационный пример кода. Также есть ссылка на пример, где демонстрируется использование данного класса.*

#### QStyleHintReturn

<http://qt-project.org/doc/qt-5.1/qtwidgets/qstylehintreturn.html#details>

QStyleHintReturn класс обеспечивает подсказки стилей, которые возвращают более, чем базовые типы. Данный класс и его подклассы используются, чтобы передать информацию от стиля назад к запрашиваемому виджету. *Затем сказано, когда это наиболее полезно, но я этого не понял.*

#### QStyleHintReturnMask

<http://qt-project.org/doc/qt-5.1/qtwidgets/qstylehintreturnmask.html#details>

класс QStyleHintReturnMask обеспечивает подсказки стилей, которые возвращают [QRegion](http://qt-project.org/doc/qt-5.1/qtgui/qregion.html).

#### QStyleHintReturnVariant

<http://qt-project.org/doc/qt-5.1/qtwidgets/qstylehintreturnvariant.html#details>

QStyleHintReturnVariant класс обеспечивает подсказки стиля, которые возвращают QVariant.

#### QStylePainter

<http://qt-project.org/doc/qt-5.1/qtwidgets/qstylepainter.html#details>

класс QStylePainter является удобным классом для рисования элементов [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html) внутри виджета. Он расширяет класс [QPainter](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html) при помощи набора высокоуровневых функций рисования. Данный класс инициализируется вместе с виджетом, сводя на нет необходимость определять классы [QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html), the [QPainter](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html), and the [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html) при каждом вызове виджета. *Есть хороший демонстрационный пример использования данного класса. Пока со стилями всё, но в будущем следует рассмотреть данную проблему более подробно.*

#### QProxyStyle

[*http://qt-project.org/doc/qt-5.1/qtwidgets/qproxystyle.html#details*](http://qt-project.org/doc/qt-5.1/qtwidgets/qproxystyle.html#details)

QProxyStyle класс является удобным классом, который упрощает динамическое переписывание элементов [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html). Он является обёрткой класса [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html), для достижения динамического переписывания рисования или другого специфического поведения стиля. *Далее указывается непонятное для меня предупреждение.*

#### ДРУГИЕ КЛАССЫ СТИЛЕЙ

[QStyleOptionButton](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleoptionbutton.html), [QStyleOptionComboBox](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleoptioncombobox.html), [QStyleOptionComplex](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleoptioncomplex.html), [QStyleOptionDockWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleoptiondockwidget.html), [QStyleOptionFocusRect](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleoptionfocusrect.html), [QStyleOptionFrame](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleoptionframe.html), [QStyleOptionGraphicsItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleoptiongraphicsitem.html) и т. д. – это классы, которые описывают настройки стиля для разных элементов управления.

## Некоторые другие классы виджетов

#### QFontDatabase

<http://qt-project.org/doc/qt-5.1/qtgui/qfontdatabase.html#details>

класс QFontDatabase обеспечивает информацию о шрифтах, доступных в нижележащей системе. Этот класс используют для запроса о доступных семействах шрифтов. *Есть интересный пример кода, который можно было бы подробнее изучить и реализовать.*

#### QGraphicsAnchorLayout

<http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsanchorlayout.html#details>

QGraphicsAnchorLayout класс обеспечивает макет, где макет может якорить несколько виджетов вместе в Graphics View. Данный класс позволяет разработчикам определить то, как виджеты должны лежать друг относительно друга в макете. *Далее приводится некоторое описание кода, а также некоторые примеры использования данного класса.*

#### QGraphicsAnchor

<http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsanchor.html#details>

QGraphicsAnchor класс представляет якорь между двумя элементами в классе [QGraphicsAnchorLayout](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsanchorlayout.html). Данный класс предоставляет программный интерфейс, который предоставляет возможность вам запрашивать и манипулировать свойствами, которые имеет якорь. *Далее сказано про использование в программе данного класса.*

## ШИТЫ СТИЛЕЙ

<http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet.html>

Qt Style Sheets является мощным механизмом qt, который позволяет вам настраивать внешность виджетов в дополнение к тому, что уже доступно при помощи класса [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html). Понятия, терминология и синтаксис данного механизма во многом появились под вилянием HTML [Cascading Style Sheets (CSS)](http://www.w3.org/Style/CSS/), но адаптировано к миру виджетов. *Есть ссылка на CSS.* Шиты стилей – это текстовые спецификации, которые могут быть установлены во всём приложении с использованием [QApplication::setStyleSheet](http://qt-project.org/doc/qt-5.1/qtwidgets/qapplication.html#styleSheet-prop)() или на определённом виджете с использованием [QWidget::setStyleSheet](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#styleSheet-prop)().

Шиты намного более мощные, нежели палитры. Шиты применяются сверху от текущего стиля виджета. Это означает, что ваши приложения будут выглядеть как можно ближе к нативным, однако принимать во внимание также и информацию из шита. Когда шит активен, то возвращаемый стиль является обёрткой шит стиля, но специфического для данной платформы стиля. Пока шиты не поддерживаются для частных подклассов QStyle, что планируется исправить в ближайшее будущее.

### СИНТАКСИС ШИТОВ СТИЛЕЙ

<http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-syntax.html>

терминология и правила шита почти идентичны таковым для HTML CSS. Шиты стилей состоят из набора правил стиля. Правило стиля изготовлено из селектора и объявления. Селектор определяет, какие виджеты находятся под влиянием данного правила. Объявление определяет, какие свойства следует установит на виджет. *Есть пример.* Шит не является чувствительным к регистру. Селекторов может быть несколько для одного объявления, и они отделяются при помощи запятых. Объявления могут отделяться при помощи «;» и располагаться при этом в одну линию.

Шиты поддерживают все типы селекторов, определённых при помощи CSS2. *Далее приводится таблица, которая содержит наиболее полезные типы селекторов. Невероятно полезная таблица.* Для стилизации сложных виджетов необходимо иметь доступ к его под элементам. *Далее показан пример кода, как обращаться к под элементам управления.* «::» оператор в данном случае отличается по смыслу от того, как он используется в CSS. [subcontrol-origin](http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-reference.html#subcontrol-origin-prop) свойство позволяет изменить прямоугольник. [subcontrol-position](http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-reference.html#subcontrol-position-prop) изменяет выравнивание. *Есть крутой пример того, как изменять вид стрелки при нажатии на неё.* Для сложных виджетов следует учитывать, что если одно свойство подконтроля настраивается, то могут также настраиваться автоматически и другие свойства. Селекторы могут содержать псевдо-состояния, которые обозначают, что приложение ограничивается правилом, основанным на состоянии виджета. Они ограничиваются «:». Также псевдо-состояния можно отрицать. *Есть хорошие примеры кода.* Также можно создавать цепочки псевдо-состояний. Псевдо-состояния могут появляться в комбинации с под элементам управления.

Конфликты возникают, когда несколько правил стиля применяются к одним и тем же свойствам, но с разными значениями. *Есть пример кода.* Чем более специфицированным является селектор, тем более приоритетным является правило. Более специфичными являются селекторы с псевдо-состояниями. *Есть замечательные примеры кода. Далее размещён порядок спецификации, который взят из CSS. Также есть пример демонстрации расчёта специфичности. Он мне довольно понятен.* Спецификация селектора рассчитывается следующим образом:

* Считается число атрибутов идентификатора в селекторе (= а)
* Считается число других атрибутов и псевдо-состояний в селекторе (= b)
* Считается число имён элементов в селекторе (= с)
* Игнорируются псевдо-элементы

Шиты стилей могут быть установлены в [QApplication](http://qt-project.org/doc/qt-5.1/qtwidgets/qapplication.html), в родительских виджетах и в дочерних виджетах. Шит конкретного виджета получается при помощи слияния шитов его родительских виджетов. В приоритете находятся шиты виджета, а в дальнейшем – его предков, начиная от прямых родителей. Каскад шитов стилей является сложной темой. Здесь следует учитывать спецификацию правил стиля, а также положения виджета, для которого это определяется в иерархии наследования. За деталями следует обращаться к документации CSS2, на *которую тут же дана некоторая ссылка.* В классическом CSS, если цвет и шрифт элемента не установлены явно, то они наследуются от его родителя. В шитах этого не происходит. *Есть примеры кода и объяснение того, как это делается* При определении стиля для виджета, который находится в пространстве имён, могут возникать некоторые проблемы. Там следует использовать символ «--» вместо «::» Для их решения следует сделать то*, что показано в примере кода в данной части. В конце показано, как в стиле устанавливать значения свойств, которые определены при помощи макроса Q\_PROPERTY. Для этого используется qproperty.*

### ИНТЕГРАЦИЯ ШИТОВ СТИЛЕЙ С QT ДИЗАЙНЕРОМ

<http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-designer.html>

Qt Designer является великолепным инструментом для шитов. Вы можете нажать правой кнопкой мыши на любой виджет в дизайнере и выбрать **Change** [**styleSheet**](http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet.html#stylesheet)**...**, чтобы установить шит. *Далее описаны и показаны диалоги для установки шитов стилей.*

### НАСТРОЙКА QT ВИДЖЕТОВ С ИСПОЛЬЗОВАНИЕМ ШИТОВ СТИЛЕЙ

[*http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-customizing.html*](http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-customizing.html)

при использовании шитов стилей, каждый виджет обрабатывается как ящик с четырьмя концентрическими прямоугольниками:

* прямоугольник полей,
* прямоугольник границ,
* прямоугольник содержания,
* прямоугольник заполнения.

*Далее показана схема их размещения друг относительно друга.* В шитах стиля есть свойства, которые характеризуют данные прямоугольники. По умолчанию многие из них установлены в нуль. Свойство [background-image](http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-reference.html#background-image-prop) позволяет установить фон для виджета. *Далее приведены некоторые свойства для управления цветом фона.* [border-image](http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-reference.html#border-image-prop) свойство задаёт фон, который масштабируется в соответствии с размерами виджета. [image](http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-reference.html#image-prop) свойство позволяет рисовать изображение над [border-image](http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-reference.html#border-image-prop). [image-position](http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-reference.html#image-position-prop) задаёт местоположения изображения в виджете. Если в данном свойстве задать изображение SVG, то оно будет масштабироваться в соответствии с размерами виджета. Таким образом, правила рисования следующие:

* Устанавливается область, где выполняется операция рисования (border-radius)
* Рисуется фон (background-image)
* Рисуется граница (border-image, border)
* Рисуется изображение наложения (overlay image)

Виджет рассматривается как иерархия под элементов управления, нарисованных сверху друг друга. *Есть пример правил рисования.* Под элементы управления размещаются внутри своих родителей при помощи использования свойств [subcontrol-position](http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-reference.html#subcontrol-position-prop) and [subcontrol-origin](http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-reference.html#subcontrol-origin-prop). Разместившись, под элементы управления могут быть стилизованы при помощи [box model](http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-customizing.html#box-model). В случае сложных виджетов, если одно из свойств под контроля было настроено, то все другие свойства под элементов управления также должны быть настроены.

### СПРАВОЧНИК ПО ШИТАМ СТИЛЕЙ

<http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-reference.html>

*в данной части приводится ссылка на все свойства данного механизма. Рассмотрим их. Вначале для каждого виджета описано то, что можно стилизовать. Затем перечисляются все свойства, которые поддерживаются* Qt Style Sheets. **alternate-background-color – используется в подклассах класса** [QAbstractItemView](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractitemview.html), - *непонятно.* **Background – экливалентно определению многих свойств фона.** background-color, background-image. background-repeat – будет ли и как изображение фона повторяться для заполнения прямоугольника. background-position – выравнивание изображения фона. **background-attachment – определяет, прокручивается ли изображение фона по отношению к вьюпорту. По умолчанию, фон прокручивается вместе с вьюпортом. background-clip – прямоугольник виджета, в котором рисуется фон: куда вставляются цвет фона и изображение фона. background-origin – прямоугольник фона. border – краткое определение многих свойств границы.** border-top, border-right, border-bottom, border-left – то же самое, но для конкретных границ. **border-color – цвет граней границы.** border-top-color, border-right-color, border-bottom-color, border-left-color. **border-image – изображение, которое используется для заполнения границы. border-radius – радиус углов границы.** border-top-left-radius, border-top-right-radius, border-bottom-right-radius, border-bottom-left-radius. **border-style – стиль всех граней границы.** border-top-style, border-right-style, border-bottom-style, border-left-style. **border-width,** border-top-width, border-right-width, border-bottom-width, border-left-width. **Bottom – для выставления под элементов управления внутри родительского виджета. button-layout – макеты кнопок для диалогов. сolor – цвет для рисования текста. dialogbuttonbox-buttons-have-icons – показывают ли кнопки некоторого диалога иконки. font – краткая запись установки шрифта текста.** font-family, font-size, font-style, font-weight. **gridline-color** - цвет стеки в таблице. **Height – высота под контроля. icon-size – размеры иконки внутри виджета. image** – изображение, которое рисуется в прямоугольнике содержания под контроля. **image-position – выравнивание положения изображения. left - для установки под элементов управления. lineedit-password-character – *не понимаю.* Margin – поля виджета.** margin-top, margin-right, margin-bottom, margin-left. **max-height – максимальная высота виджета или под контроля. max-width messagebox-text-interaction-flags – *не понимаю.* min-height, min-width. opacity – непрозрачность виджета. Padding – заполнение виджета.** padding-top, padding-right, padding-bottom, padding-left. **paint-alternating-row-colors-for-empty-area – не понимаю. Position – являются ли смещения, определённые при помощи** [left](http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-reference.html#left-prop), [right](http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-reference.html#right-prop), [top](http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-reference.html#top-prop), and [bottom](http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-reference.html#bottom-prop), относительными или абсолютными. **Right. selection-background-color – фоновый цвет выбранного текста или элемента. selection-color – передний цвет выделенного текста или элементов. show-decoration-selected – *не понимаю.* Spacing – внутреннее расстояние виджета. subcontrol-origin – начальный прямоугольник под контроля внутри родительского элемента. subcontrol-position – выравнивание под контроля внутри прямоугольника, определённого при помощи свойства** [subcontrol-origin](http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-reference.html#subcontrol-origin-prop). **text-align – выравнивание текста и иконки внутри содержания виджета. text-decoration – дополнительные эффекты текста. top, width.** *далее перечислены свойства, которые могут быть использованы для настройки иконок в qt. Также над таблицей есть некоторое замечание по поводу иконок* [QDialogButtonBox](http://qt-project.org/doc/qt-5.1/qtwidgets/qdialogbuttonbox.html). *Далее приводится таблица, которая обобщает синтаксис и значение различных типов свойств.* **Alignment, Attachment (**прокрутка или без**), Background, Boolean, Border, Border Image (***есть хорошее описание того, что называется изображением границы***), Border Style, Box Colors (**цвета границ ящика**), Box Lengths, Brush (**градиент или цвет для использования в палитре**), Brush, Font, Font Size, Font Style, Font Weight, Gradient, Icon, Length, Number, Origin (**определяет, какой из четырёх прямоугольников следует использовать**),** [**PaletteRole**](http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-reference.html#paletterole) **(**роли цвета в палитре виджета**), Radius (**смотри описание, определяет закругление углов**), Repeat (**значение определяет природу повторений**), Url.** *Далее размещён список псевдо-состояний. Затем приводится список подэлементов управления.*

### ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ QT ШИТОВ СТИЛЕЙ

<http://qt-project.org/doc/qt-5.1/qtwidgets/stylesheet-examples.html>

*на данной странице показаны некоторые примеры использования данного механизма, которые я позже изучу*. *Переходим к изучению макетов.*

## УПРАВЛЕНИЕ МАКЕТАМИ

<http://qt-project.org/doc/qt-5.1/qtwidgets/layout.html>

система макетов qt обеспечивает простой и мощный способ автоматической организации дочерних виджетов внутри виджета, чтобы гарантировать, что они будут качественно использоваться в некотором пространстве. Макеты автоматически изменяют размер и положение виджетов, когда количество пространства, доступного для них, изменяется. Когда макет установлен в виджете, он следит за выполнением следующих задач:

* Размещение виджетов
* Разумные размеры окон по умолчанию
* Разумные минимальные размеры окон
* Обработка изменения размера
* Автоматически обновляет при изменении содержания:
* Размер шрифта, текста или других содержаний дочерних виджетов.
* Сокрытие или показ дочерних виджетов
* Удаление дочерних виджетов.

*Далее перечислены классы макетов, которые следует в дальнейшем изучить очень поверхностно, так для их использования и установки лучше использовать Qt Designer. Затем проиллюстрированы некоторые типы макетов. Затем показан код помещения виджетов в макеты.* Макет автоматически изменяет родителя виджета, поэтому при построении виджетов не следует явно устанавливать родителя. Виджеты в макете являются дочерними для того виджета, в котором установлен макет. Виджеты вообще не могут иметь родителей макетов, а только классы, унаследованные от QWidget.

*Далее представлен процесс, который происходит при добавлении виджета в макет.*

1. Вначале всем виджетам выделяется место пространства в соответствии с их [QWidget::sizePolicy](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#sizePolicy-prop)() and [QWidget::sizeHint](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#sizeHint-prop)().
2. Если некоторые виджеты имеют установленные факторы растяжения со значением более нуля, то тогда они динамически размещаются в пространстве пропорционально их фактору растяжения.
3. Если некоторые виджеты имеют фактор растяжения, установленный в нуль, то они только будут получать больше пространства, если никаким другим виджетам не нужно пространство. Исходя из этого, пространство вначале выделяется виджетам с политикой изменения размера [Expanding](http://qt-project.org/doc/qt-5.1/qtwidgets/qsizepolicy.html#Policy-enum).
4. Всем виджетам, которые занимают меньшее пространство, чем их минимальные размеры (или minimum size hint, если минимальных размеров не установлено), выделяется то минимальное пространство, которое они требуют. (виджеты, которые не имеют минимальных размеров или подсказки относительно минимального размера, обрабатываются так, что определяющим фактором является как раз фактор растяжения).
5. Всем виджетам, которым выделено больше пространства, нежели их максимальные размеры, выделяется пространство, соответствующие их максимальным размерам.

При помещении в макет, пространство между виджетами делится в соответствии с их [QWidget::sizePolicy](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#sizePolicy-prop)() или их minimum size hint в зависимости от того, что больше. Факторы растяжения устанавливаются, чтобы показать, насколько больше пространства следует выделить виджетам по сравнению с другими виджетами. При создании собственного виджета, следует также налаживать его свойства для макета. Если виджет содержит некоторый макет, то это сразу принимается во внимание. Иначе можно поступить в данном случае тремя способами:

* Переопределить [QWidget::sizeHint](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#sizeHint-prop)(), чтобы вернуть желаемый размер виджета.
* Переопределить [QWidget::minimumSizeHint](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#minimumSizeHint-prop)(), чтобы вернуть наименьший размер, который может иметь виджет.
* Вызвать [QWidget::setSizePolicy](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#sizePolicy-prop)(), чтобы задать пространство, требуемое для виджета.

Каждый раз при изменении size hint, minimum size hint или size policy вызывайте функцию [QWidget::updateGeometry](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#updateGeometry)(). Но не вызывайте её много раз подряд, так как это вызовет множественный пересчёт макета. Если предпочтительная высота виджета зависит от его ширины, то установите флаг [height-for-width](http://qt-project.org/doc/qt-5.1/qtwidgets/qsizepolicy.html#hasHeightForWidth) в [size policy](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#sizePolicy-prop) виджета и переопределите [QWidget::heightForWidth](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#heightForWidth)(). При этом всё равно хорошо было бы определить разумный sizeHint().

Использование форматированного текста в метке может ввести некоторые сложности в макет его родительского виджета. Проблемы возникают из-за способа, которым форматированный текст обрабатывается менеджерами макета qt, когда метка является обёрткой слова.

В некоторых случаях родительских макет выставляется в моду QLayout::FreeResize, означающую, что он не будет адаптировать макет его содержимого, чтобы подогнать внутри малых окон, или даже предостерегать пользователя от делания окна очень малым, чтобы использовать его. Это можно преодолеть при помощи специализации проблемных виджетов, и определения подходящих функций [sizeHint()](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#sizeHint-prop) and [minimumSizeHint()](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#minimumSizeHint-prop).

Если вы создаёте единственный в своём роде макет, вы можете также сделать частный виджет, как описано выше. Переопределите [QWidget::resizeEvent](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#resizeEvent)(), чтобы рассчитать требуемое распространение размеров и вызовите [setGeometry()](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#geometry-prop) для всех дочерних виджетов. *Рассмотрены некоторые другие аспекты.*

Можно также создать подкласс класса [QLayout](http://qt-project.org/doc/qt-5.1/qtwidgets/qlayout.html). *Есть ссылки на некоторые примеры. В данной части также есть некоторый пример кода. Очень полезный пример кода, особенно, замечания после него.*

#### QGroupBox

[*http://qt-project.org/doc/qt-5.1/qtwidgets/qgroupbox.html#details*](http://qt-project.org/doc/qt-5.1/qtwidgets/qgroupbox.html#details)

QGroupBox виджет обеспечивает группу с названием. Он отображает другие виджеты внутри себя.

#### QButtonGroup

<http://qt-project.org/doc/qt-5.1/qtwidgets/qbuttongroup.html>

QButtonGroup класс обеспечивает контейнер для организации групп виджетов кнопок. Он не визуализирует входящие в него кнопки, но только устанавливает состояние каждой кнопки, которая входит в него. *Возвращаемся к виджетам. Рассмотрим модель/представление в qt.*

## ПРОГРАММИРОВАНИЕ МОДЕЛИ/ПРЕДСТАВЛЕНИЯ

<http://qt-project.org/doc/qt-5.1/qtwidgets/model-view-programming.html>

qt содержит набор классов представлений элементов, которые используют архитектуру модель/представление, чтобы управлять отношениями между данными и способом, которым они представляются пользователю. Если представление и контроллер объединить, то получится архитектура модель/представление. Для обеспечения гибкого управления ввода пользователя мы вводим понятие делегата. Преимуществом делегата в данном случае является то, что он позволет настроить способ рисования и редактирования элементов. Делегаты, модели и представления общаются друг с другом при помощи механизма сигналов и слотов.

* Сигналы от модели информируют представление об изменениях данных, удерживаемых в источнике данных.
* Сигналы от представления обеспечивают информацию о взаимодействии пользователя с элементами, которые отображены.
* Сигналы от делегата используются во время редактирования, чтобы сообщить модели и представлению о состоянии редактора.

Все элементы моделей основаны на классе [QAbstractItemModel](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html). Он определяет интерфейс, который используется делегатами и представлением для доступа к элементам модели данных. Модель только предоставляет доступ к данным, но не хранит их. [QStyledItemDelegate](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleditemdelegate.html) используется как делегат по умолчанию в стандартных представлениях qt.

Есть два типа сортировки в архитектуре модель/представление. Если ваша модель сортируемая, т.е. если она реализует функцию [QAbstractItemModel::sort](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html#sort)(), и [QTableView](http://qt-project.org/doc/qt-5.1/qtwidgets/qtableview.html), и [QTreeView](http://qt-project.org/doc/qt-5.1/qtwidgets/qtreeview.html) обеспечивают программный интерфейс, который позволяет вам сортировать вашу модель данных программно. В дополнение вы можете разрешить сортировку интерактивную (пользователь сортирует данные), при помощи соединения [QHeaderView::sortIndicatorChanged](http://qt-project.org/doc/qt-5.1/qtwidgets/qheaderview.html#sortIndicatorChanged)() сигнала с слотом [QTableView::sortByColumn](http://qt-project.org/doc/qt-5.1/qtwidgets/qtableview.html#sortByColumn)().

[QListWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qlistwidget.html), [QTreeWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qtreewidget.html), and [QTableWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qtablewidget.html). – удобные классы для представления моделей, но не предполагается, что их будут специализировать. Представления строятся также как и все другие виджеты.

[QStandardItemModel](http://qt-project.org/doc/qt-5.1/qtgui/qstandarditemmodel.html) является многоцелевой моделью, которая может быть использована для представления различных структур данных, необходимых спискам, таблицам и деревьям. [QFileSystemModel](http://qt-project.org/doc/qt-5.1/qtwidgets/qfilesystemmodel.html) является моделью, которая удерживает информацию о содержании данной директории.

Вне зависимости от того, как хранятся данные, в классе [QAbstractItemModel](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html) они репрезентируются как иерархические структуры, содержащие таблицы элементов.

Чтобы гарантировать, что представление данных держится отдельно от способа доступа к ним, вводится понятие индекса модели. Делегаты и представления используют данные индексы для доступа к данным модели.

В результате только модели необходимо знать, как получить данные, и тип данных, управляемый моделью, может быть выбран достаточно общим. Индексы модели представляют из себя временные ссылки на некоторые части информации и могут быть использованы для получения или изменения данных через модели. Индексы могут стать недействительными и поэтому их не следует сохранять. Индексы характеризуются номерами строк, столбцов и родительским индексом. *Есть хорошая визуализация того, как устроены индексы. Э*лементы в модели могут выполнять различные роли для других компонентов, позволяя обеспечивать различные типы данных для различных ситуаций. Мы можем запросить у модели элемент данных при помощи передачи ей индекса модели, соответствующего элементу, а также при помощи задания роли, чтобы получить тип данных, который мы желаем. Наиболее общие случаи использования для данных покрываются стандартными ролями, описанными в [Qt::ItemDataRole](http://qt-project.org/doc/qt-5.1/qtcore/qt.html#ItemDataRole-enum). *Как я понимаю, в модели данных для одного и того же индекса можно выставлять различные роли, чтобы отображать для каждой роли некоторое значение. То есть элемент данных содержит сами данные, цвет фона и т.д.* Также возможно создать дополнительные роли для специфических для приложения целей. *Далее приводится пример получения данных из модели данных.* Для доступа к элементам высокого уровня в модели, определите нулевой индекс модели как родительский индекс при помощи QModelIndex(). *Далее приводится пример использования класса представления элементов.*

Что касается выделений, то если для нескольких представлений существует общая модель данных, то также лучше создавать и общую модель выделения. Но можно создавать и различные модели выделения, хотя модель данных для этих представлений будет одинаковой. [QItemSelectionModel](http://qt-project.org/doc/qt-5.1/qtcore/qitemselectionmodel.html) обеспечивает механизм для обработки выделений внутри представлений. Большинство стандартных представлений используют свои собственные модели выделения и нормально работать именно с ними. В общем, несмотря на то, что вы специализируете модель или представление, вам нет необходимости манипулировать содержанием выделений напрямую. Но при желании это можно сделать. *Далее показывается, как разделять модели выделения между различными представлениями.*

[QAbstractItemDelegate](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractitemdelegate.html) является стандартным классом для управления делегатами. Простые, основанные на виджетах делегаты могут специализировать класс [QItemDelegate](http://qt-project.org/doc/qt-5.1/qtwidgets/qitemdelegate.html) для написания делегата, так как в нём содержатся реализации некоторых полезных функций. Редакторы для делегатов могут быть реализованы или при помощи виджетов или при помощи управления процессом редактирования при помощи обработки событий напрямую. *Далее показан пример кода создания простого делегата.* [createEditor()](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractitemdelegate.html#createEditor) обеспечивает всё, что нужно делегату для установки подходящего виджета в качестве своего редактора. В общем можно возвращать различные делегаты в зависимости от того, в каком месте представления мы находимся. Делегат должен содержать функцию, которая копирует содержимое модели в редактор setEditorData(). В этой функции происходит приведение типа, так как в представлении может быть несколько делегатов. По окончании редактирования пользователь желает обновить данные в модели. Для этого он использует функцию [setModelData()](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractitemdelegate.html#setModelData). В классе [QItemDelegate](http://qt-project.org/doc/qt-5.1/qtwidgets/qitemdelegate.html) сигнал [closeEditor()](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractitemdelegate.html#closeEditor) информирует об окончании редактирования. Также в обязанности делегата входит управление геометрией редактора. Геометрию следует устанавливать при изменении положения элемента в представлении, а также при создании редактора. К счастью, представления обеспечивают всю необходимую информацию о геометрии внутри опции представления в функции updateEditorGeometry(). После редактирования делегатам необходимо предоставлять некоторые подсказки другим компонентам о результатах процесса редактирования, и обеспечивать подсказки, которые будут помогать дальнейшим операциям редактирования. это достигается при помощи испускания сигнала [closeEditor()](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractitemdelegate.html#closeEditor) с соответствующей подсказкой. Можно, например, начинать редактирование следующего элемента в представлении.

[QItemSelectionModel](http://qt-project.org/doc/qt-5.1/qtcore/qitemselectionmodel.html) класс предназначен для того, чтобы сортировать информацию об элементах, которые выделены в представлении. Данная модель не зависит ни от каких представлений. Выделения содержатся в диапазонах выделения, которые содержат только информацию о начальном и конечном индексах выделения. Выделения реализованы как наборы индексов моделей, к которым применяются некоторые операции. Наиболее недавнее выделение известно как текущее выделение. В представлении есть два независимых состояния: текущий элемент и элементы выделения. *Затем есть пример использования модели выделения. Далее показан пример кода, как получать информацию о состоянии выделения.* Есть совокупность флагов выделения, которые говорят о том, как обновлять совокупность индексов в выделениях. *Есть многочисленные примеры кода. Далее показано, как выделить все элементы в модели.*

При создании новой модели для существующей структуры данных важно рассмотреть, какой тип модели следует использовать для обеспечения интерфейса для данных. *Есть примеры кода создания частной модели данных. Они все написаны без операторов switch. Далее показан пример создания редактируемой модели данных. После этого показан пример вставки и удаления строк. Далее идут примеры использования стандартных виджетов. Есть ссылки на все тип виджетов.*

Иногда бывает удобнее спрятать некоторые элементы в представлении вместо того, чтобы удалить их. Для всех стандартных виджетов данная возможность предусмотрена. *Далее показано, как в данных виджетах считывать выделения.* Все три удобных виджета обеспечивают функции для поиска элементов в представлении. *Есть пример кода. Этот пример кода показывает, что использование данных удобных классов является очень даже продуктивным.*

Инфраструктура перетаскивания и опускания qt полностью поддерживается данным каркасом. Элементы можно перетаскивать внутри представлений, а данные могут импортироваться или экспортироваться как MIME-закодированные данные. Стандартные представления поддерживают внутренний механизм перетаскивания. Все стандартные виджеты могут быть использованы как источники операций перетаскивания и опускания. *Затем указаны флаги, которые следует установить для использования данной возможности qt.* При перетаскивании элементы копируются, но не удаляются. *Затем показано, как осуществлять данные операции между стандартными представлениями элементов. При этом также следует и в модели сказать, что перетаскивание возможно. Также при этом в модели следует установить соответствующие флаги. Также в модели есть функция, которая объявляет MIME-типы, используемые в модели.*

Также модель должна содержать код для кодирования данных в рекламируемый формат. Это достигается при помощи переопределения функции [QAbstractItemModel::mimeData](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html#mimeData)(). *Далее показан пример кода.* Способ, при помощи которого модель обрабатывает опущенные данные, зависит как от её типа, так и от способа, которым предпочтительнее представлять её содержимое пользователю. Таблицы и списки могут создавать новые строки и столбцы. Деревья могут создавать новые дочерние элементы. Опущенные данные обрабатываются при помощи переопределения функции [QAbstractItemModel::dropMimeData](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html#dropMimeData)(). *Далее показан простой пример кода обработки опущенных данных.* Также данная функция должна декодировать данные и вставлять их в нижележащую структуру данных. *Есть хороший пример декодирования.*

В архитектуре модель/представление различные представления могут удерживать одни и те же данные, но представлять их совершенно различным способом. Предположим, представления желают представлять данные, например, по-разному отсортированные. Делать это в представлении чревато потерей производительности. Для решения такого рода задач в архитектуре модель/представление содержатся так называемые прокси модели, чтобы управлять информацией, обеспечиваемой отдельными представлениями и моделями. Они ведут себя как обычные модели с точки зрения представления, а с точки зрения данных, они ведут себя как средства доступа со стороны представления. Прокси модели могут быть вставлены между существующей моделью и любым количеством представлений. [QSortFilterProxyModel](http://qt-project.org/doc/qt-5.1/qtcore/qsortfilterproxymodel.html) – это прокси модель в qt. *Показан пример кода данного класса.* В общем, тип обработки используемых моделей прокси вовлекает отображение каждого элемента данных из начального положения в исходной модели в другое положение в прокси модели. В некоторых прокси моделях некоторые элементы могут не иметь соответствующего положения. Это так называемые фильтрующие прокси модели. Представления получают доступ к элементам с использованием индексов моделей, обеспечиваемых прокси моделью, и эти индексы содержат информацию об исходной модели или о положении исходных элементов в этой модели. [QSortFilterProxyModel](http://qt-project.org/doc/qt-5.1/qtcore/qsortfilterproxymodel.html) предоставляет возможность отфильтровать данные исходной модели перед их передачей представлению. [QSortFilterProxyModel](http://qt-project.org/doc/qt-5.1/qtcore/qsortfilterproxymodel.html) обеспечивает фильтрующую модель, которая довольно универсальна, и которая может быть использована в большом многообразии стандартных ситуаций. Можно специализировать данную модель для предоставления частных фильтров. *Показано, как это сделать. Далее показано, как проводить сортировку, а также специализировать данный класс так, чтобы выполнять некоторое частное сравнение.* Подклассы модели должны обеспечить реализацию многих виртуальных функций, определённых в базовом классе [QAbstractItemModel](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html). Эти функции можно разделить на три группы:

* Обработка элементов данных
* Навигация и создание индексов
* Поддержка перетаскивания и опускания и обработка MIME-типов

*Затем перечисляются функции, которые следует специализировать для тех или иных целей. Очень полезная и важная информация.*

Медленное распространение модели данных позволяет отложить запросы о модели данных, пока она действительно не понадобится представлению. Некоторые модели должны получать информацию из удалённых источников или должны производить времязатратные операции для получения информации об организации данных. Например, в древовидных моделях времязатратно вызывать каждый раз функцию rowCount() для всей модели. Её можно переопределить для получения только количества детей. *Затем даются очень неплохие пояснения данной возможности и некоторые примеры её реализации для модели дерева. Далее также перечисляются все необходимые функции.* Функция [canFetchMore()](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html#canFetchMore) проверяет, имеет ли родитель больше доступных данных и возвращает правду или ложь соответственно. [fetchMore()](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html#fetchMore) функция извлекает данные, основанные на определённом родителе. Обе эти функции могут быть объединены. *Не понимаю до конца их предназначение.* Первую функцию используют для проверки того, нет ли ещё больше данных для извлечения. А вторую используют для их непосредственного извлечения.

### РУКОВОДСТВО ПО МОДЕЛИ/ПРЕДСТАВЛЕНИЮ

<http://qt-project.org/doc/qt-5.1/qtwidgets/modelview.html>

Есть два способа использования виджетов для моделей. Первый подразумевает хранение внутри виджета некоторого своего контейнера для отсортированных данных. В этом случае возникает дублирование данных. Второй способ заключается в предоставлении виджетом только интерфейса для доступа к реальным данным. Второй подход также даёт все преимущества модели/представления. Модель/представление – это технология, призванная отделить данные от представления, которое их использует. Стандартные виджеты не разработаны для данных целей. *В данной части есть хорошая иллюстративная схема.*

*Модель/представление устраняет проблемы совместимости данных, которые могут возникнуть со стандартными виджетами. Также данный подход делает легче использование более одного представления для одних и тех же данных, так как одна модель может быть передана многим представлениям. Одно из важных отличий состоит в том, что виджеты из модели/представления не сохраняют данные внутри табличных ячеек. Действительно, они работают прямо с вашими данными. Так как классы представления не знают структуру ваших данных, вам следует обеспечить обёртку для того, чтобы ваши данные соответствовали интерфейсу* [QAbstractItemModel](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html). Как только виджет получает ссылку на модель, он может отображать её содержание и может быть её редактором. *есть таблица сравнения виджетов и классов модели/представления. Теперь мне стала понятна разница между ними. Виджеты сохраняют локальную копию данных, а представления позволяют успешно работать со внешними данными. Отмечу, что* [QComboBox](http://qt-project.org/doc/qt-5.1/qtwidgets/qcombobox.html) может работать, как обычный виджет и как представление для модели данных.

[QDataWidgetMapper](http://qt-project.org/doc/qt-5.1/qtwidgets/qdatawidgetmapper.html) отображает форму виджетов на строку таблицы и делает очень лёгким построение форм для таблиц баз данных. [QCompleter](http://qt-project.org/doc/qt-5.1/qtwidgets/qcompleter.html) предоставляет авто завершения для некоторых виджетов. *Далее приводится пример построения модели только для чтения. Затем в таблицу добавляются некоторые другие роли для отображения данных. Затем приводится список ролей. Затем показывается пример того, как отображать в ячейке таблицы время с точностью до секунд. В данном случае таблица должна обновлять своё содержимое каждую секунду. В данной части демонстрируется код для решения данной проблемы.* При испускании сигнала [dataChanged()](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html#dataChanged) представление автоматически запрашивает данные у модели. Что касается заголовков, то их можно устанавливать. Также есть функции для того, чтобы прятать. [setData()](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html#setData) будет вызываться каждый раз, когда пользователь будет редактировать ячейку. Различные свойства ячейки, которые она имеет, могут быть приведены в соответствие при помощи функции [flags()](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html#flags). *Далее приводится некоторая информация о представлении дерева, а также говорится, что* [QListView](http://qt-project.org/doc/qt-5.1/qtwidgets/qlistview.html), [QTableView](http://qt-project.org/doc/qt-5.1/qtwidgets/qtableview.html) and [QTreeView](http://qt-project.org/doc/qt-5.1/qtwidgets/qtreeview.html) могут использовать одну и ту же модель для отображения. [QStandardItemModel](http://qt-project.org/doc/qt-5.1/qtgui/qstandarditemmodel.html) лучше использовать для построения деревьев. *Далее приведены стандартные модели данных, которые встроены в qt.* Представление имеет метод [setItemDelegate()](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractitemview.html#setItemDelegate), который заменяет делегат по умолчанию на некоторый частный делегат. *Приводится пример, который позволяет создать делегат, который показывает рейтинг в виде звёзд. Для этого переопределяется лишь две функции класса* [*QStyledItemDelegate*](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleditemdelegate.html)*: paint() и sizeHint().* *Далее есть ссылки на примеры создания новых делегатов.*

Пассивная природа моделей является ещё одним вызовом для программистов. Несовместимости в модели могут вызвать крах приложения. Так как модель подвергается воздействию бесчисленных вызовов из представления, тяжело найти, какой вызов обрушил приложения и какая операция вводит проблемы. Qt Labs предоставляет программное обеспечение, называемое [ModelTest](http://qt-project.org/wiki/Model_Test), которое проверяет модели, пока ваше программирование выполняется. Каждый раз при изменении модели данный сервис сканирует модель и сообщает об ошибках с утверждениями. Это особенно важно для моделей деревьев, так как их иерархическая природа оставляет много возможностей для тонких несоответствий. Несмотря на классы представления, данный сервис использует индексы, выходящие за пределы, чтобы протестировать модель. Это означает, что ваше приложение может обрушится вместе с данным сервисом, даже если оно прекрасно работает без него. Поэтому вам также необходимо обрабатывать все индексы, которые выходят за пределы, когда вы пользуетесь данным сервисом.

#### QAbstractItemModel

<http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html#details>

QAbstractItemModel класс обеспечивает абстрактный интерфейс для классов модели элементов. Его следует специализировать. Нижележащая структура данные показывается делегатам и представлениям как иерархия таблиц. *Далее идёт некоторая общая информация, которую я уже читал в данной части.*

#### QAbstractListModel

<http://qt-project.org/doc/qt-5.1/qtcore/qabstractlistmodel.html#details>

QAbstractListModel класс обеспечивает абстрактную модель, которая может быть специализирована для создания одномерной модели списка. Это стандартный интерфейс для моделей, которые представляют их данные как простой неиерархический ряд элементов. Он должен быть специализирован. *Затем ведётся речь о его специализации.*

#### QAbstractTableModel

<http://qt-project.org/doc/qt-5.1/qtcore/qabstracttablemodel.html#details>

QAbstractTableModel класс обеспечивает абстрактную модель, которая может быть специализирована для создания моделей таблиц. Класс обеспечивает стандартный интерфейс для моделей, которые представляют их данные как двумерный массив элементов. Он не используется напрямую, но его следует специализировать. *Далее ведётся речь о специализации данного класса.*

#### QModelIndex

<http://qt-project.org/doc/qt-5.1/qtcore/qmodelindex.html#details>

QModelIndex класс используется для локализации данных в модели данных. Он используется как индекс к моделям, порождённым от [QAbstractItemModel](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html). Индекс используется представлениями элементов, делегатами, моделями выделения. Индексы можно получать из модели, но также и модели можно получать из индекса. *Также приведены другие важные функции данного класса.* Индексы следует использовать немедленно, а затем отбрасывать. Структура модели может измениться. Тогда индексы перестанут быть действительными.

#### QPersistentModelIndex

<http://qt-project.org/doc/qt-5.1/qtcore/qpersistentmodelindex.html#details>

QPersistentModelIndex класс используется для локализации данных в модели данных. Этот индекс может сохраняться приложением, а затем использоваться для доступа к информации модели. Данный класс является безопасным, так как он гарантирует, что ссылка на модель будет действительной до тех пор, пока она будет доступна в модели. Хорошим стилем является проверка доступности данных индексов перед доступом к ним.

#### QAbstractProxyModel

<http://qt-project.org/doc/qt-5.1/qtcore/qabstractproxymodel.html#details>

QAbstractProxyMode класс обеспечивает базовый класс для прокси моделей элементов, которые можно сортировать, фильтровать или выполнять другие задачи обработки данных. Он не инстанцируется напрямую. Все стандартные прокси модели производятся из данного класса. Если модель источника удаляется или она не задана, то прокси модель оперирует моделью с пустым указателем.

#### QIdentityProxyModel

<http://qt-project.org/doc/qt-5.1/qtwidgets/model-view-programming.html>

QIdentityProxyModel класс замещает свою исходную модель неизменной. Эта функция подходит для прокси моделей, которые преобразуют функцию [data](http://qt-project.org/doc/qt-5.1/qtcore/qabstractproxymodel.html#data)(). *Есть примеры кода и пояснения.*

#### QItemSelection

<http://qt-project.org/doc/qt-5.1/qtcore/qitemselection.html#details>

QItemSelection класс управляет информацией о выбранных элементах в модели. Класс описывает элементы модели, которые были выбраны пользователем. Это список [QItemSelectionRange](http://qt-project.org/doc/qt-5.1/qtcore/qitemselectionrange.html). Он обеспечивает функции для создания и манипулирования выделениями и выделения диапазона элементов из модели.

#### QItemSelectionModel

<http://qt-project.org/doc/qt-5.1/qtcore/qitemselectionmodel.html#details>

QItemSelectionModel класс следит за выделенными в представлении элементами. В данной модели в функции [select](http://qt-project.org/doc/qt-5.1/qtcore/qitemselectionmodel.html#select)() класс [QItemSelection](http://qt-project.org/doc/qt-5.1/qtcore/qitemselection.html) обеспечивает нужное выделение. Для обновления выделения следует использовать флаги.

#### QItemSelectionRange

<http://qt-project.org/doc/qt-5.1/qtcore/qitemselectionrange.html#details>

класс QItemSelectionRange управляет информацией о диапазоне выбранных элементов в модели.

#### QSortFilterProxyModel

<http://qt-project.org/doc/qt-5.1/qtcore/qsortfilterproxymodel.html#details>

QSortFilterProxyModel класс обеспечивает поддержку для сортировки и фильтрации данных, переданных между моделями и представлением. Данный класс может быть использован для сортировки элементов, фильтрации или и того, и другого. Модель преобразует структуру исходной модели при помощи отображения индексов модели, которую она обеспечивает, в соответствии с другими локациями для использования в представлении. Данный подход позволяет реструктурировать данную исходную модель, также как и позволить рассматривать представления без каких-либо преобразований нижележащих данных, а также без дублирования данных в памяти. *далее приводятся примеры кода.* По умолчанию, модель динамически пересортирует и перефильтровывает данные, где бы не изменялась исходная модель. Это поведение можно переопределить. *Далее приводятся некоторые примеры кода и нюансы использования сортировки, фильтрации и порождения подклассов.*

#### QStringListModel

<http://qt-project.org/doc/qt-5.1/qtcore/qstringlistmodel.html#details>

QStringListModel класс обеспечивает модель, которая обеспечивает строки для представлений. Это редактируемая модель, которая может быть использована для простых случаев, где вам необходимо отображать определённое количество строк в новом виджете, таком как [QListView](http://qt-project.org/doc/qt-5.1/qtwidgets/qlistview.html) или a [QComboBox](http://qt-project.org/doc/qt-5.1/qtwidgets/qcombobox.html). *Есть примеры кода. Довольно полезный класс.*

#### QStandardItem

<http://qt-project.org/doc/qt-5.1/qtgui/qstandarditem.html#details>

QStandardItem класс обеспечивает элемент для использования вместе с классом [QStandardItemModel](http://qt-project.org/doc/qt-5.1/qtgui/qstandarditemmodel.html). Данные объекты обычно содержат текст, иконки или флажки. Каждый элемент может иметь некоторый фон, шрифт. Для каждого элемента существует различные флаги. Каждый элемент может иметь двумерную таблицу дочерних элементов. *Также содержится информация о специализации данного класса для создания частных элементов.*

#### QStandardItemModel

<http://qt-project.org/doc/qt-5.1/qtgui/qstandarditemmodel.html#details>

QStandardItemModel класс обеспечивает основную модель для сохранения частных данных. Данный класс может быть использован как репозиторий для стандартных типов qt. Это модель, где каждая ячейка может содержать также некоторую таблицу. *Затем рассказывается, как создавать список или дерево. Далее приводится хороший пример кода относительно данного класса.*

#### QFileSystemModel

<http://qt-project.org/doc/qt-5.1/qtwidgets/qfilesystemmodel.html#details>

QFileSystemModel класс обеспечивает модель данных для локальной файловой системы. Есть функции для переименования или удаления файлов и директорий, а также для создания новых директорий. Данный класс требует наличия экземпляра приложения GUI. *Далее приводится пример использования данного класса.* Данный класс не извлекает никаких файлов или директорий, пока не вызывается функция [setRootPath](http://qt-project.org/doc/qt-5.1/qtwidgets/qfilesystemmodel.html#setRootPath)(). Это предотвратит любые ненужные запросы файловой системы. *Немного непонятен смысл данного предложения.* Данный класс использует другой поток для своей работы, так что он не будет влиять на Главный поток, который не будет зависать во время запроса файловой системы. Данный класс содержит кэш с и файловой информацией. Кэш автоматически обновляется с использованием класса [QFileSystemWatcher](http://qt-project.org/doc/qt-5.1/qtcore/qfilesystemwatcher.html).

#### QAbstractItemDelegate

<http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractitemdelegate.html#details>

QAbstractItemDelegate класс используется для отображения и редактирования данных из модели. Он обеспечивает общий интерфейс и функциональность для делегатов в qt. Делегаты отображают отдельные данные в представлении и управляют редактированием данных. *Далее приводится пример кода использования данного класса.* Для обеспечения частного редактирования существует два подхода. Первый подход заключается в создании редактора виджета и отображения его прямо на элементе. Для этого следует переопределить функции [createEditor](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractitemdelegate.html#createEditor)(),[setEditorData](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractitemdelegate.html#setEditorData)(),[setModelData](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractitemdelegate.html#setModelData)(). Второй способ – это обрабатывать события пользователя напрямую при помощи переопределения функции [editorEvent](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractitemdelegate.html#editorEvent)().

#### QAbstractItemView

<http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractitemview.html#details>

QAbstractItemView класс обеспечивает базовую функциональность для элементов в классах представления. Это базовый класс для любого представления, которое использует в качестве модели класс [QAbstractItemModel](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html). Этот класс предоставляет поддержку для навигации при помощи клавиатуры и мыши, для прокрутки вьюпорта, для редактирования элементов и выделения. *Далее приводится функциональность, которую реализовывает класс для поддержки навигации. Показан некоторый пример кода касательно полосы прокрутки.* Если вы наследуете данный класс и стремитесь обновить содержание вьюпорта, то вам следует использовать функцию viewport->[update](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractitemview.html#update)() вместо [update()](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#update), так как все операции рисования имеют место в вьюпорте.

#### QColumnView

<http://qt-project.org/doc/qt-5.1/qtwidgets/qcolumnview.html#details>

класс QColumnView обеспечивает реализацию модель/представление для представлений колонки. Он отображает модель в некотором количестве QListView, каждый на каждую иерархию в дереве. *Видимо, он довольно хорош для деревьев. Там есть неплохой визуальный пример.*

#### QDataWidgetMapper

<http://qt-project.org/doc/qt-5.1/qtwidgets/qdatawidgetmapper.html#details>

QDataWidgetMapper класс обеспечивает отображение между секцией данных и виджетами. Данные класс может быть использован для создания умных по отношению к данным виджетов при помощи их отображения на секцию модели данных. Секция – это колонка, если ориентация модели горизонтальна, а иначе – это колонка. Каждый раз, когда текущий индекс изменяется, каждый виджет обновляется данными из модели через свойство, определённое, когда отображение было сделано. *Есть пример кода. Класс хороший. Он позволяет отображать некоторую секцию модели в некоторый набор виджетов, в котором каждый элемент модели изменяется по-своему.* Также в данном классе есть две политики утверждения.

#### QHeaderView

<http://qt-project.org/doc/qt-5.1/qtwidgets/qheaderview.html#details>

QHeaderView класс обеспечивает заголовок строки или колонки для элементов представления. Он используется в заголовках из классов [QTableView](http://qt-project.org/doc/qt-5.1/qtwidgets/qtableview.html) and [QTreeView](http://qt-project.org/doc/qt-5.1/qtwidgets/qtreeview.html). *Далее описываются некоторые функции данного класса.* В данном классе можно реализовать функциональность перемещаемых заголовочных файлов. Заголовки можно стилизовать, но для них нельзя устанавливать делегатов.

#### QItemDelegate

<http://qt-project.org/doc/qt-5.1/qtwidgets/qitemdelegate.html#details>

QItemDelegate класс обеспечивает отображение и редактирование данных из модели. *Далее представлены функции, которые следует переопределить для создания собственного делегата на основе данного класса. Далее приведены роли элементов данных в модели.* Если вы создаёте делегат, который отображает элементы с использованием частного рисовальщика, то важно гарантировать, что делегат может рисовать элементы во всех требуемых состояниях: e.g. selected, disabled, checked. Документация функции paint содержит некоторые подсказки того, как это можно сделать. Также вы можете обеспечить частные редакторы при помощи класса [QItemEditorFactory](http://qt-project.org/doc/qt-5.1/qtwidgets/qitemeditorfactory.html). *Есть ссылка на пример.* В данном случае нет необходимости специализировать класс QItemDelegate. Существует два класса делегатов: QItemDelegate and [QStyledItemDelegate](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleditemdelegate.html). Хотя, делегат по умолчанию является [QStyledItemDelegate](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleditemdelegate.html). Эти два класса являются независимыми альтернативами для рисования и обеспечения редактора для элементов в представлениях. Отличие между ними заключается в том, что [QStyledItemDelegate](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleditemdelegate.html) использует текущий стиль для рисования элементов. Поэтому он рекомендуется к использованию. Код, требуемый для каждого класса, будет одинаковым, несмотря на то, что частный делегат нуждается в стиле для рисования.

#### QItemEditorFactory

<http://qt-project.org/doc/qt-5.1/qtwidgets/qitemeditorfactory.html#details>

QItemEditorFactory класс обеспечивает виджеты для редактирования элементов данных в представлениях и делегатах. Класс [QItemDelegate](http://qt-project.org/doc/qt-5.1/qtwidgets/qitemdelegate.html) использует данный класс для создания редакторов. Фабрика содержит набор [QItemEditorCreatorBase](http://qt-project.org/doc/qt-5.1/qtwidgets/qitemeditorcreatorbase.html) экземпляров, которые являются специальными редакторами, которые производят редакторов для одного конкретного типа [QVariant](http://qt-project.org/doc/qt-5.1/qtcore/qvariant.html). *Далее приводятся некоторые виджеты для некоторых типов данных.* Можно зарегистрировать некоторые дополнительные редакторы или создать новую фабрику.

#### QItemEditorCreatorBase

<http://qt-project.org/doc/qt-5.1/qtwidgets/qitemeditorcreatorbase.html#details>

QItemEditorCreatorBase класс обеспечивает абстрактный базовый класс, который должен быть специализирован, когда вы реализуете новые создатели редакторов элементов. Редактор должен обеспечивать свойство пользователя для данных, которые он редактирует. QItemDelagates могут затем получить доступ к свойству с использованием мета-объектной системы qt, чтобы установить и получить редактируемые данные. Свойство устанавливается как пользовательское при помощи ключевого слова USER. Если редактор не обеспечивает пользовательского совйства, он должен возвращать имя свойства из функции [valuePropertyName](http://qt-project.org/doc/qt-5.1/qtwidgets/qitemeditorcreatorbase.html#valuePropertyName)(); а делегаты будут затем использовать имя для доступа к свойству. Если данное свойство существует, то делегаты не вызывают данную функцию. [QStandardItemEditorCreator](http://qt-project.org/doc/qt-5.1/qtwidgets/qstandarditemeditorcreator.html) – это удобный класс, который можно использовать для регистрации виджетов без необходимости создания подкласса данного класса.

#### QItemEditorCreator

<http://qt-project.org/doc/qt-5.1/qtwidgets/qitemeditorcreator.html#details>

QItemEditorCreator класс обеспечивает возможность создания создателя редактора элемента без специализации класса [QItemEditorCreatorBase](http://qt-project.org/doc/qt-5.1/qtwidgets/qitemeditorcreatorbase.html). Это удобный шаблонный класс. В конструктор передаётся имя свойства, которое содержит редактируемые данные. Используйте данный класс, только если ваш редактор не определил пользовательское свойство. Иначе следует использовать класс [QStandardItemEditorCreator](http://qt-project.org/doc/qt-5.1/qtwidgets/qstandarditemeditorcreator.html). *Всё равно не понимаю насчёт пользовательского свойства.*

#### QStandardItemEditorCreator

<http://qt-project.org/doc/qt-5.1/qtwidgets/qstandarditemeditorcreator.html#details>

QStandardItemEditorCreator класс обеспечивает возможность регистрации виджетов без необходимости специализации класса [QItemEditorCreatorBase](http://qt-project.org/doc/qt-5.1/qtwidgets/qitemeditorcreatorbase.html). *Есть пример кода.* Редактор должен обеспечивать пользовательское свойство, которое должно содержать редактируемые данные. *И я всё ещё не понимаю, что такое пользовательское свойство.*

#### QListView

<http://qt-project.org/doc/qt-5.1/qtwidgets/qlistview.html#details>

QListView класс обеспечивает представление списка или иконки на модель.

#### QListWidgetItem

<http://qt-project.org/doc/qt-5.1/qtwidgets/qlistwidgetitem.html#details>

QListWidgetItem класс обеспечивает элемент для использования с [QListWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qlistwidget.html). Он представляет отдельный элемент в данном виджете. Для более гибких виджетов для списков следует рассмотреть класс [QListView](http://qt-project.org/doc/qt-5.1/qtwidgets/qlistview.html). Элементы списка используются для отображения текста или иконки. Можно также специализировать данный класс.

#### QListWidget

<http://qt-project.org/doc/qt-5.1/qtwidgets/qlistwidget.html#details>

QListWidget класс обеспечивает основанный на элементах виджет списка. Он содержит дополнительный основанный на элементах интерфейс, основанный на добавлении и удалении элементов. *Далее приводятся аспекты использования данного класса.*

#### QStyledItemDelegate

<http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleditemdelegate.html#details>

QStyledItemDelegate класс обеспечивает отображение и редактирование данных из модели. Если делегат не поддерживает рисование типов данных, которые вам необходимы, или вы желаете настроить рисование элементов, вам необходимо специализировать класс QStyledItemDelegate и переопределить функцию [paint](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleditemdelegate.html#paint)() и, возможно, [sizeHint](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleditemdelegate.html#sizeHint)(). *Далее речь ведётся о изменении стиля показа элементов. Данную часть я изучу подробнее в будущем.*

#### QTableView

<http://qt-project.org/doc/qt-5.1/qtwidgets/qtableview.html#details>

QTableView класс обеспечивает реализацию по умолчанию представления таблицы.

#### QTableWidgetSelectionRange

<http://qt-project.org/doc/qt-5.1/qtwidgets/qtablewidgetselectionrange.html#details>

QTableWidgetSelectionRange класс обеспечивает способ взаимодействия с выделением в модели без использования индексов модели и модели выделения. Если в выделении элемент определён как не выделяющийся, то он не будет включен в выделение.

#### QTableWidgetItem

<http://qt-project.org/doc/qt-5.1/qtwidgets/qtablewidgetitem.html#details>

QTableWidgetItem класс обеспечивает элемент для использования с классом [QTableWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qtablewidget.html).

#### QTableWidget

<http://qt-project.org/doc/qt-5.1/qtwidgets/qtablewidget.html#details>

QTableWidget класс обеспечивает основанную на элементах таблицу вместе с моделью по умолчанию. Класс определяет стандартные средства отображения таблицы для приложения.

#### QTreeView

<http://qt-project.org/doc/qt-5.1/qtwidgets/qtreeview.html#details>

QTreeView класс обеспечивает реализацию по умолчанию представления деревьев.

#### QTreeWidgetItem

<http://qt-project.org/doc/qt-5.1/qtwidgets/qtreewidgetitem.html#details>

QTreeWidgetItem класс обеспечивает элемент для использования в удобном классе [QTreeWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qtreewidget.html).

#### QTreeWidget

<http://qt-project.org/doc/qt-5.1/qtwidgets/qtreewidget.html#details>

QTreeWidget обеспечивает представление дерева, которое использует предопределённую модель дерева. *На этом пока с данной частью всё. В будущем следует прояснить вопрос относительно создания редакторов для делегатов, пользовательских свойств, относительно того, чем отличаются удобные виджеты от стандартных представлений. Теперь переходим к графу графических изображений.*

## ГРАФИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ

[*http://qt-project.org/doc/qt-5.1/qtwidgets/graphicsview.html*](http://qt-project.org/doc/qt-5.1/qtwidgets/graphicsview.html)

графическое представление обеспечивает поверхность для управления и взаимодействия с большим количеством частных двумерных графических элементов, и виджеты для визуализации элементов с поддержкой зума и вращения. Включает архитектуру распространения событий, которая позволяет уточнить с вещественной точностью взаимодействие элементов на экране. Элементы могут обрабатывать события клавиш, нажатие мыши, движение, отпускание и двойные щелчки, а также они могут следить за движением мыши. Графические представления используют BSP (Binary Space Partitioning) дерево для обеспечения очень быстрого открытия элементов, и как результат этого, он может визуализировать большие декорации в реальном времени, даже с миллионами элементов. Графическое представление обеспечивает основанный на элементах подход к модели-представлению, во многом подобно [InterView](http://qt-project.org/doc/qt-5.1/qtwidgets/itemviews-interview.html)'s convenience classes [QTableView](http://qt-project.org/doc/qt-5.1/qtwidgets/qtableview.html), [QTreeView](http://qt-project.org/doc/qt-5.1/qtwidgets/qtreeview.html) and [QListView](http://qt-project.org/doc/qt-5.1/qtwidgets/qlistview.html). Несколько представлений могут следить за одним и тем же сином. [QGraphicsScene](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscene.html) обеспечивает син. Син имеет следующие ответственности:

* Обеспечение быстрого интерфейса для управления большим количеством элементов.
* Распространение событий каждому элементу.
* Управление состоянием элемента, таким как выделение или фокус.
* Обеспечение непреобразуемой функциональности рисования; особенно для печати.

Син служит в качестве контейнера для объектов [QGraphicsItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html). Все функции открытия элементов возвращают элементы в нисходящем порядке (это значит первой возвращается самый верхний элемент, а последним – самый нижний). *Есть пример кода.* [QGraphicsScene](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscene.html) архитектура распространения событий планирует события сина для доставки к элементам, и также управляет распространением между элементами. Если син получает событие нажатия мыши в некоторой позиции, он передаёт событие тому элементу, который находится в той позиции. Также данный класс управляет определёнными состояниями элемента, такими как выделение элемента или фокус. *В данном классе есть вся необходимая функциональность для этого.* Также данный класс позволяет рисовать вам части сина на устройство рисования при помощи специальной функции. [QGraphicsView](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsview.html) обеспечивает виджет представления, который визуализирует содержание сина. Вы можете присоединить несколько представлений к одному и тому же сину, чтобы обеспечить несколько вьюпортов для одного и того же набора данных. Также данный виджет обеспечивает полосы прокрутки для навигации в больших синах. Представление получает события ввода от клавиатуры и мыши и преобразует их в события сина. При помощи некоторых функций представление может трансформировать координатную систему сина. Это позволяет обеспечить продвинутые особенности навигации, такие как зум или вращение. А также есть некоторые функции для преобразования между координатами сина и представления. [QGraphicsItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html) является базовым классом для графических элементов на сине. Graphics View обеспечивает несколько стандартных элементов для типичных форм, таких как прямоугольники, эллипсы, и текстовые элементы, Но наиболее мощными являются те классы, которые специализируют [QGraphicsItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html). Данный класс, на ряду со всеми остальными, поддерживает следующие особенности:

* Нажатие мыши, движение, отпускание и двойной щелчок, также как и события покрытия мыши, колёсика и события контекстного меню.
* Фокус ввода клавиатуры и события клавиш.
* Перетаскивание и опускание.
* Группирование, также как и отношения родитель-ребёнок, а также при помощи [QGraphicsItemGroup](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitemgroup.html).
* Детектирование соударений.

Элементы живут в локальной системе координат, поэтому данный класс содержит множество функций для отображения координат между элементом и сином, а также от элемента к элементу. Также он может преобразовывать свою систему координат. Поэтому можно вращать или масштабировать отдельные элементы. Элементы могут содержать другие элементы. Все преобразования родителя наследуются всеми его детьми. Даже если они накапливают преобразования внутри своих собственных функций. [QGraphicsItem::shape](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#shape)() и QGraphicsItem::collidesWith() функции обеспечивают детектирование соударений. Обе эти функции являются виртуальными. При помощи возврата формы ваших элементов как локальных координат [QPainterPath](http://qt-project.org/doc/qt-5.1/qtgui/qpainterpath.html) из [QGraphicsItem::shape](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#shape)() класс элемента будет обрабатывать все детектирования соударений для вас. Если вы желаете ваши собственные детектирования соударений, вы можете переопределить функцию QGraphicsItem::collidesWith(). *Далее приводится список классов из данной части модуля виджетов.*

Графическое представление основано на картезианской системе координат; позиции элементов и геометрия сина представляются при помощи набора двух чисел: координат. В графическом представлении существует три эффективных системы координат: элемента, сина и представления. Син координаты во время рисования соответствуют локальным координатам рисовальщика QPainter, а координаты представления такие же как координаты устройства. Элементы живут в их собственной локальной системе. Их координаты обычно центрируются вокруг центральной точки. При построении собственных примитивов вам следует беспокоиться только о координатах элемента, так как все преобразования можно потом будет сделать. Позиция элемента – это координаты центра элемента в системе координат его родителя. Преобразование элемента и позиция элемента являются относительными к родителю. Син представляет базовую систему координат для всех элементов. Данная система координат описывает позицию каждого элемента верхнего уровня. Координаты представления – это координаты виджета. Каждая единица в координатах представления соответствуют одному пикселу. Она является относительной по отношению к виджету или вьюпорту, а также не поддаётся влиянию сина. Все события первоначально посылаются в координатах представления, а эти координаты следует отобразить на син для взаимодействия с элементами. *Далее приведены функции для преобразования различного типа координат.*

[QGraphicsView](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsview.html) поддерживает те же аффинные преобразования, что и [QPainter](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html) делает через [QGraphicsView::setMatrix](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsview.html#setMatrix)(). Также в данном классе есть поддержка зума и вращения. *Есть пример кода. Также есть ссылка на некоторый пример кода.* Графическое представление обеспечивает однострочную печать через функции рисования. *Есть пример кода использования данных функций.* Разница между функциями рисования представления и сина заключается в том, что первая управляет координатами сина, а вторая координатами представления. Функции сина считаются более предпочтительными для печати целых сегментов сина, которые не преобразуются. Например, печать геометрических данных или текстовых документов. Функции представления подходят для скриншотов. Их поведение по умолчанию заключается в рисовании точного содержания вьюпорта с использованием обеспечиваемого рисовальщика.

[QGraphicsView](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsview.html) предоставляет ту же поддержку перетаскивания и опускания, что и [QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html). Также он предоставляет данную функциональность для сина и для каждого элемента. Представление преобразует события в [QGraphicsSceneDragDropEvent](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscenedragdropevent.html) и передаёт их сину. Син пересылает данное событие ближайшему элементу, который допускает опускание. Затем описывается, как реализовать данные события. *Есть также ссылка на некоторый пример.*

В графическом представлении есть поддержка курсоров и подсказок. *В данном случае есть также ссылка на некоторый пример, который демонстрирует использование данной особенности.*

Графическое представление поддерживает анимацию на нескольких уровнях. Вы можете просто монтировать анимацию с использованием анимации. *В данной части указано, как это делается.* Для этого следует унаследовать [QGraphicsObject](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsobject.html) и связать [QPropertyAnimation](http://qt-project.org/doc/qt-5.1/qtcore/qpropertyanimation.html) с ним. Также можно унаследовать класс от [QObject](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html) and [QGraphicsItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html), а затем установить таймер для управления анимацией.

Устанавливая QGLWidget в качестве вьюпорта для графического представления можно сделать возможным OpenGL рисование. *Есть пример кода.*

Если сделать один элемент дочерним по отношению к другим, то все передвижения и трансформации над ними будут выполняться одинаково. Вдобавок, [QGraphicsItemGroup](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitemgroup.html) – это специальный элемент, который позволяет создавать группы элементов.

Есть поддержка для геометрии и элементов, умных с точки зрения макетов. Этот специальный базовый элемент подобен на тот, который обеспечивает виджет, но в отличие от виджета он не наследует [QPaintDevice](http://qt-project.org/doc/qt-5.1/qtgui/qpaintdevice.html); но наследует [QGraphicsItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html). Это позволяет вам написать полные виджеты с событиями, сигналами и слотами, а также подсказками размеров и политиками, и вы можете также управлять геометриями ваших виджетов в макетах при помощи [QGraphicsLinearLayout](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicslinearlayout.html) and [QGraphicsGridLayout](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsgridlayout.html).

[QGraphicsWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicswidget.html) обеспечивает лучшее из двух частей: экстра функциональность [QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html), такую как стили, шрифты, палитра, направление макета и его геометрия, а также независимость разрешения, и поддержку трансформации из [QGraphicsItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html). В данном виджете все операции выполняются с вещественной точностью. Возможности данного класса будут расти по мере роста обратной связи от сообщества qt. [QGraphicsLayout](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicslayout.html) – это часть второго поколения макетов, разработанного специально для [QGraphicsWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicswidget.html).

Графическое представление обеспечивает бесшовную поддержку для внедрения любого виджета в син. Например, можно внедрить кнопку или окно редактирования. *для этого есть специальная функция. А также можно создать экземпляр* [QGraphicsProxyWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsproxywidget.html) для внедрения вашего виджета вручную. Данный класс предполагает, что железо пользователя обеспечивает достаточно быстрые операции с вещественными числами. Поэтому на некоторых устройствах некоторые эффекты могут быть медленнее, чем ожидается. Для этих целей можно использовать OpenGL для рисования сина.

#### QGraphicsEffect

<http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicseffect.html#details>

QGraphicsEffect класс является базовым классом для всех графических эффектов. Вы можете использовать один из стандартных эффектов, или использовать свой собственный эффект при помощи специализации данного класса. *Далее рассказывается про написание собственных эффектов.*

#### QGraphicsGridLayout

<http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsgridlayout.html#details>

QGraphicsGridLayout класс обеспечивает макет в виде сетки для управления виджетами в графическом представлении. *Далее приводится инструкция использования данного класса.* Данный класс принимает во внимание политики и подсказки размеров каждого элемента.

<http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#details>

QGraphicsItem класс является базовым классом для всех графических элементов в классе [QGraphicsScene](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscene.html). Он позволяет вам создавать ваши собственные элементы. *Затем приводится набор стандартных графических элементов для общих форм.* Вся геометрическая информация об элементе основана на его логической системе координат. Только функция [pos](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#pos)() возвращает позицию в родительской системе координат. Можно на элемент устанавливать различного рода флаги. Для создания собственных элементов следует специализировать данный класс и переопределить его две чисто виртуальные функции [boundingRect](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#boundingRect)() и [paint](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#paint)(), которая реализует само рисование. [QGraphicsScene](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscene.html) ожидает, что все элементы [boundingRect](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#boundingRect)() and [shape](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#shape)() остаются неизменными, несмотря на уведомление их. Если вы желаете изменить геометрию элемента любым способом, вам следует сначала вызвать [prepareGeometryChange](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#prepareGeometryChange)(), чтобы позволить [QGraphicsScene](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscene.html) обновить его бухгалтерию. Детектирование соударений может быть сделано двумя способами:

1. Переопределение [shape](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#shape)() для возврата точной формы вашего элемента, или полагаясь на реализацию по умолчанию [collidesWithItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#collidesWithItem)() для взаимодействия формы с формой. Это может быть гораздо дороже, если формы очень сложные.
2. Переопределить функцию [collidesWithItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#collidesWithItem)(), чтобы обеспечить ваш собственный алгоритм соударения.

Над элементами можно совершать преобразования. Многие аспекты данной темы были рассмотрены ранее. Некоторые трансформации производят различный выход в зависимости от их порядка. Функция [paint](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#paint)() вызывается для рисования содержимого элемента. Также можно перерисовывать элемент. При этом, если он не будет виден, то перерисовывание не произойдёт. Это отличает данную функцию от [QWidget::repaint](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#repaint)(). Элементы рисуются, начиная с родительского. Но этот порядок можно изменять при помощи функции [setZValue](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#setZValue)(). Все элементы рисуются в определённом стабильном порядке, и этот порядок определяет, какие элементы получат ввод мыши первыми, когда вы кликаете на син. Обычно вам не приходится беспокоится о сортировке, так как элементы следуют естественному порядку, установленному логической структурой сина. (в порядке добавления). *Есть также ссылка на пример.* Для продвинутых пользователей есть способы изменить сортировку элементов:

* [setZValue](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#setZValue)()
* [stackBefore](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#stackBefore)()
* [ItemStacksBehindParent](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#GraphicsItemFlag-enum) flag – для расположения элемента под его родителем.

QGraphicsItem получает события от [QGraphicsScene](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscene.html) через виртуальную функцию [sceneEvent](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#sceneEvent)(). Эта функция распространяет большинство общих событий набору удобных обработчиков событий: *далее они перечислены.* Можно также устанавливать фильтры событий. Эта функциональность располагается отдельно от регулярных фильтров событий qt. *Далее рассказывается об установке фильтра.* Иногда полезно зарегистрировать некоторые частные данные внутри элемента. Это делает элемент частным в отличие от стандартного элемента. Для этого есть функции [setData](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#setData)() и [data](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#data)(), которые созданы для удобства пользователя.

#### QGraphicsObject

<http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsobject.html#details>

QGraphicsObject класс обеспечивает базовый класс для всех графических элементов, которые требуют сигналов, слотов и свойств. Данный класс расширяет [QGraphicsItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html) при помощи механизмов сигналов и слотов, а также механизма свойств, доступных из QObject. Каждый объект может быть построен при помощи родительского элемента. Это гарантирует, что элемент будет удалён при удалении его родительского элемента. Для построения отношений между родителем и детьми следует использовать функции [QGraphicsItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html), а не [QObject](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html).

#### QAbstractGraphicsShapeItem

<http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractgraphicsshapeitem.html#details>

QAbstractGraphicsShapeItem класс обеспечивает общую основу для всех элементов путей. Данный класс не реализует полностью элемент; в частности, он не реализует [boundingRect](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#boundingRect)() and [paint](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#paint)(). Этот элемент можно специализировать для обеспечения простой базовой реализации асессоров для пера и кисти элемента.

#### [QGraphicsRotation](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsrotation.html)

#### [QGraphicsScale](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscale.html)

#### КЛАССЫ РАЗНЫХ ТИПОВ ЭЛЕМЕНТОВ СИНА

[QGraphicsPathItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicspathitem.html), [QGraphicsRectItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsrectitem.html), [QGraphicsEllipseItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsellipseitem.html), [QGraphicsPolygonItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicspolygonitem.html), [QGraphicsLineItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicslineitem.html), [QGraphicsPixmapItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicspixmapitem.html), [QGraphicsTextItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicstextitem.html), [QGraphicsSimpleTextItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicssimpletextitem.html) *– разновидности различных элементов, предназначение которых понятно из их названия*.

#### QGraphicsItemGroup

<http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitemgroup.html#details>

QGraphicsItemGroup класс обеспечивает контейнер, который обрабатывает группу элементов как единичный элемент. Он используется, если пользователь хочет объединить много маленьких элементов в одно целое, которое можно копировать и передвигать. Если вы желаете сохранить элементы внутри других элементов, то вы можете использовать любой [QGraphicsItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html) напрямую при помощи передачи подходящего родителя функции [setParentItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html#setParentItem)(). По отношению к группе все элементы обрабатываются как преобразуемые. Способы создания группы: [QGraphicsScene::createItemGroup](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscene.html#createItemGroup)(); ручное построение группы и добавление в неё элементов. *Есть пример кода уничтожения группы.*

#### QGraphicsLayout

<http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicslayout.html#details>

QGraphicsLayout класс обеспечивает базовый класс для всех макетов в графическом представлении. Это абстрактный класс для организации детей [QGraphicsWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicswidget.html). *Далее описывается данный класс. Его лучше изучать после изучения примеров.*

#### QGraphicsLayoutItem

<http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicslayoutitem.html#details>

QGraphicsLayoutItem класс может быть унаследован, чтобы позволить управлять вашими частными элементами при помощи макетов. Это абстрактный класс, который определяет набор виртуальных функций, описывающих размеры, политики, и подсказки для любого объекта, организованного при помощи класса [QGraphicsLayout](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicslayout.html). *Далее есть описание данного класса.*[QGraphicsLinearLayout](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicslinearlayout.html) – горизонтальные и вертикальные макеты для управления виджетами в графическом представлении.

#### QGraphicsProxyWidget

<http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsproxywidget.html#details>

QGraphicsProxyWidget класс обеспечивает прокси слой для внедрения [QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html) in a [QGraphicsScene](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscene.html). *Далее идут примеры кода данного класса, а также его описание, откуда всё следует.*

#### QGraphicsScene

<http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscene.html#details>

QGraphicsScene класс обеспечивает поверхность для управления большим количеством двумерных графических элементов. Класс служит контейнером для QGraphicsItems. Он используется совместно с [QGraphicsView](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsview.html) для визуализации графических элементов, таких как линии, прямоугольники, текст или даже частные элементы, на двумерной поверхности. Также данный класс предоставляет вам функциональность, которая позволяет вам как эффективно определить локацию элементов, так и для определения, какие элементы видимы внутри произвольной области сина. Вместе с виджетом [QGraphicsView](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsview.html), вы можете или визуализировать полный син, или зумировать в представлении только части сина. *Есть пример кода.* QGraphicsScene не имеет сам по себе визуального представления. Он просто управляет элементами. Для визуализации сина следует создать [QGraphicsView](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsview.html) виджет. *Есть удобные функции для добавления элементов.* Для эффективного управления положением элементов класс QGraphicsScene использует алгоритм индексирования. По определению создаётся BSP. Алгоритм подходит для больших синов, где большинство элементов остаются статичными. Можно также установить ограничивающий прямоугольник сина. Одной из сильных сторон сина является его способность эффективно определить положение элементов. Даже для миллиона элементов [items](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscene.html#items)() функция определит положение элементов за миллисекунды. Также син может содержать информацию о выделении элементов в сине. Также данный класс отвечает за распространение событий от [QGraphicsView](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsview.html). для посылки события сину вы строите событие, которое наследует [QEvent](http://qt-project.org/doc/qt-5.1/qtcore/qevent.html), и затем высылаете его с использованием, например, [QApplication::sendEvent](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html#sendEvent)(). [event](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscene.html#event)() ответственна за высылку события отдельным элементам. Некоторые общие события обрабатываются при помощи удобных обработчиков событий. События клавиш отсылаются элементу с фокусом. Есть функции для его установки. Для отображения эффектов, связанных с прохождением мыши, син отправляет *hover events*. Все события мыши высылаются текущему *mouse grabber* элементу.

#### QGraphicsSceneEvent

<http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicssceneevent.html#details>

QGraphicsSceneEvent класс обеспечивает базовый класс для всех графических представлений, связанных с событиями. Когда [QGraphicsView](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsview.html) получает события qt мыши, клавиатуры, или события перетаскивания-опускания, он преобразует их в экземпляры подклассов QGraphicsSceneEvent и направляет их в [QGraphicsScene](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscene.html), который он отображает. Син затем передаёт события подходящим элементам. *Далее приводятся некоторые пояснения к данному классу.*

#### КЛАССЫ СОБЫТИЙ СИН ГРАФА

[QGraphicsSceneMouseEvent](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscenemouseevent.html), [QGraphicsSceneWheelEvent](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscenewheelevent.html), [QGraphicsSceneContextMenuEvent](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscenecontextmenuevent.html), [QGraphicsSceneHoverEvent](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscenehoverevent.html), [QGraphicsSceneHelpEvent](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscenehelpevent.html), [QGraphicsSceneDragDropEvent](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscenedragdropevent.html) – *классы, предназначение которых понятно из их названия.*

QGraphicsSceneResizeEvent класс обеспечивает события для изменения размеров виджета в графическом представлении. QGraphicsSceneMoveEvent класс обеспечивает события для передвижения виджета в графическом представлении.

#### QGraphicsTransform

<http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicstransform.html#details>

QGraphicsTransform класс является абстрактным базовым классом для построения продвинутых преобразований для QGraphicsItems. В качестве альтернативы QGraphicsItem::transform данный класс позволяет вам создать и управлять продвинутыми преобразованиями, которые могут быть настроены независимо с использованием специальных свойств. [QGraphicsItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html) позволяет вам присвоить любое количество экземпляров QGraphicsTransform к [QGraphicsItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html). Данный класс в частности полезен для анимаций, так как при использовании второго варианта нет прямого перехода между различными преобразованиями. С использованием данного класса вы можете интерполировать значения свойств каждого независимого преобразования. Результирующая операция затем объединяется в единое преобразование, которое применяется к [QGraphicsItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html). При помощи специализации данного класса вы можете создавать собственные преобразования, при этом следует переопределить чисто виртуальную функцию [applyTo](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicstransform.html#applyTo)() (*так что, этот класс абстрактный ???*).

#### QGraphicsView

<http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsview.html#details>

QGraphicsView класс обеспечивает виджет для отображения содержания [QGraphicsScene](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscene.html). Данный класс визуализирует содержание сина в окне с прокруткой. Класс можно использовать для визуализации всего сина или некоторой его части. *В данной части рассказывается, как это устанавливается.* По умолчанию элементы рисуются с использованием обычного рисовальщика. Также можно изменять виджет отображения, производить различные трансформации. Также данный класс обеспечивает трансляцию событий. Вы также можете создать ваше собственное взаимодействие с сином при помощи специализации данного класса и переопределения обработчиков событий мыши и клавиатуры. *Далее приводятся некоторые функции, которые облегчают работу для этого.*

#### QGraphicsWidget

<http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicswidget.html#details>

QGraphicsWidget класс является базовым классом для всех элементов виджета в [QGraphicsScene](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscene.html). Это расширенный базовый элемент, который обеспечивает экстра функциональность для [QGraphicsItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html). Он похож во многих аспектах на [QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html). *Они все далее перечислены.* Это не абстрактный класс. Данный класс может быть использован как базовый класс для ваших собственных частных элементов, если вы требуете продвинутой обработки фокуса ввода, например таб фокус, активация или макеты. Виджеты, основанные на [QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html), могут быть напрямую внедрены в [QGraphicsScene](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsscene.html) с использованием класса [QGraphicsProxyWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsproxywidget.html). *Далее представлена таблица различий между виджетами и данным классом. Далее представлена таблица атрибутов класса виджета, которые доступны в данном классе.* Также в данном классе следует использовать только функции от [QGraphicsItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html), но не от [QObject](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html), чтобы управлять взаимоотношениями между дочерними и родительскими объектами. Функции [QObject::parent](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html#parent)() для таких виджетов лучше всегда возвращать нуль.

#### QStyleOptionGraphicsItem

<http://qt-project.org/doc/qt-5.1/qtwidgets/qstyleoptiongraphicsitem.html#details>

QStyleOptionGraphicsItem класс используется для описания параметров, необходимых для рисования [QGraphicsItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html). По соображениям производительности доступ к переменным членам является прямым. Их следует воспринимать просто как параметры.*Есть ссылка на пример использования данного класса.*

## РУКОВОДСТВО ПО ВИДЖЕТАМ

<http://qt-project.org/doc/qt-5.1/qtwidgets/widgets-tutorial.html>

виджеты – это базовые строительные блоки для графического интерфейса пользователя приложений, построенных при помощи qt. Каждый элемент интерфейса – это виджет, который помещён в некоторую область окна или который изображается как независимое окно. [QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html) не является абстрактным классом. Он может быть использован как контейнер для других виджетов, а также он может быть специализирован для создания нового виджета. Он часто используется в качестве окна, в котором размещаются другие виджеты. Виджеты могут иметь родителей. Каждый дочерний виджет располагается в области, занятой его родительским виджетом. При удалении родительского виджета его дочерние виджеты также удаляются. *Далее рассказывается про функцию main(). Далее есть ссылки на простые примеры касательно виджетов. Далее показано, как модифицировать или перекомпилировать примеры.*

### БАЗОВЫЕ ВИДЖЕТЫ

<http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#details>

QWidget класс является базовым классом для всех объектов интерфейса пользователя.

Виджет – это атом интерфейса пользователя: он получает события мыши, клавиатуры и другие события от оконной системы, а также рисует представление себя на экране. Каждый виджет является прямоугольным, а также они отсортированы в z порядке. Виджет усекается своим родителем и виджетами впереди данного.

Виджет, который не внедрён в родительский виджет, называется окном. Обычно окна имеют и название.

Конструктор каждого виджета принимает один или два стандартных аргуемнта:

* QWidget \*parent = 0 – это родитель виджета.
* Qt::WindowFlags f = 0 (where available) устанавливает флаги окна.

Подклассы данного класса обеспечивают реальную функциональность некоторых функций, которые обычно не используются самим данным классом. Когда виджет используется как контейнер для группы нескольких виджетов, то он известен как композитный виджет. Они могут быть созданы при помощи построения виджета с требуемыми визуальными свойствами - [QFrame](http://qt-project.org/doc/qt-5.1/qtwidgets/qframe.html), например, - и добавление дочерних виджетов к нему, обычно управляемое макетом.

Так как QWidget является также подклассом [QPaintDevice](http://qt-project.org/doc/qt-5.1/qtgui/qpaintdevice.html), подклассы могут быть использованы для отображения частного содержания, которое собирается с использованием набора рисований при помощи экземпляра класса QPainter. Каждый виджет выполняет все свои операции рисования внутри функции [paintEvent](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#paintEvent)(). Она вызывается всякий раз, как виджет следует перерисовать, является ли это результатом некоторого внешнего изменения или когда этого требует приложение. *Есть ссылка на пример использования данной функции для управления виджетом.*

При реализации нового виджета, почти всегда полезно переопределить функцию [sizeHint](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#sizeHint-prop)(), чтобы обеспечить разумные размеры по умолчанию для виджета и установить корректную политику размеров при помощи функции [setSizePolicy](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#sizePolicy-prop)(). По умолчанию, композитные виджеты, которые не обеспечивают подсказку размеров, будут иметь размер в соответствии с требованиями к пространству его дочерних виджетов. Размер виджетов высокого уровня ограничивается 2/3 размерами рабочего стола. Вы можете однако изменить размеры виджета, если эти размеры не являются адекватными.

Виджеты отвечают событиям, которые обычно вызываются действиями пользователей. Qt доставляет события виджетам при помощи вызова специальных обработчиков событий с экземпляром одного из подклассов класса [QEvent](http://qt-project.org/doc/qt-5.1/qtcore/qevent.html), содержащего информацию о событии. Если вы имеете дело с композитным виджетом, то вам не следует использовать в некоторых случаях обработчики событий. Вместо этого вызовете функцию для дочернего виджета [underMouse](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#underMouse)(). *Есть ссылка на пример, посвящённый как раз обработке событий.* Вам нужно будет обеспечить поведение и содержание ваших собственных виджетов. *Далее описываются все обработчики событий для виджетов.* Также есть несколько других непрозрачных событий, описанных в документации для [QEvent::Type](http://qt-project.org/doc/qt-5.1/qtcore/qevent.html#Type-enum). Для их обработки следует переопределить функцию [event](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#event)() напрямую.

*Далее приведены группы свойств виджета. Очень замечательная таблица для будущей работы.*

Вдобавок к стандартным стилям виджета для каждой платформы, виджеты также могут быть стилизованы в соответствии с правилами, определёнными в шите стилей.

Виджет автоматически double-buffers своё рисование, так что нет необходимости для написания кода двойной буферизации в [paintEvent](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#paintEvent)(), чтобы избежать мерцания. *Далее приводится не очень понятный для меня пример. Далее идёт рекомендация, как быстро обновлять фоновый цвет частных виджетов*. *Далее говорится о том, какое свойство лучше установить, если нежелательно, чтобы фон виджета перерисовывался (например, при проигрывании внутри него некоторого видео). Далее показан пример, в котором демонстрируется некоторое неожиданное поведение виджета при распространении фона от родительского виджета на дочерние.*

Также возможно создавать окна с полупрозрачными регионами. Для этого следует установить определённое свойство. *Также есть некоторые замечания относительно системы по поводу использования данного свойства.*

Чуждые виджеты – это виджеты, неизвестные оконной системе. Они не имеют нативной обработки окна, связанной с ними. Эта особенность значительно ускоряет рисование виджета, изменение его размеров, а также устраняет мерцание. *Далее показано, как установить нативные виджеты. Мне многое из данной части непонятно.*

#### КЛАССЫ ВИДЖЕТОВ

[QCheckBox](http://qt-project.org/doc/qt-5.1/qtwidgets/qcheckbox.html) – флажок.

[QComboBox](http://qt-project.org/doc/qt-5.1/qtwidgets/qcombobox.html) – выпадающий список.

#### QCommandLinkButton

<http://qt-project.org/doc/qt-5.1/qtwidgets/qcommandlinkbutton.html#details>

QCommandLinkButton виджет обеспечивает Vista стиля кнопку командной связки. Это новый элемент управления, введённый в Windows виста.

#### Виджеты редактирования дат и времён

<http://qt-project.org/doc/qt-5.1/qtwidgets/qdatetimeedit.html#details>

QDateTimeEdit класс обеспечивает виджет для редактирования дат и времён. *Есть примеры кода, иллюстрации и описание.* QTimeEdit класс обеспечивает виджет для редактирования времени, основанного на [QDateTimeEdit](http://qt-project.org/doc/qt-5.1/qtwidgets/qdatetimeedit.html).QDateEdit класс обеспечивает виджет для редактирования дат, основанных на виджете [QDateTimeEdit](http://qt-project.org/doc/qt-5.1/qtwidgets/qdatetimeedit.html).

#### QDial

<http://qt-project.org/doc/qt-5.1/qtwidgets/qdial.html#details>

QDial класс обеспечивает круглый контроллер диапазона, на подобие спидометра. Он используется, когда пользователь желает управлять значением внутри определённого в программе значения. Он ведёт себя подобно ползунку, так как наследует [QAbstractSlider](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractslider.html).

#### QFocusFrame

<http://qt-project.org/doc/qt-5.1/qtwidgets/qfocusframe.html#details>

QFocusFrame класс обеспечивает фрейм фокуса, который может быть вне нормальной области рисования виджета. Обычно приложению не нужен данный класс, так как [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html) будет обрабатывать эти детали для вас. Пользователь может использовать данный класс для определения фокуса виджета за пределами области его рисования. Но это не обязательно.

#### QFontComboBox

<http://qt-project.org/doc/qt-5.1/qtwidgets/qfontcombobox.html#details>

QFontComboBox виджет является выпадающим списком, который позволяет пользователю выбирать семейство шрифтов. Он часто используется в панелях инструментов вместе с [QComboBox](http://qt-project.org/doc/qt-5.1/qtwidgets/qcombobox.html) для управления размера шрифта, а также две кнопки [QToolButton](http://qt-project.org/doc/qt-5.1/qtwidgets/qtoolbutton.html)s для полужирного или курсива.

#### [QLabel](http://qt-project.org/doc/qt-5.1/qtwidgets/qlabel.html)

#### QLCDNumber

<http://qt-project.org/doc/qt-5.1/qtwidgets/qlcdnumber.html#details>

QLCDNumber виджет отображает число вместе с LCD-like цифрами. *Что это такое?* Данный класс внутренне является очень древней частью qt. *Мне непонятно его предназначение.*

#### [QLineEdit](http://qt-project.org/doc/qt-5.1/qtwidgets/qlineedit.html)

#### QMenu

<http://qt-project.org/doc/qt-5.1/qtwidgets/qmenu.html#details>

QMenu класс обеспечивает меню виджет для использования в меню, контекстных меню и других всплывающих меню. Это меню выбора. Это может быть выпадающее меню или отдельное контекстное меню. *Далее описываются различные варианты меню.* Меню состоит из действий. Действие рисуется при помощи [QStyle](http://qt-project.org/doc/qt-5.1/qtwidgets/qstyle.html). Действия могут иметь текстовую метку, необязательную иконку, нарисованную слева, а также последовательность горячих клавиш (Ctrl+X). Есть четыре типа действий: разделители, действия, которые показывают подменю, виджеты, и действия, которые выполняют действие. При вставке действия вы обычно определяете получателя и слот. *Далее описано, что это означает с точки зрения написания кода.* Также данный класс может представлять отрывное меню. Это окно верхнего уровня, которое содержит копию меню. Это позволяет пользователю отрывать часто используемые меню и располагать их в удобных для себя местах. *Далее показывается, как это можно осуществить.* На Windows отрывные меню используются редко и пользователи с этим не очень знакомы.

#### [QWidgetAction](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidgetaction.html)

[QWidgetAction](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidgetaction.html) класс позволяет вставлять в меню виджеты. Действия могут также быть добавлены в виджеты. *Далее приводятся некоторые замечания для некоторых платформ.*

#### Некоторые другие классы виджетов

[QProgressBar](http://qt-project.org/doc/qt-5.1/qtwidgets/qprogressbar.html), [QPushButton](http://qt-project.org/doc/qt-5.1/qtwidgets/qpushbutton.html), [QRadioButton](http://qt-project.org/doc/qt-5.1/qtwidgets/qradiobutton.html) – *следует затем определить, как делать их взаимно исключающими.*

#### QScrollArea

[*http://qt-project.org/doc/qt-5.1/qtwidgets/qscrollarea.html#details*](http://qt-project.org/doc/qt-5.1/qtwidgets/qscrollarea.html#details)

QScrollArea класс обеспечивает представление для прокрутки на другом виджете. Он используется для отображения содержания дочернего виджета внутри фрейма. Если виджет превышает размер фрейма, представление может обеспечить полосы прокрутки, так что полная область дочерних виджетов может быть просмотрена. Дочерний виджет устанавливается при помощи функции [setWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qscrollarea.html#setWidget)(). *Есть пример кода, который показывает использование данного класса.* Внешность полос прокрутки можно контролировать при помощи наследуемой функциональности от класса [QAbstractScrollArea](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractscrollarea.html). Можно изменять политики полос прокрутки, приводить их в порядок динамически. Дочерний виджет может быть с изменяемым размером. *Среди указываемых функций предназначение некоторых функций мне осталось не очень понятным.* При использовании данного класса по отношению к некоторому виджету важно гарантировать, что подсказки размеров виджета установлены в правильные значения. *Так же есть некоторое замечание по отношению к виджетам, внутри которых используются макеты. Но я пока не очень понял данные замечания.*

#### QScrollBar

<http://qt-project.org/doc/qt-5.1/qtwidgets/qscrollbar.html#details>

QScrollBar виджет обеспечивает вертикальные и горизонтальные полосы прокрутки. Полоса прокрутки – это элемент управления, который позволяет пользователю получить доступ к частям документа, которые являются большими, чем виджет, используемый для его отображения. Он обеспечивает визуальное показывание текущей позиции пользователя внутри документа и количество документа, которое видимо. Обычно они снабжены другими элементами управления, которые обеспечивают более точную навигацию. Qt отображает полосы прокрутки способом, который подходит для каждой платформы. Если вам нужна область прокрутки для другого виджета, то лучше использовать [QScrollArea](http://qt-project.org/doc/qt-5.1/qtwidgets/qscrollarea.html), так как он обеспечивает виджет вьюпорт и полосы прокрутки. Данный класс полезен, если вам необходимо реализовать подобную функциональность для определённых виджетов с использованием класса [QAbstractScrollArea](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractscrollarea.html); например, если вы решили специализировать [QAbstractItemView](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractitemview.html). Для большинства других ситуаций, где используется ползунок для получения значения внутри данного диапазона, [QSlider](http://qt-project.org/doc/qt-5.1/qtwidgets/qslider.html) класс является более подходящим для ваших нужд. *Далее описывается структура полосы прокрутки.*

Заметьте, что диапазон используемых значений независим от действительных размеров виджета полосы прокрутки. Вам нет необходимости принимать это во внимание, когда вы выбираете значения для диапазона и шага страницы. В отличие от ползунка в случае строки прокрутки длина ползунка является важной, так как она также несёт некоторую информацию. *Далее приводится формула, которая позволяет связать размер документа с численными значениями полосы прокрутки.*

Данный класс предоставляет только целочисленные значения. Полосы прокрутки не могут качественно отображать диапазоны размером более примерно 100000 пикселов. Кроме этого пользователю становится тяжело управлять ползунком с использованием клавиатуры или мыши, а стрелки прокрутки имеют ограниченное использование. *Далее описаны различные свойства данного класса.* Также для полосы прокрутки предусмотрено взаимодействие с клавиатурой.

#### QSizeGrip

<http://qt-project.org/doc/qt-5.1/qtwidgets/qsizegrip.html#details>

QSizeGrip класс обеспечивает обработку изменения размеров окон верхнего уровня. В X11 его поведение может отличаться от типичного поведения менеджера окон в данной операционной системе. Поместите данный виджет в любом месте дерева виджетов и пользователь сможет использовать его для изменения размеров окна верхнего уровня или любого другого виджета с установленным флагом [Qt::SubWindow](http://qt-project.org/doc/qt-5.1/qtcore/qt.html#WindowType-enum). Обычно его следует располагать в правом нижнем углу. Если в вашем окне есть строка состояния, то он уже встроен в неё и его не следует использовать. Данный класс наследует [QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html) и переопределяет функции [mousePressEvent()](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#mousePressEvent) and [mouseMoveEvent()](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#mouseMoveEvent) для функциональности изменения размера и функцию [paintEvent()](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#paintEvent) для рисования виджета изменения размера.

#### QSlider

<http://qt-project.org/doc/qt-5.1/qtwidgets/qslider.html#details>

QSlider виджет обеспечивает вертикальный или горизонтальный ползунок. Это классический виджет для управления ограниченным значением. Большинство функциональности данного класса взято из класса [QAbstractSlider](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractslider.html). Данный класс предоставляет функциональность управления галочками. Данный класс обеспечивает только целочисленные значения. Но использовать данный виджет для пользователя, если диапазон значений очень велик, проблематично. *В конце описан интерфейс клавиатуры данного класса.*

#### QDoubleSpinBox

<http://qt-project.org/doc/qt-5.1/qtwidgets/qdoublespinbox.html#details>

QDoubleSpinBox класс обеспечивает счётчик для вещественных чисел. Данный виджет округляет вещественные числа в соответствии с текущей установленной точностью. Отображаемое значение может сопровождаться произвольными строками, например, для указания единицы измерения. Также желательно для пользователя всегда иметь некоторое значение по умолчанию.

#### QSpinBox

<http://qt-project.org/doc/qt-5.1/qtwidgets/qspinbox.html#details>

QSpinBox класс обеспечивает виджет счётчик. Он разработан для обработки целочисленных значений. *Также показано, как создавать подкласс данного класса с целью лучшего контроля поясняющего текста. Есть пример кода и ссылка на полноценный пример.*

#### QTabBar

<http://qt-project.org/doc/qt-5.1/qtwidgets/qtabbar.html#details>

QTabBar класс обеспечивает полосу табуляции, для использования табулированных диалогов. Данный класс можно прямо использовать. Он рисует табуляции с использованием одной из предопределённых форм и испускает сигнал, когда таб выбирается. Он может быть специализирован для того чтобы адаптировать его вид и ощущение. Также qt обеспечивает готовый класс [QTabWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qtabwidget.html). Далее перечислены атрибуты, которые имеет каждая табуляция. Свойство формы определяет внешность табуляции. Наиболее важной частью программного интерфейса данного класса является сигнал [currentChanged](http://qt-project.org/doc/qt-5.1/qtwidgets/qtabbar.html#currentIndex-prop)(). Он изменяется всякий раз при изменении табуляции. Также есть слот для программного выбора табуляции. Данный виджет создаёт автоматически мнемонические ключи. Далее перечислены функции, которые нужно переопределить для адаптации табуляции или для сохранения в нём экстра данных.

#### QTabWidget

<http://qt-project.org/doc/qt-5.1/qtwidgets/qtabwidget.html#details>

QTabWidget класс обеспечивает стек виджетов табуляции. Виджет табуляции обеспечивает полосу табуляции и область страницы, которая используется для отображения связанных с каждой табуляцией страниц. В области страницы показана только текущая страница, а все остальные страницы спрятаны. Пользователь может показать разные страницы, кликая на соответствующую табуляцию. Каждой странице соответствует некоторый отдельный виджет. Д*алее указан обычный способ использования данного класса.* Каждая табуляция может быть или доступна или недоступна. Но даже если табуляция является недоступной, то все виджеты страницы всё равно могут быть видимыми. Виджеты табуляции могут быть очень хорошим способом разделения сложных диалогов. Альтернативой является использование класса [QStackedWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qstackedwidget.html), для которого вы обеспечиваете некоторые средства для навигации между страницами, например, [QToolBar](http://qt-project.org/doc/qt-5.1/qtwidgets/qtoolbar.html) или a [QListWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qlistwidget.html). Большиснтво функциональности данного класса обеспечивается классами [QTabBar](http://qt-project.org/doc/qt-5.1/qtwidgets/qtabbar.html) и [QStackedWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qstackedwidget.html).

#### QToolBox

<http://qt-project.org/doc/qt-5.1/qtwidgets/qtoolbox.html#details>

QToolBox класс обеспечивает колонку табулированных элементов виджета. Данный виджет представляет из себя колонку некоторых элементов, которые являются виджетами. Между данными элементами можно переключаться. *Для лучшего понимания следует экспериментально изучить данный класс.*

#### QToolButton

<http://qt-project.org/doc/qt-5.1/qtwidgets/qtoolbutton.html#details>

QToolButton класс обеспечивает кнопку быстрого доступа к командам или настройкам, обычно используемым внутри [QToolBar](http://qt-project.org/doc/qt-5.1/qtwidgets/qtoolbar.html). Кнопка инструмент – это специальная кнопка, которая обеспечивает быстрый доступ к определённым командам или настройкам. В отличие от обычной кнопки команды, кнопка инструмент обычно не показывает текстовую метку, но показывает вместо этого иконку. Они обычно создаются при создании новых экземпляров [QAction](http://qt-project.org/doc/qt-5.1/qtwidgets/qaction.html) при помощи [QToolBar::addAction](http://qt-project.org/doc/qt-5.1/qtwidgets/qtoolbar.html#addAction)() или для существующих действий. Они также могут быть помещены в любой другой виджет. Одним из классических применений такого рода кнопки является выбор инструментов. Например, выбор кистей в программе для рисования. Это может быть реализовано при помощи использования QToolButton как кнопки-переключателя. *Далее рассказывается про непонятное мне понятие* auto-raising. Можно определять различные растры для действительного и нет состояния. Можно устанавливать внешность стиль кнопки. Также данная кнопка может предлагать дополнительные выборы во всплывающем меню. *Видимо, это когда наводишь на некоторую кнопку курсор, а затем под ней возникают другие кнопки, которые указывают на некоторые другие варианты.*

## КЛАССЫ СЛОЖНЫХ ВИДЖЕТОВ

#### QColumnView

<http://qt-project.org/doc/qt-5.1/qtwidgets/qcolumnview.html#details>

QColumnView класс обеспечивает реализацию представления колонки в модели/представлении. Данный класс отображает модель в нескольких QListViews, по одному на каждую иерархию в дереве. На него иногда ссылаются как на каскадный список.

#### QDataWidgetMapper

<http://qt-project.org/doc/qt-5.1/qtwidgets/qdatawidgetmapper.html#details>

QDataWidgetMapper класс обеспечивает отображение между секцией модели данных на виджеты. Он отображает данные из модели элементов на группу виджетов. Каждый раз, когда изменяется индекс элемента, содержимое внутри виджета изменяется. *Есть пример кода использования данного класса.* Модель отображается на виджеты при помощи делегатов. Они используются для синхронизации модель и виджетов. Есть две политики сохранения изменений в модели после редактирования в виджетах: ручная и автоматическая. Есть в данном классе хорошие функции навигации по элементам. Также данный класс следит за внешними модификациями. Если модель была где-то ещё изменена, то это отразится на данном классе.

#### [QListView](http://qt-project.org/doc/qt-5.1/qtwidgets/qlistview.html)

#### [QTableView](http://qt-project.org/doc/qt-5.1/qtwidgets/qtableview.html)

#### [QTreeView](http://qt-project.org/doc/qt-5.1/qtwidgets/qtreeview.html)

#### QDesktopWidget

<http://qt-project.org/doc/qt-5.1/qtwidgets/qdesktopwidget.html#details>

QDesktopWidget класс предоставляет информацию экрана для много экранных систем. Системы с более чем одним графическим рабочим столом пользователя могут управлять физическим экранным пространством, доступным или как множественные рабочие столы или как один большой виртуальный рабочий стол. Этот класс обеспечивает информацию пользовательских рабочих столах, такую как общий размер, число экранов, геометрия каждого экрана, и настраиваются ли они как отдельные экраны или как один большой виртуальный экран. Функция [QApplication::desktop](http://qt-project.org/doc/qt-5.1/qtwidgets/qapplication.html#desktop)() используется для получения экземпляра данного класса. Также есть другие функции для получения информации о доступных экранах. *Далее представлены функции для получения информации о геометрии различных экранов.* Экран, в котором существует Главный виджет приложения, называется первичным экраном. Он сохраняет в свойстве [primaryScreen](http://qt-project.org/doc/qt-5.1/qtwidgets/qdesktopwidget.html#primaryScreen-prop). Все окна, открытые в контексте приложения следует ограничить границами первичного экрана; например, будет неудобно, если диалог будет всплывать в другом экране, или разделяться между двумя экранами. *Не понимаю тогда, зачем нужны множественные экраны. Не понимаю смысл данного абзаца.*

#### QUndoView

<http://qt-project.org/doc/qt-5.1/qtwidgets/qundoview.html#details>

QUndoView класс отображает содержание [QUndoStack](http://qt-project.org/doc/qt-5.1/qtwidgets/qundostack.html). Это [QListView](http://qt-project.org/doc/qt-5.1/qtwidgets/qlistview.html), которое отображает список команд, которые протолкнули в стек. Наиболее недавно выполненная команда всегда выделена. *Далее описываются некоторые аспекты использования данного класса.*

#### QCalendarWidget

<http://qt-project.org/doc/qt-5.1/qtwidgets/qcalendarwidget.html#details>

QCalendarWidget класс обеспечивает основанный на месяцах виджет календаря, который позволяет пользователю выбирать дату. Обычно календарь инициализируется с текущей датой, которая также изначально автоматически выбрана. Но это можно изменить при помощи специальных функций. Начертание заголовков, уикендов или одиночных дней может быть значительно настроено при помощи установки [QTextCharFormat](http://qt-project.org/doc/qt-5.1/qtgui/qtextcharformat.html)'s для некоторых специальных дней недели, специальных дат или для заголовков. Только некоторое подмножество свойств последнего класса используется в данном классе. [QMacCocoaViewContainer](http://qt-project.org/doc/qt-5.1/qtwidgets/qmaccocoaviewcontainer.html), [QMacNativeWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qmacnativewidget.html) – пока что классы мне не интересны.

## АБСТРАКТНЫЕ КЛАССЫ ВИДЖЕТОВ

#### QAbstractButton

<http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractbutton.html#details>

QAbstractButton класс является абстрактным базовым классом виджетов кнопок, обеспечивающим функциональность, общую для всех кнопок. Подклассы данного класса обрабатывают действия пользователя и определяют, как кнопка рисуется. Данный класс поддерживает как кнопки нажатия, так и кнопки выбора. Ко вторым относятся [QRadioButton](http://qt-project.org/doc/qt-5.1/qtwidgets/qradiobutton.html) and [QCheckBox](http://qt-project.org/doc/qt-5.1/qtwidgets/qcheckbox.html) классы. К первому типу кнопок относятся [QPushButton](http://qt-project.org/doc/qt-5.1/qtwidgets/qpushbutton.html) and [QToolButton](http://qt-project.org/doc/qt-5.1/qtwidgets/qtoolbutton.html) классы, для которых также можно обеспечить поведение переключения. Каждая кнопка может отображать метку и иконку. Если в названии кнопки установить символ ‘&’, то можно для данной кнопки создать горячую клавишу. Если на кнопке не предполагается текст, то горячую клавишу можно установить вручную. Кнопки в диалогах с использованием соответствующих функций могут делаться кнопками по умолчанию. *Далее перечислены состояния кнопок.* В отличие от других виджетов кнопки допускают события мыши и контекстного меню, даже когда они недоступны. *Далее описываются некоторые состояния кнопки. Далее перечисляются возможные сигналы кнопки.* Для специализации кнопок следует переопределить функции [paintEvent](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractbutton.html#paintEvent)(), чтобы рисовать контур кнопки и её текст или растр. Также советуется переопределить функции [sizeHint](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#sizeHint-prop)() и иногда [hitButton](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractbutton.html#hitButton)(). Для кнопок с более чем двумя состояниями вам также будет нужно переопределить функции [checkStateSet](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractbutton.html#checkStateSet)() and [nextCheckState](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractbutton.html#nextCheckState)().

#### QAbstractScrollArea

<http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractscrollarea.html#details>

QAbstractScrollArea виджет обеспечивает область прокрутки с полосами прокрутки по требованию. Класс является низкоуровневой абстракцией области прокрутки. Область обеспечивает центральный виджет, называемый вьюпортом, в котором содержание области прокручивается. Можно установить политику, по которой полосы прокрутки будут или видимы или прятаться, освобождая место другим полосам. Также в данном виджете можно установить некоторые поля. *Затем излагается то, что необходимо делать при специализации данного класса.* И полосу прокрутки и вьюпорт следует обновлять всякий раз, как вьюпорт получает сообщения об изменении размеров содержания. Также вьюпорт следует обновлять, когда изменяются значения полосы прокрутки. *Далее показан маленький пример кода использования полосы прокрутки. Далее даются советы по поводу настройки поведения прокрутки.* Для удобства данный класс делает все события доступными в функции [viewportEvent](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractscrollarea.html#viewportEvent)(). События виджета затем отображаются на события вьюпорта. [QScrollArea](http://qt-project.org/doc/qt-5.1/qtwidgets/qscrollarea.html) обеспечивает Гладкую прокрутку для любого виджета. При этом виджет прокручивается пиксел за пикселем. Вам следует специализировать данный класс, только если вы желаете более специализированного поведения.

#### QAbstractSlider

<http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractslider.html#details>

QAbstractSlider класс обеспечивает целочисленное значение внутри некоторого диапазона. Он разработан как общих суперкласс для виджетов [QScrollBar](http://qt-project.org/doc/qt-5.1/qtwidgets/qscrollbar.html), [QSlider](http://qt-project.org/doc/qt-5.1/qtwidgets/qslider.html) and [QDial](http://qt-project.org/doc/qt-5.1/qtwidgets/qdial.html). *Далее приводится список Главных свойств данного класса. Также далее приводится список сигналов, которые испускает данный класс.*

#### QAbstractSpinBox

<http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractspinbox.html#details>

QAbstractSpinBox класс обеспечивает счётчик и строку редактирования для отображения значений. Он разработан как общий суперкласс для виджетов [QSpinBox](http://qt-project.org/doc/qt-5.1/qtwidgets/qspinbox.html), [QDoubleSpinBox](http://qt-project.org/doc/qt-5.1/qtwidgets/qdoublespinbox.html) and [QDateTimeEdit](http://qt-project.org/doc/qt-5.1/qtwidgets/qdatetimeedit.html). *Далее приводятся Главные свойства счётчика. Далее перечисляются виртуальные функции, которые можно переопределить.*

## КЛАССЫ МАКЕТОВ ВИДЖЕТОВ

#### QFrame

<http://qt-project.org/doc/qt-5.1/qtwidgets/qframe.html#details>

QFrame класс является базовым классом для виджетов, которые могут иметь фрейм. *Теперь я понял, как работает фрейм. Он размещает виджеты в некоторых областях экрана и отображает их в некотором стиле.* Стиль фрейма определяется при помощи свойств [frame shape](http://qt-project.org/doc/qt-5.1/qtwidgets/qframe.html#Shape-enum) and a [shadow style](http://qt-project.org/doc/qt-5.1/qtwidgets/qframe.html#Shadow-enum). *Далее они перечислены.* Есть атрибуты, которые описывают толщину границы. *Далее они описываются.* Также можно устанавливать во фреймы поля. *Далее приводится хорошая иллюстрация, которая демонстрирует различные стили фрейма. Пока не очень понятно, как конкретно использовать данный класс.*

#### QButtonGroup

[*http://qt-project.org/doc/qt-5.1/qtwidgets/qbuttongroup.html#details*](http://qt-project.org/doc/qt-5.1/qtwidgets/qbuttongroup.html#details)

QButtonGroup класс обеспечивает контейнер для организации кнопок. Класс обеспечивает абстрактный контейнер, в который можно поместить кнопки. Класс не обеспечивает визуальное представление данного контейнера, но управляет состоянием каждой кнопки в группе. По умолчанию группа кнопок является взаимоисключающей. Если группа является взаимоисключающей, то при её создании должна быть выбрана какая-нибудь из кнопок. *Далее описываются некоторые важные функции данного класса.* Вдобавок данный класс может проводить отображение между целыми числами и кнопками. Кнопке можно присвоить целочисленный идентификатор. *Далее указаны некоторые функции для работы с идентификаторами.*

#### QGroupBox

[*http://qt-project.org/doc/qt-5.1/qtwidgets/qgroupbox.html#details*](http://qt-project.org/doc/qt-5.1/qtwidgets/qgroupbox.html#details)

QGroupBox виджет обеспечивает групповой ящик с названием. Групповой ящик обеспечивает фрейм, название сверху от него, горячую клавишу и отображает различные другие виджеты внутри себя. Горячая клавиша передвигает фокус ввода клавиатуры на один из виджетов в групповом ящике. При помощи свойства [flat](http://qt-project.org/doc/qt-5.1/qtwidgets/qgroupbox.html#flat-prop) можно минимизировать затраты пространства на групповой ящик при помощи удаления некоторых из границ. Групповой ящик не размещает виджеты в макет. *В данной части есть пример кода того, как можно разместить виджеты в макет.*

#### QSplitterHandler

<http://qt-project.org/doc/qt-5.1/qtwidgets/qsplitterhandle.html#details>

QSplitterHandler класс обеспечивает функциональность обработки для разделителя. Обычно данный класс не используется, а вместо него используется класс [QSplitter](http://qt-project.org/doc/qt-5.1/qtwidgets/qsplitter.html). Однако данный класс можно специализировать для обеспечения некоторой дополнительной функциональности для разделителя. Например, всплывающее меню. Однако с таким же успехом можно специализировать и класс [QSplitter](http://qt-project.org/doc/qt-5.1/qtwidgets/qsplitter.html). *Для этого продемонстрирован пример кода. Далее есть пример кода создания частного разделителя, который рисует себя по-разному в зависимости от ориентации.*

#### QSplitter

<http://qt-project.org/doc/qt-5.1/qtwidgets/qsplitter.html#details>

QSplitter класс реализует разделительный виджет. Разделитель позволяет пользователю управлять размерами дочерних виджетов при помощи перетаскивания границы между ними. Любое количество виджетов можно контролировать при помощи одного разделителя. *Есть пример кода использования разделителя.* Разделители можно размещать как горизонтально, так и вертикально. Первоначальное распределение размеров между виджетами определяется при помощи умножения начальной длины на фактор растяжения. Но можно определить размеры виджетов вручную. *Далее идёт описание некоторых аспектов использования данного класса.* Вставка в разделитель макетов не поддерживается.

#### QStackedWidget

<http://qt-project.org/doc/qt-5.1/qtwidgets/qstackedwidget.html#details>

QStackedWidget класс обеспечивает стек для виджетов, где только один виджет видим в единицу времени. Класс может быть использован для создания интерфейса пользователя, похожего на то, что обеспечивается при помощи класса [QTabWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qtabwidget.html). Это удобный виджет, построенный на вершине [QStackedLayout](http://qt-project.org/doc/qt-5.1/qtwidgets/qstackedlayout.html) класса. *Есть пример создания данного виджета.* Данный класс обеспечивает внутренние средства для пользователя для переключения между страницами. Это обычно делается через [QComboBox](http://qt-project.org/doc/qt-5.1/qtwidgets/qcombobox.html) или a [QListWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qlistwidget.html), которые сохряняют названия страниц данного виджета. *Есть пример кода использования данного класса. Далее описываются некоторые важные функции данного класса.*

#### QSizePolicy

<http://qt-project.org/doc/qt-5.1/qtwidgets/qsizepolicy.html#details>

QSizePolicy класс является атрибутом макета, описывающего горизонтальную или вертикальную политику изменения размера. Политика размера – это выражение его готовности быть изменённым в размерах различными способами, и влияет на то, как виджеты обрабатываются при помощи движка макетов. Каждый виджет возвращает экземпляр данного класса, который описывает политику изменения размеров данного виджета, которые используются при помещении виджета в макет. Помимо прочих функций в данном классе есть операции сравнения различных политик между собой.

#### QTabWidget

<http://qt-project.org/doc/qt-5.1/qtwidgets/qtabwidget.html#details>

QTabWidget класс обеспечивает стек табулированных виджетов. Класс обеспечивает полосу табуляции и область страницы, которая используется для отображения связанных с каждой табуляцией. Каждая табуляция связана с разным виджетом. Только текущая страница показывается в текущей табуляции, а все остальные страницы спрятаны. Также каждую страницу можно показывать при помощи нажатия клавиши Alt + горячая клавиша. *Далее рассказывается обычный способ использования данного типа виджета. Далее описываются некоторые функции данного виджета.* Виджеты табуляции могут быть хорошим способом разделения сложных диалогов. Альтернативой является использование [QStackedWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qstackedwidget.html). Большинство функциональности данного класса обеспечивается при помощи класса [QTabBar](http://qt-project.org/doc/qt-5.1/qtwidgets/qtabbar.html) и [QStackedWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qstackedwidget.html).

## ДРУГИЕ КЛАССЫ ВИДЖЕТОВ

#### QTextEdit::ExtraSelection

<http://qt-project.org/doc/qt-5.1/qtwidgets/qtextedit-extraselection.html#details>

QTextEdit::ExtraSelection структура обеспечивает способ определения символьного формата для данного выделения в документе.

#### QCompleter

<http://qt-project.org/doc/qt-5.1/qtwidgets/qcompleter.html#details>

QCompleter класс обеспечивает завершения, основанные на модели элементов. Его можно использовать для обеспечения автозавершений в виджетах, таких как [QLineEdit](http://qt-project.org/doc/qt-5.1/qtwidgets/qlineedit.html) and [QComboBox](http://qt-project.org/doc/qt-5.1/qtwidgets/qcombobox.html). Данный класс предлагает возможные завершения, когда пользователь начинает ввод слов, основанный на модели списка слов, которая обеспечивается [QAbstractItemModel](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html). *Далее приведён пример использования данного класса. Приведённые примеры очень ценны с точки зрения практического применения данного класса. Далее описываются некоторые аспекты использования данного класса.* Также для завершений можно использовать модели деревьев.

#### QErrorMessage

<http://qt-project.org/doc/qt-5.1/qtwidgets/qerrormessage.html#details>

QErrorMessage класс обеспечивает отображение диалога с сообщением об ошибке. *Далее идёт некоторое описание данного класса, которое я не очень понял. После это есть ссылка на пример, в котором используется данный класс.*

#### QFileIconProvider

<http://qt-project.org/doc/qt-5.1/qtwidgets/qfileiconprovider.html#details>

QFileIconProvider класс обеспечивает иконки файлов для устаревшего класса [QDirModel](http://qt-project.org/doc/qt-5.1/qtwidgets/qdirmodel.html) и для классов [QFileSystemModel](http://qt-project.org/doc/qt-5.1/qtwidgets/qfilesystemmodel.html). *Также я пропустил классы графических эффектов, которые затем следует изучить.*

#### QPlainTextEdit

<http://qt-project.org/doc/qt-5.1/qtwidgets/qplaintextedit.html#details>

QPlainTextEdit класс обеспечивает виджет, который используется для редактирования и отображения простого текста. Данный виджет является продвинутым редактором/обзором, поддерживающим простой текст. Он оптимизирован для обработки больших документов и быстро откликается на ввод пользователя. Он использует те же понятия и технологии, что и [QTextEdit](http://qt-project.org/doc/qt-5.1/qtwidgets/qtextedit.html), но оптимизирован для обработки простого текста. Данный текст работает с параграфами и символами. Документ состоит из параграфов, которые отделены линиями разрыва. *Далее рассматривается использование данного класса в качестве редактора, только для чтения, а также его отличие от класса* [QTextEdit](http://qt-project.org/doc/qt-5.1/qtwidgets/qtextedit.html).

#### QRubberBand

<http://qt-project.org/doc/qt-5.1/qtwidgets/qrubberband.html#details>

QRubberBand класс обеспечивает прямоугольник или линию, которые могут указывать на выделение или границу. Он часто используется для показа новой ограничивающей области. *Есть пример использования данного класса. Но я пока не понимаю его предназначение. Следует изучить данный класс экспериментально.*

#### QScroller

<http://qt-project.org/doc/qt-5.1/qtwidgets/qscroller.html#details>

QScroller класс предоставляет возможность кинетической прокрутки для любых виджетов прокрутки или графических элементов. Данный класс представляет объект, который сохраняет текущее положение и скорость прокрутки, а также следит за их обновлением. В документации рекомендуется оставлять все свойства в классе [QScrollerProperties](http://qt-project.org/doc/qt-5.1/qtwidgets/qscrollerproperties.html) таковыми, какими они являются по умолчанию. [QScrollerProperties](http://qt-project.org/doc/qt-5.1/qtwidgets/qscrollerproperties.html) – сохраняет свойства для класса QScroller.

#### QSpacerItem

<http://qt-project.org/doc/qt-5.1/qtwidgets/qspaceritem.html#details>

QSpacerItem класс обеспечивает пустое пространство в макете. Обычно вам нет необходимости использовать данный класс напрямую. Встроенные qt менеджеры макетов обеспечивают функции для манипулирования пустого пространства в макетах.

#### QSplashScreen

<http://qt-project.org/doc/qt-5.1/qtwidgets/qsplashscreen.html#details>

QSplashScreen виджет обеспечивает splash screen, который может быть показан во время начала приложения. Это виджет, который обычно показывается при начале приложения. Они часто используются для приложения с большим временем загрузки, чтобы обеспечить пользователя откликом о том, что приложение загружается. *Есть пример кода использования данного класса.* Так как данный объект обычно отображается перед началом цикла обработки событий, необходимо периодично вызывать функцию [QApplication::processEvents](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html#processEvents)() для получения откликов мыши, чтобы обрабатывать взаимодействие пользователя с данным экраном. Иногда полезно обновить данный экран с сообщениями, например, анонсирование установленных соединений или загруженных модулей при начале приложения. *Есть пример кода реализации данной возможности.*

#### QTileRules

<http://qt-project.org/doc/qt-5.1/qtwidgets/qtilerules.html#details>

QTileRules обеспечивает правила, используемые для рисования растров или изображений, разделённых на девять сегментов. Расщепление похоже на [CSS3 border-images](http://www.w3.org/TR/css3-background/).

#### QToolTip

<http://qt-project.org/doc/qt-5.1/qtwidgets/qtooltip.html#details>

QToolTip класс обеспечивает подсказки для любого виджета. Это маленький кусочек текста, напоминающий пользователю функцию виджета. Он изображается немедленно ниже данного положения в строгой цветовой комбинации чёрного на жёлтом. Подсказкой может быть любой форматированный текст. Наиболее простой и более общий способ установки подсказки – это вызов функции [QWidget::setToolTip](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#toolTip-prop)(). Также важно показывать разные подсказки для разных областей виджета при помощи [QHelpEvent](http://qt-project.org/doc/qt-5.1/qtgui/qhelpevent.html) типа [QEvent::ToolTip](http://qt-project.org/doc/qt-5.1/qtcore/qevent.html#Type-enum). *Есть ссылка на пример, который иллюстрирует данную технику.* В архитектуре модель/представление можно установить подсказку для элемента. *Способ этого описан в данной части. Речь идёт об использовании функции* [QAbstractItemView::viewportEvent](http://qt-project.org/doc/qt-5.1/qtwidgets/qabstractitemview.html#viewportEvent)(). Также можно настраивать текст и цвет подсказки.

#### QUndoCommand

[*http://qt-project.org/doc/qt-5.1/qtwidgets/qundocommand.html#details*](http://qt-project.org/doc/qt-5.1/qtwidgets/qundocommand.html#details)

QUndoCommand класс является базовым классом для всех команд, сохранённых в [QUndoStack](http://qt-project.org/doc/qt-5.1/qtwidgets/qundostack.html). Данный класс представляет одиночное действие редактирования над документом. *Есть описание и пример кода того, как нужно использовать данный класс.* Нужно специализировать данный класс для всех возможных действий редактирования. Также в данном классе есть некоторый текст, который представляет то, что должна делать команда. Владельцем экземпляров данного класса является стек, в который их протолкнули. Данный объект может иметь любое количество дочерних команд. Операции над родительской командой вызывают также и операции над всеми его дочерними командами. Обычно родительские команды являются пустыми. *Также в данном классе говорится нечто о создании макроса. Но я не очень понимаю, о чём конкретно идёт речь в данном случае.*

#### QUndoGroup

<http://qt-project.org/doc/qt-5.1/qtwidgets/qundogroup.html#details>

QUndoGroup класс является группой объектов [QUndoStack](http://qt-project.org/doc/qt-5.1/qtwidgets/qundostack.html). Приложение обычно имеет много стеков отката, по одному на каждый открытый документ. Но в то же время приложение обычно имеет одно действие отката, которое переключается в активном документе. Данный класс представляет из себя группу [QUndoStack](http://qt-project.org/doc/qt-5.1/qtwidgets/qundostack.html) объектов, каждый из которых может быть активен. Обязанностью программиста является определение того, какой стек является активным. Группа не является владельцем стека. Это означает, что стек следует удалять отдельно от группы. При удалении стек автоматически удаляется из группы. Стек может принадлежать только одной группе. *Рассмотрим некоторый материал, посвящённый поддержки отмены/повтора в qt.* Qt's Undo Framework является реализацией паттерна Команда для реализации функциональности отмены/повтора в приложении. *Далее описаны некоторые классы данного каркаса.* Класс [QUndoView](http://qt-project.org/doc/qt-5.1/qtwidgets/qundoview.html) показывает список команд для отмены/повтора. Нажимая на некоторую команду, можно вернуть документ к её выполнению. Поддреживает следующие понятия:

* Чистое состояние: используется для сигнализирования того, когда в документ входят и оставляют состояние, которое было сохранено на диске. Это обычно используется для разрешения или запрещения действия сохранения, а также обновления названия документа.
* Сжатие команды: используется для сжатия последовательностей команд в одну команду. *Есть некоторый пример.*
* Макрос команды: последовательность команд, все из которых отменяются или повторяются за один шаг. Это упрощает задачу написания приложения, так как набор простых команд может быть составлен в более сложные команды. *Есть также некоторый пример.*

#### QWhatsThis

<http://qt-project.org/doc/qt-5.1/qtwidgets/qwhatsthis.html#details>

QWhatsThis класс обеспечивает простое описание любого виджета, т.е. ответ на вопрос «Что это?». Данный класс снабжает пользователей информацией о функциональности и использовании данного виджета. Обычно это более длинный и информативный текст. Но он предоставляет меньше информации, чем обеспечивает отдельное окно помощи. Данный класс содержит отдельное окно, которое всплывает, когда пользователь запрашивает «Что это?». Иначе для этого можно использовать следующую процедуру: навести на нужный виджет фокус ввода и нажать клавиши Shift+F1. [QWidget::setWhatsThis](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html#whatsThis-prop)() используется для добавления данной подсказки. *Есть пример кода. Далее описываются некоторые аспекты использования данного класса.*

<http://qt-project.org/doc/qt-5.1/qtwidgets/qwidgetitem.html#details>

QWidgetItem класс является макетом, который представляет виджет. Обычно не нужно использовать данный класс напрямую. *Далее приведена таблица, которая содержит функции, которые следует использовать для манипулирования виджетов в макетах.*

#### QWizard

<http://qt-project.org/doc/qt-5.1/qtwidgets/qwizard.html#details>

QWizard класс обеспечивает для визардов. Визард – это специальный тип диалога ввода, который состоит из последовательности страниц. Его целью является продвижение пользователя через некоторый процесс шаг за шагом. Визарды полезны для сложных и нечастых задач, которые окажутся сложными для изучения пользователей. *В данной части есть пример создания простого визарда. А также есть несколько ссылок на более сложные примеры.*Данный класс поддерживает четыре стиля визарда. Также есть некоторые функции для управления видом и ощущением визарда. Визард состоит из набора страниц. Страница имеет следующие атрибуты: [title](http://qt-project.org/doc/qt-5.1/qtwidgets/qwizardpage.html#title-prop)., [subTitle](http://qt-project.org/doc/qt-5.1/qtwidgets/qwizardpage.html#subTitle-prop)., несколько растров (*перечислены*). *Есть иллюстрация касательно этих атрибутов.* В некоторых визардах ввод на одних страницах влияет на некоторые значения по умолчанию на других страницах. В данном классе введена функциональность при помощи механизма полей, которая позволяет устанавливать значение поля из любой страницы. Также можно определить обязательные поля, которые пользователь должен заполнить перед тем, как перейти на следующую страницу. *Есть пример кода того, как можно регистрировать поля. Также указано, как создать обязательное поле.* Большинство визардов имеют линейную структуру. *есть ссылка на пример создания такого визарда. Далее рассказывается, как создавать линейные визарды при помощи данного класса.* Некоторые визарды являются более сложными в том, что они позволяют разные пути, основанные на информации, обеспечиваемой при помощи пользователя. *есть ссылка на пример создания такого визарда.* В сложных визардах страницы имеют некоторые идентификаторы. Эти идентификаторы обычно определяются с использованием перечислений. *Далее показан пример кода того, как создавать некоторый простой нелинейный визард.*

#### QWizardPage

[*http://qt-project.org/doc/qt-5.1/qtwidgets/qwizardpage.html#details*](http://qt-project.org/doc/qt-5.1/qtwidgets/qwizardpage.html#details)

QWizardPage класс является базовым классом для страниц визарда. Страница обеспечивает пять виртуальных функций, которые следует переопределить для создания некоторого частного поведения. *далее описывается типичное использование данного класса. Итак, всё, с виджетами конец. Теперь приступим к изучению модуля Qt WebKit.*

# QT WEBKIT

#### [WebView](http://qt-project.org/doc/qt-5.1/qtwebkit/qml-qtwebkit3-webview.html)

<http://qt-project.org/doc/qt-5.1/qtwebkit/qtwebkit-index.html>

[WebView](http://qt-project.org/doc/qt-5.1/qtwebkit/qml-qtwebkit3-webview.html) программный интерфейс позволяет qml приложениям рисовать регионы динамического веб содержания. Данный qml компонент может делить экран вместе с другими qml компонентами или охватывать полный экран. Третья версия данного элемента несовместима с предыдущими и позволяет загружать веб страницу в данное представление как при помощи url, так и при помощи HTML строки. *Есть пример кода использования данного элемента. Теперь переходим к изучению модуля* [Qt WebKit Widgets](http://qt-project.org/doc/qt-5.1/qtwebkit/qtwebkitwidgets-index.html).

# QT WEBKIT ВИДЖЕТЫ

<http://qt-project.org/doc/qt-5.1/qtwebkit/qtwebkitwidgets-index.html>

Qt WebKit Widgets модуль обеспечивает движок веб браузера, который делает лёгким внедрение веб содержания из всемирной паутины в ваше qt приложение. В то же время веб содержание может быть enhanced с нативных элементов управления. Данный модуль представляет средства для рисования Hyper Text Markup Language (HTML), Extensible Hyper Text Markup Language (XHTML) и Scalable Vector Graphics (SVG) документов, стилизованных с использованием Cascading Style Sheets (CSS) и скриптов JavaScript. Мост между java script средой выполнения и qt объектной моделью делает возможным сгенерировать скрипт для [QObjects](http://qt-project.org/doc/qt-5.1/qtwebkit/qtwebkit-bridge.html#qobjects). Интеграция с qt сетевым модулем предоставляет возможность веб страницам быть прозрачно загруженными из веб серверов, локальных файловых систем или даже из системы ресурсов qt. В дополнение к обеспечению чисто особенностей рисования, HTML документы могут быть полностью редактируемыми для пользователя через использование атрибута редактирования содержания на HTML элементах. Данный модуль основан на Open Source WebKit движке. Больше информации о самом WebKit может быть получено по ссылке [WebKit Open Source Project](http://qt-project.org/doc/qt-5.1/qtxml/http-webkit-org.html). *Далее показано включение модуля в проект.*

Построение данного модуля вместе с отладочными символами является проблематичным на многих платформах из-за размера движка WebKit. Авторы рекомендуют построение данного модуля только в моде выпуска для встроенных платформ. *Далее есть некоторая информация о решении данной проблемы, которую я пока не могу решить.* Иконки веб сайтов в настоящий момент недоступны для Windows. Мы планируем решить эту проблему в будущих выпусках.

Наиболее лёгким способом рисования содержания является класс [QWebView](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebview.html). Он может быть внедрён в любую вашу форму или графическое представление, а также он обеспечивает удобные функции для загрузки и рисования веб сайтов. *Есть пример использования данного класса.* Экземпляр данного класса содержит один экземпляр класса [QWebPage](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebpage.html), который обеспечивает доступ к структуре документа на странице, описывая особенности, такие как фреймы, историю навигации, а также стек отмены/повтора для редактируемого содержания.

HTML документы могут быть вложенными с использованием фреймов в наборы фреймов. Отдельный фрейм в HTML представляется при помощи класса [QWebFrame](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebframe.html). Этот класс включает мост на объект окна java script и может быть нарисован с использованием [QPainter](http://qt-project.org/doc/qt-5.1/qtgui/qpainter.html). Каждый объект [QWebPage](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebpage.html) имеет один [QWebFrame](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebframe.html) объект как Главный фрейм, а уже Главный фрейм может содержать много дочерних фреймов.

Отдельные элементы HTML документа могут быть получены при помощи DOM java script интерфейсов внутри страницы. Эквивалентом данного программного интерфейса в WebKit является [QWebElement](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebelement.html). Объекты данного класса получаются с использованием [QWebFrame](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebframe.html)'s [findAllElements()](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebframe.html#findAllElements) and [findFirstElement()](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebframe.html#findFirstElement) функций при помощи CSS функций селекторов. Общие особенности веб браузера, настройки по умолчанию и другие настройки могут быть настроены через класс [QWebSettings](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebsettings.html).

Поддержка плагина Netscape доступна только для десктопных платформ. Так как данный модуль поддерживает данные плагины, приложения qt могут отображать веб страницы, которые внедряют общие плагины на платформы, для которых эти плагины доступны. Для предоставления возможности поддержки плагинов, пользователь должен иметь соответствующие двоичные файлы для этих плагинов, а также для приложения должен быть доступен атрибут [QWebSettings::PluginsEnabled](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebsettings.html#WebAttribute-enum).

*Затем приведены локации, которые просматриваются при поиске плагинов. Есть ссылка на примеры касательно данного модуля.*

## QT WEBKIT МОСТ

[*http://qt-project.org/doc/qt-5.1/qtwebkit/qtwebkit-bridge.html*](http://qt-project.org/doc/qt-5.1/qtwebkit/qtwebkit-bridge.html)

qt WebKit мост является механизмом, который расширяет WebKit java script среду для доступа к нативным объектам, представленным как [QObject](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html)s. Он имеет все преимущества самоанализа данного класса, является частью объектной модели, которая делает легче интеграцию с динамической средой java script. Есть два основных прецедента использования моста WebKitа:

* веб контент в нативных приложениях и
* тонких клиентах.

В нативных приложениях он используется в том случае, когда необходимо в некоторой его части обеспечить динамическое содержание из интернета. *В данной части приводится хороший пример касательно медиа плейера и магазина музыки в нём.*

Вторым прецедентом использования данного модуля является использование qt как нативного бэкенда для полностью веб приложений, так называемого тонкого клиента. В этом прецеденте весь интерфейс пользователя управляется HTML, JS and CSS. В дополнение к этому он использует основанные на qt компоненты для доступа к нативным особенностям, обычно не показываемым в веб, или для предоставления возможности компонентов справки, которые наилучшим образом пишутся при помощи С++. *Далее приводится пример такого рода клиента.*

Конечно, WebKit не является единственной технологией мостом. NPAPI является давним стандартом для веб-нативных мостов. Благодаря мета-объектной системе qt, полные приложения, использующие веб технологии, являются более лёгкими в разработке, нежели в NPAPI. Однако второй является лучшим для кросс-браузерных плагинов, так как он является принятым стандартом. Когда вы разрабатываете плагин для браузера, лучше использовать NPAPI. При разработке полного приложения, использующего HTML-рисование, лучше использовать qt WebKit мост.

qt WebKit является частью функциональности скриптовой поддержки qt.

По умолчанию никакие [QObjects](http://qt-project.org/doc/qt-5.1/qtwebkit/qtwebkit-bridge.html#qobjects) недоступны через веб среду по соображениям безопасности. Когда веб приложение хочет получить доступ к нативному объекту, оно должно явно получить грант на доступ к этому QObject. *Есть пример кода того, как можно добавлять объект в среду веб.*

Qt WebKit мост адаптирует центральные сигналы и слоты для генерации скрипта. Есть три принципиальных способа использования сигналов и слотов при помощи моста WebKit:

* Гибридное С++/скрипт: код С++ приложения соединяет сигнал с функцией скрипта. Этот подход полезен, если вы имеете объект, но не желаете показывать объект сам по себе скриптовой среде. Вы только хотите определить, как скрипт откликается на сигнал и оставляет его на стороне С++ кода вашего приложения, чтобы установить соединения между С++ сигналом и java script слотом.
* Гибридный скрипт/С++: скрипт может соединять сигналы и слоты, чтобы установить соединения между предопределёнными объектами, которые приложение показывает с скриптовой средой. В этом сценарии слоты сами по себе всё ещё написаны на С++, но определение соединений полностью динамическое (определённое при помощи скрипта).
* Полностью определённый с помощью скрипта: скрипт может как определить функции обработчика сигнала (слоты, написанные на java script), так и установить соединения, которые используют эти обработчики.

*Далее идёт пример соединения сигнала и функции.* Это осуществляется при помощи функции connect, которой в качестве аргумента можно передать как java script функцию, так и слот qt. *Далее показан пример соединения сигнала и функции члена. Далее показан пример соединения сигнала с именованной функций. Далее показано, как получать информацию об ошибке, если соединение или разрыв соединения не установлены.*

Если в некотором объекте слоты являются перегруженными, то получить доступ к определённому слоту можно или при помощи задания правильного аргумента, или при помощи использования специальной сигнатуры, *которая показана в данной части*. Как сигналы, так и слоты являются запрашиваемыми из скриптов по умолчанию. Также возможно определить метод, который можно запрашивать из скрипта. Это особенно ценно для функций, которые возвращают некоторые значения, так как слоты не возвращают обычно никаких значений. Для этого перед соответствующим методом следует поместить макрос Q\_INVOKABLE. *Есть пример кода.*

Свойства qt объекта доступны как свойства соответствующего java script объекта. При работе со свойством qt его методы чтения и записи будут автоматически запрашиваться. *Есть пример кода.* Каждый дочерний объект QObject по умолчанию доступен как свойство обёртки java script объекта. *есть примеры кода.* Мост делает автоматическое преобразование между java script типом и ожидаемым типом qt. Каждый qt тип имеет соответствующий набор правил того, как java script обрабатывает его.

Все численные типы данных qt преобразуются из и в java script числа. *Далее имеются некоторые правила для преобразования типа QChar.* Рекомендуется использовать стандартные типы чисел, которые легко преобразуются. Если там, где требуется число, задаётся не число, то вызывается соответствующая функция (parseInt / parseFloat). *Далее описаны правила преобразования QString. Далее описываются правила преобразования типов времени и даты.* Также WebKit автоматически преобразует регулярные выражения из java script в qt. *Далее перечислены некоторые типы списков, которые обрабатываются мостом специальным образом.* При этом происходит преобразование java script массива в соответствующий список.

Составные java script объекты (compound objects – JSON) – это переменные, которые содержат список пар ключ-значение, где все ключи – это строки, а значение может иметь любой тип. Данный тип очень хорошо преобразуется в [QVariantMap](http://qt-project.org/doc/qt-5.1/qtcore/qvariant.html#QVariantMap-typedef). Данное преобразование позволяет очень удобный способ передачи любых структур данных между С++ и java script. Заметьте, что типы, которые не поддерживаются java script, такие как java script функции или геттеры и сеттеры, не преобразуются. Когда слот или свойство требуют [QVariant](http://qt-project.org/doc/qt-5.1/qtcore/qvariant.html), то WebKit мост создаёт такой [QVariant](http://qt-project.org/doc/qt-5.1/qtcore/qvariant.html), который наилучшим образом совпадает с аргументом, переданным при помощи java script. Использование типа [QVariant](http://qt-project.org/doc/qt-5.1/qtcore/qvariant.html) в общем в С++ делает С++ программирование похожим на java script программирование, так как это придаёт языку дополнительный уровень перенаправления. Но при этом использование данного подхода уменьшает безопасность типов и надёжность С++. Авторы рекомендуют использовать варианты только для высокоуровневых функций, и держать большинство ваших QObject безопасными с точки зрения типа.

Мост WebKit обрабатывает [QPixmap](http://qt-project.org/doc/qt-5.1/qtgui/qpixmap.html)s and [QImage](http://qt-project.org/doc/qt-5.1/qtgui/qimage.html)s специальным образом. Так как WebKit сохраняет растры для представления HTML изображений, поэтому данные объекты могут быть напрямую использованы прямо в WebKit. *Далее показан java script объект, в который на самом деле преобразуются данные объекты. Отмечается также, что функция* toDataURL() является дорогостоящей, и её следует использовать с осторожностью. *Есть хороший пример использования данного класса.*

Сигнал, слот или свойство, которое ожидает или возвращает [QWebElement](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebelement.html), может работать бесшовно с java script ссылками на DOM объекты.

При работе с WebKit гибридными особенностями, часто можно попасть в ловушку, когда делаешь программный интерфейс, видимый для java script, очень богатым и используешь его все особенности. Это, однако, приводит к сложности и может создать баги, которые трудно найти. Вместо этого советуется держать гибридные слои маленькими и управляемыми: создавать шлюз, только когда есть в этом действительная необходимость, т. е. существует некоторый новый нативный enabler, который требует прямого интерфейса в слой приложения. Иногда новая функциональность лучше обрабатывается внутренне в нативных слоях или в веб слоях; простота – это ваш друг. Это становится более очевидно, когда гибридный слой может создавать или уничтожать объекты, или использует сигналы, слоты или свойства с аргументом [QObject](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html)\*. Советуется быть очень внимательным и обрабатывать показываемые [QObject](http://qt-project.org/doc/qt-5.1/qtcore/qobject.html) как системные – с очень большим вниманием к управлению памяти и владению объектами.

Когда вы показываете нативные объекты в открытую веб среду, то важно понимать предположения безопасности. Подумайте о том, показывает ли enabler объекта веб окружению доступ к вещам, которые не следует открывать, и приходит ли из источника с доверием загружаемое веб содержание. В общем, когда показываются нативные QObjects, которые дают доступ к приватной информации или к функциональности, которая потенциально вредна для клиента, такой показ следует сбалансировать при помощи ограничения доступа к веб страницам либо с проверенными url только при помощи HTTPS, или при помощи использования других мер, как части стратегии безопасности.

#### QWebView

<http://qt-project.org/doc/qt-5.1/qtwebkit/qwebview.html#details>

QWebView класс обеспечивает виджет, который используется для отображения и редактирования веб документов. Это Главный виджет компонент qt WebKit модуля просмотра. Он может быть использован в различных приложениях для отображения веб содержания из интернета. [load](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebview.html#load)() используется для загрузки веб сайта. *Есть пример кода использования данного класса.* Также можно использовать функции [setUrl](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebview.html#url-prop)() и [setHtml](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebview.html#setHtml)(). *есть различные сигналы, сигнализирующие о процессе загрузки.* [page](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebview.html#page)() функция возвращает указатель на объект веб страницы. Для изменения установок веб представления вы можете получить доступ к объекту [QWebSettings](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebsettings.html) при помощи функции [settings](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebview.html#settings)(). При помощи этого класса можно, например, изменить шрифты по умолчанию или отменить такие особенности как java script или плагины. Название html документа может быть получено при помощи свойства [title](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebview.html#title-prop)(). Также можно использовать иконку, изменять размер текста. Можно делать частное контекстное меню. *Здесь есть описание этой функциональности.* Данный класс может быть напечатан при помощи [QPrinter](http://qt-project.org/doc/qt-5.1/qtprintsupport/qprinter.html).

Если вы желаете обеспечить поддержку для веб сайтов, которые позволяют пользователю открывать новые окна, вам следует специализировать данный класс и переопределить функцию [createWindow](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebview.html#createWindow)(). QWebView состоит из других объектов, таких как [QWebFrame](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebframe.html) and [QWebPage](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebpage.html). *Есть схема, показывающая структуру этих элементов.* Веб представление состоит из одной веб страницы, которая состоит из многих веб фреймов.

Возможно использовать [QWebPage](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebpage.html) and [QWebFrame](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebframe.html) без использования QWebView, если вам не нужны атрибуты [QWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qwidget.html). Однако всё равно данный модуль зависит от [QtGui](http://qt-project.org/doc/qt-5.1/qtgui/qtgui-module.html), так что следует использовать [QApplication](http://qt-project.org/doc/qt-5.1/qtwidgets/qapplication.html) вместо [QCoreApplication](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html).

#### QWebPage

<http://qt-project.org/doc/qt-5.1/qtwebkit/qwebpage.html#details>

QWebPage класс обеспечивает объект, чтобы отображать и редактировать веб документы. Он удерживает Главный фрейм, ответственный за веб содержание, настройки, историю ссылок навигации и действия. API данного класса очень похоже на API класса [QWebView](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebview.html). Перед тем, как рисовать объект данного класса, вам необходимо установить размер вьюпорта при помощи вызова функции [setViewportSize](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebpage.html#viewportSize-prop)(). Затем, вы запрашиваете функцию рисования Главного фрейма. *Есть пример кода. Я его не очень хорошо воспринял.*

#### QWebFrame

<http://qt-project.org/doc/qt-5.1/qtwebkit/qwebframe.html#details>

класс QWebFrame представляет фрейм в веб странице. Каждый объект [QWebPage](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebpage.html) содержит по крайней мере один фрейм, Главный фрейм, получаемый при помощи функции [QWebPage::mainFrame](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebpage.html#mainFrame)(). Дополнительные фреймы могут быть созданы для элементов html <frame> или <iframe>.

*Далее описывается некоторая функциональность данного класса.* Объекты данного класса создаются и управляются при помощи веб страницы. Есть несколько способов программно проверить содержание фрейма. [hitTestContent](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebframe.html#hitTestContent)() можно использовать для нахождения элементов по координате. Для доступа к нижележащему дереву DOM можно использовать функции [documentElement](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebframe.html#documentElement)(), [findAllElements](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebframe.html#findAllElements)() and [findFirstElement](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebframe.html#findFirstElement)(). Также данный класс может быть напечатан при помощи [QPrinter](http://qt-project.org/doc/qt-5.1/qtprintsupport/qprinter.html).

#### QWebElement

<http://qt-project.org/doc/qt-5.1/qtwebkit/qwebelement.html#details>

QWebElement класс обеспечивает удобный доступ к DOM элементам в [QWebFrame](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebframe.html). Данный объект позволяет легко получить доступ к модели документа, представляемой древоподобной структурой DOM элементов. Корень дерева называется элементом документа и может быть получен с использованием [QWebFrame::documentElement](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebframe.html#documentElement)(). Определённые элементы идентифицируются с использованием CSS селекторов. *Есть пример кода. Также демонстрируется другой способ ручного просмотра документа. есть пример кода. М*ожно изменять отдельные элементы.

Также можно эмулировать событие нажатия на элемент. *Есть пример кода.* Нижележащее содержание данного класса явно разделяется. Создание копии данного объекта не создаёт копию содержания. Вместо этого оба экземпляра указывают на один и тот же элемент. *Есть ссылки на примеры.*

#### QWebSettings

[*http://qt-project.org/doc/qt-5.1/qtwebkit/qwebsettings.html#details*](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebsettings.html#details)

QWebSettings класс обеспечивает объект, чтобы сохранять настройки, используемые [QWebPage](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebpage.html) and [QWebFrame](http://qt-project.org/doc/qt-5.1/qtwebkit/qwebframe.html). Каждый объект страницы имеет свой собственный объект настроек. Если настройки не сконфигурированы, то просматривается Глобальный объект настроек. Данный класс позволяет настраивать свойства браузера. Также данный класс может настраивать Глобальные свойства. *Они перечислены в данной части.* Также можно осуществлять поддержку плагинов браузера. *Есть некоторое описание касательно данного вопроса.* Данный класс поддерживает особенности [HTML 5](http://qt-project.org/doc/qt-5.1/qtxml/http-www-w3-org-tr-html5.html), которые улучшают производительность и возможности веб приложений. *Они перечислены и описаны. Далее следует изучить модуль, посвящённый sql.*

# QT SQL

<http://qt-project.org/doc/qt-5.1/qtsql/qtsql-index.html>

qt SQL является ключевым модулем, который обеспечивает поддержку для баз данных SQL. Он разделяется на несколько слоёв:

* Слой драйверов
* Слой SQL программного интерфейса
* Слой интерфейса пользователя

## SQL ПРОРАММИРОВАНИЕ

<http://qt-project.org/doc/qt-5.1/qtsql/sql-programming.html>

*Есть ссылка на книгу о базах данных*. *Далее перечислены темы. Далее перечислены классы данного модуля.* Слой драйверов: [QSqlDriver](http://qt-project.org/doc/qt-5.1/qtsql/qsqldriver.html), [QSqlDriverCreator](http://qt-project.org/doc/qt-5.1/qtsql/qsqldrivercreator.html), [QSqlDriverCreatorBase](http://qt-project.org/doc/qt-5.1/qtsql/qsqldrivercreatorbase.html), [QSqlDriverPlugin](http://qt-project.org/doc/qt-5.1/qtsql/qsqldriverplugin.html), and [QSqlResult](http://qt-project.org/doc/qt-5.1/qtsql/qsqlresult.html). Слой программного интерфейса SQL: [QSqlDatabase](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html), [QSqlQuery](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquery.html), [QSqlError](http://qt-project.org/doc/qt-5.1/qtsql/qsqlerror.html), [QSqlField](http://qt-project.org/doc/qt-5.1/qtsql/qsqlfield.html), [QSqlIndex](http://qt-project.org/doc/qt-5.1/qtsql/qsqlindex.html), and [QSqlRecord](http://qt-project.org/doc/qt-5.1/qtsql/qsqlrecord.html). Слой интерфейса пользователя: [QSqlQueryModel](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquerymodel.html), [QSqlTableModel](http://qt-project.org/doc/qt-5.1/qtsql/qsqltablemodel.html), and [QSqlRelationalTableModel](http://qt-project.org/doc/qt-5.1/qtsql/qsqlrelationaltablemodel.html). Перед использованием данных классов должен быть установлен объект класса [QCoreApplication](http://qt-project.org/doc/qt-5.1/qtcore/qcoreapplication.html).

## СОЕДИНЕНИЕ С БАЗАМИ ДАННЫХ

<http://qt-project.org/doc/qt-5.1/qtsql/sql-connecting.html>

Для доступа к базе данных при помощи [QSqlQuery](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquery.html) или [QSqlQueryModel](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquerymodel.html), вы должны создать и открыть одно или более соединений базы данных. Они обычно идентифицируются по имени соединения, но не по имени базы данных. Можно иметь множественные соединения для одной и той же базы данных. Также класс [QSqlDatabase](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html) поддерживает понятие соединения по умолчанию, которое является неименованным соединением. Если функциям классов [QSqlQuery](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquery.html) или [QSqlQueryModel](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquerymodel.html) не передавать соединение в качестве аргумента, то используется соединение по умолчанию. Есть разница между созданием соединения и его открытием. Создание осуществляется при создании экземпляра класса [QSqlDatabase](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html). Соединение нельзя использовать, пока оно не будет открыто. *Далее приведён пример кода создания соединения по умолчанию и его открытия. Затем показано, как создавать соединения, которые не являются соединениями по умолчанию.* В случае ошибки соединения можно вызвать функцию [QSqlDatabase::lastError](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html#lastError)() для получения информации об ошибке соединения. Как только соединение установлено, мы можем вызвать статическую функцию [QSqlDatabase::database](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html#database)() из любого места с именем соединения, чтобы получить указатель на это соединение базы данных. Не передавая имени соединения, будет возвращено соединение по умолчанию. *Есть примеры кода вызова соединений. Это работает как небезызвестный пул соединений.* Для удаления соединения с базой данных можно вначале закрыть базу данных с использованием функции [QSqlDatabase::close](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html#close)(), а затем удалить его с использованием статического метода [QSqlDatabase::removeDatabase](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html#removeDatabase)().

## SQL ДРАЙВЕРЫ БАЗ ДАННЫХ

<http://qt-project.org/doc/qt-5.1/qtsql/sql-driver.html>

Модуль SQL использует плагины драйверов, чтобы осуществлять коммуникацию с разными программными интерфейсами базы данных. Программный же интерфейс модуля является независимым от базы данных, а весь код базы данных содержится в данных драйверах. Qt обеспечивает несколько драйверов, а также несколько можно добавить. Исходный код драйверов открыт и может быть использован как источник для написания собственных драйверов. *Далее перечислены доступные драйверы в qt. Не все они доступны бесплатно.* Чтобы построить плагин драйвера, вам нужно иметь подходящую клиентскую базу данных для вашей системы управления базой данных (Database Management System – DBMS). *Далее показано, как строить плагин драйвера на вашей системе.*

ODBC – это общий интерфейс, который позволяет вам использовать одинаковый интерфейс для многих баз данных. Драйвер для постгрес автоматически детектирует, поддерживает ли ваша база данных кодировку юникод или нет. Заметьте, что сам драйвер поддерживает только UTF-8.

В постгрес большие сериализованные объекты доступны через тип BYTEA. *В данной части для всех драйверов показана процедура создания или построения драйверов.*

Вы должны всегда использовать клиентские библиотеки, которые были скомпилированы при помощи того же компилятора, который вы используете для вашего проекта.

*Далее перечислено то, что необходимо проверить, если плагин построен успешно, а всё равно он не загружается.*

[QSqlDatabase](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html) отвечает за загрузку и управление плагинами драйверов базы данных. Когда база данных добавляется (see [QSqlDatabase::addDatabase](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html#addDatabase)()), соответствующий плагин драйвера загружается (using [QSqlDriverPlugin](http://qt-project.org/doc/qt-5.1/qtsql/qsqldriverplugin.html)). [QSqlDatabase](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html) полагается на плагин драйвера, чтобы обеспечить интерфейсы для [QSqlDriver](http://qt-project.org/doc/qt-5.1/qtsql/qsqldriver.html) and [QSqlResult](http://qt-project.org/doc/qt-5.1/qtsql/qsqlresult.html). [QSqlDriver](http://qt-project.org/doc/qt-5.1/qtsql/qsqldriver.html) - это абстрактный базовый класс, который определяет функциональность SQL драйвера базы данных. Он отвечает за соединение с базой данных, установку соответствующей среды и т.д. он также может использовать объекты [QSqlQuery](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquery.html), подходящие для данного программного интерфейса базы данных. [QSqlDatabase](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html) направляет многие из вызовов функций классу [QSqlDriver](http://qt-project.org/doc/qt-5.1/qtsql/qsqldriver.html), который обеспечивает конкретную реализацию. [QSqlResult](http://qt-project.org/doc/qt-5.1/qtsql/qsqlresult.html) – это абстрактный базовый класс, который определяет функциональность SQL запроса к базе данных. Он отвечает за посылание запросов в базу данных, возвращение результатов и т. д. [QSqlQuery](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquery.html) направляет многие из своих функций напрямую к [QSqlResult](http://qt-project.org/doc/qt-5.1/qtsql/qsqlresult.html), который обеспечивает конкретную реализацию. [QSqlDriver](http://qt-project.org/doc/qt-5.1/qtsql/qsqldriver.html) and [QSqlResult](http://qt-project.org/doc/qt-5.1/qtsql/qsqlresult.html) тесно связаны. При реализации частного SQL драйвера оба этих класса нужно специализировать, а чисто виртуальные методы в каждом классе следует переопределить.

Чтобы определить SQL драйвер как плагин, он должен использовать макрос [Q\_PLUGIN\_METADATA](http://qt-project.org/doc/qt-5.1/qtcore/qtplugin.html#Q_PLUGIN_METADATA)(). *Далее указана директория, в которой можно смотреть исходники плагинов. А также есть пример кода скелета некоторого драйвера.*

## ВЫПОЛНЕНИЕ SQL УТВЕРЖДЕНИЙ

<http://qt-project.org/doc/qt-5.1/qtsql/sql-sqlstatements.html>

[QSqlQuery](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquery.html) класс обеспечивает интерфейс для выполнения SQL утверждений и навигации через результирующий набор запроса. [QSqlQueryModel](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquerymodel.html) and [QSqlTableModel](http://qt-project.org/doc/qt-5.1/qtsql/qsqltablemodel.html) классы обеспечивают более высокоуровневый интерфейс для доступа к базе данных. *Показано, как выполнить SQL утверждение.*

[QSqlQuery](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquery.html) обеспечивает доступ к результирующему набору по одной записи за раз. *Далее описаны многочисленные функции класса* [QSqlQuery](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquery.html#size).*Далее приведены примеры выполнения других SQL утверждений.*

При вставке многих записей, вам просто нужно вызвать однажды [QSqlQuery::prepare](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquery.html#prepare)(). Затем вы вызываете [bindValue()](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquery.html#bindValue) или [addBindValue()](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquery.html#addBindValue), за которыми следует вызов функции [exec()](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquery.html#exec) столько много раз, сколько необходимо. Если нижележащий движок базы данных поддерживает транзакции, то QSqlDriver::hasFeature([QSqlDriver::Transactions](http://qt-project.org/doc/qt-5.1/qtsql/qsqldriver.html#DriverFeature-enum)) возвратит правду. Вы можете использовать [QSqlDatabase::transaction](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html#transaction)(), чтобы инициировать транзакцию, за которой следуют SQL команды, которые вы желаете выполнить в контексте транзакции, а затем вызвать или [QSqlDatabase::commit](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html#commit)() или [QSqlDatabase::rollback](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html#rollback)(). При использовании транзакции вы должны начинать транзакцию перед использованием запроса. *Есть пример кода.* Транзакции могут быть использованы, чтобы гарантировать, что сложная операция является атомистической или обеспечить средства для отмены сложных изменений в середине.

## ТИПЫ ДАННЫХ ДЛЯ ПОДДЕРЖИВАЕМЫХ QT СИСТЕМ БАЗ ДАННЫХ

<http://qt-project.org/doc/qt-5.1/qtsql/sql-types.html>

*есть таблица, которая показывает рекомендуемые типы данных для баз данных, которые поддерживает qt. Очень полезная часть документации.*

## ИСПОЛЬЗОВАНИЕ КЛАССОВ SQL МОДЕЛИ

<http://qt-project.org/doc/qt-5.1/qtsql/sql-model.html>

В дополнение к классу [QSqlQuery](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquery.html), qt предлагает три высокоуровневых класса для доступа к базам данных. Это [QSqlQueryModel](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquerymodel.html) (модель только для чтения, основанная на произвольном запросе SQL), [QSqlTableModel](http://qt-project.org/doc/qt-5.1/qtsql/qsqltablemodel.html) (модель чтения/записи, которая работает для одной таблицы), and [QSqlRelationalTableModel](http://qt-project.org/doc/qt-5.1/qtsql/qsqlrelationaltablemodel.html) (подкласс второго класса с поддержкой внешних ключей). Все эти классы происходят от [QAbstractTableModel](http://qt-project.org/doc/qt-5.1/qtcore/qabstracttablemodel.html), поэтому легко представлять данные из базы данных в классах представления модели [QListView](http://qt-project.org/doc/qt-5.1/qtwidgets/qlistview.html) and [QTableView](http://qt-project.org/doc/qt-5.1/qtwidgets/qtableview.html). Вторым преимуществом использования данных классов является то, что можно сделать более лёгким код, чтобы адаптировать один источник данных к другому. Например, если вы используете [QSqlTableModel](http://qt-project.org/doc/qt-5.1/qtsql/qsqltablemodel.html) и затем решаете использовать XML файлы, чтобы сохранить данные вместо базы данных, то, по существу, это лишь вопрос замены одной модели данных другой. [QSqlTableModel](http://qt-project.org/doc/qt-5.1/qtsql/qsqltablemodel.html) – это высокоуровневая альтернатива для класса [QSqlQuery](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquery.html) для навигации и модификации одиночных таблиц SQL. *Есть различные примеры кода по использованию данного класса.*

Когда вы заканчиваете изменение записи, то всегда вызывайте функцию [QSqlTableModel::submitAll](http://qt-project.org/doc/qt-5.1/qtsql/qsqltablemodel.html#submitAll)(), чтобы гарантировать, что изменения будут записаны в базу данных. *Есть некоторые замечания по поводу опасности фиксации при изменении каждой записи. Это безопасность и опасность изменения ключей.*

[QSqlRelationalTableModel](http://qt-project.org/doc/qt-5.1/qtsql/qsqlrelationaltablemodel.html) расширяет [QSqlTableModel](http://qt-project.org/doc/qt-5.1/qtsql/qsqltablemodel.html), чтобы обеспечить поддержку внешних ключей. Внешний ключ – это отображение между полем из одной таблицы и первичным ключом из другой таблицы. *Есть пример кода использования данной таблицы.*

## ПРЕДСТАВЛЕНИЕ ДАННЫХ В ТАБЛИЧНОМ ПРЕДСТАВЛЕНИИ

<http://qt-project.org/doc/qt-5.1/qtsql/sql-presenting.html>

[QTableView](http://qt-project.org/doc/qt-5.1/qtwidgets/qtableview.html) на практике на много более часто используется для отображения модели таблицы базы данных, так как результирующий набор SQL является обычно двумерной таблицей. *Есть пример создания представления, основанного на SQL модели. Также показано, как запретить возможность редактирования.* Одну и ту же модель можно использовать как источник данных для многих представлений. Если редактирование происходит в одном из представлений, то изменения немедленно отражаются в остальных.

Звёздочкой обозначается вставленная пользователем запись. *Подробнее об этом можно посмотреть в данной части*. Класс [QSqlTableModel](http://qt-project.org/doc/qt-5.1/qtsql/qsqltablemodel.html) оптимизирован для работы с одной таблицей. Если вы желаете модель чтения/записи, которая управляет произвольным набором данных, вы можете специализировать класс [QSqlQueryModel](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquerymodel.html) и переопределить функции [flags()](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html#flags) and [setData()](http://qt-project.org/doc/qt-5.1/qtcore/qabstractitemmodel.html#setData), чтобы сделать эту модель моделью для чтения/записи. *Есть пример кода установки данной возможности. Есть также ссылка на пример, в котором показана данная функциональность.* Специализация модели позволяет настраивать её многими способами. Вы можете обеспечить подсказки для элементов, обеспечить рассчитываемые значения, обеспечивать разные значения для чтения и редактирования. [QSqlRelationalDelegate](http://qt-project.org/doc/qt-5.1/qtsql/qsqlrelationaldelegate.html) – это делегат, который позволяет обрабатывать внешние ключи в таблице представления с использованием выпадающего списка. *Есть пример кода использования данных классов.*

## СОЗДАНИЕ ФОРМ, ОСВЕДОМЛЁННЫХ О ДАННЫХ

<http://qt-project.org/doc/qt-5.1/qtsql/sql-forms.html>

Содержание базы данных может быть представлено различными способами. Очень часто пользователям удобно работать для редактирования таблицы базы данных с набором виджетов для каждого поля. Такие формы можно создавать при помощи класса [QDataWidgetMapper](http://qt-project.org/doc/qt-5.1/qtwidgets/qdatawidgetmapper.html). Данный класс управляет одной таблицей базы данных, отображая её записи на набор виджетов. *Есть указание на пример, в котором используется данный класс. Теперь рассмотрим классы для данного модуля.*

## КЛАССЫ SQL

<http://qt-project.org/doc/qt-5.1/qtsql/qtsql-module.html>

#### QSql

<http://qt-project.org/doc/qt-5.1/qtsql/qsql.html#details>

QSql пространство имён содержит разнообразные идентификаторы, используемые повсеместно в qt SQL модуле.

#### QSqlDatabase

<http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html#details>

QSqlDatabase класс представляет соединение с базой данных. Он предоставляет интерфейс для доступа к базе данных через соединение. Экземпляр данного класса представляет соединение. Соединение предоставляет доступ к базе данных через один из поддерживаемых драйверов, которые происходят от [QSqlDriver](http://qt-project.org/doc/qt-5.1/qtsql/qsqldriver.html). Можно создавать собственные драйверы. [addDatabase](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html#addDatabase)() статические функции создают соединение, в которых вы задаёте тип драйвера или сам драйвер, а также имя соединения. *Есть пример кода создания соединения.* QSqlDatabase – это класс значение. Изменения, сделанные в базе данных в одном соединении скажутся на остальных экземплярах данного класса, которые представляют то же соединение. [cloneDatabase](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html#cloneDatabase)() используется для создания независимого соединения базы данных, которое основано на существующем. Если же вы используете множественные соединения с базой данных, то определите для них уникальные имена. [contains](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html#contains)() используется для проверки того, содержится ли соединение с таким именем. [tables](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html#tables)() показывает список таблиц базы данных после установления соединения. [primaryIndex](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html#primaryIndex)() – получение первичного индекса таблицы. [record](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html#record)() – получение мета-информации о полях таблицы. [QSqlDatabase::exec](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html#exec)() устарела. [QSqlQuery::exec](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquery.html#exec)() используется вместо неё. Есть поддержка транзакций. Есть функция для уточнения ошибок. Можно получить имена доступных SQL драйверов при помощи функции [drivers](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html#drivers)(). А также проверить наличие некоторого конкретного драйвера при помощи [isDriverAvailable](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html#isDriverAvailable)(). Частные драйверы нужно регистрировать при помощи [registerSqlDriver](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html#registerSqlDriver)().

#### QSqlDriver

<http://qt-project.org/doc/qt-5.1/qtsql/qsqldriver.html#details>

QSqlDriver класс является абстрактным базовым классом для доступа к определённым SQL базам данных. Данный класс не следует использовать напрямую, а вместо этого следует использовать [QSqlDatabase](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html). Если же вы желаете создавать свои собственные SQL драйверы, вы можете специализировать данный класс и переопределить его чисто виртуальные функции, а также те функции, которые вам необходимы.

#### QSqlDriverCreator

<http://qt-project.org/doc/qt-5.1/qtsql/qsqldrivercreator.html#details>

QSqlDriverCreator класс является шаблонным классом, который обеспечивает фабрику SQL драйверов для определённого типа драйвера. *Пока что очевидно не понял предназначение данной фабрики для пользователя библиотеки.*

#### QSqlDriverCreatorBase

[*http://qt-project.org/doc/qt-5.1/qtsql/qsqldrivercreatorbase.html#details*](http://qt-project.org/doc/qt-5.1/qtsql/qsqldrivercreatorbase.html#details)

QSqlDriverCreatorBase класс является базовым классом для фабрик SQL драйверов.

#### QSqlDriverPlugin

<http://qt-project.org/doc/qt-5.1/qtsql/qsqldriverplugin.html#details>

QSqlDriverPlugin класс обеспечивает абстрактный базовый класс для частных плагинов [QSqlDriver](http://qt-project.org/doc/qt-5.1/qtsql/qsqldriver.html). Это простой интерфейс, который позволяет создавать собственные плагины SQL драйверов, которые могут быть загружены динамически qt. *Далее немного описано, как создавать собственный плагин.*

#### QSqlError

<http://qt-project.org/doc/qt-5.1/qtsql/qsqlerror.html#details>

QSqlError класс обеспечивает информацию об ошибке SQL базы данных.

#### QSqlField

<http://qt-project.org/doc/qt-5.1/qtsql/qsqlfield.html#details>

QSqlField класс манипулирует полями в таблицах SQL базы данных и представлении. Предоставляет характеристики одной колонки таблицы базы данных или представления (тип данных и название). Также класс содержит значение в данном поле, которое можно изменить. Данные сохраняются как QVariant. Данные объекты обычно не создаются напрямую в коде, но они содержаться в функциях класса [QSqlRecor](http://qt-project.org/doc/qt-5.1/qtsql/qsqlrecord.html)d, который обычно используется для доступа к записям. *Есть примеры коды.* Данный класс может обеспечить некоторую мета-информацию о поле, например, его имя, тип, длину, точность, значение по умолчанию, идентификатор типа и другую информацию.

#### QSqlIndex

<http://qt-project.org/doc/qt-5.1/qtsql/qsqlindex.html#details>

QSqlIndex класс обеспечивает функции для манипулирования и описания индексов базы данных. Индекс указывает на одну таблицу или на представление в базе данных. Информация о полях, которые включаю индекс, может быть использована для генерации SQL выражений.

#### QSqlQuery

<http://qt-project.org/doc/qt-5.1/qtsql/qsqlquery.html#details>

QSqlQuery класс обеспечивает средства для выполнения и манипулирования SQL утверждений. Данный класс инкапсулирует функциональность, вовлечённую в создание, навигацию и получение данных из SQL запросов, которые выполняются при помощи [QSqlDatabase](http://qt-project.org/doc/qt-5.1/qtsql/qsqldatabase.html). Он может выполнять как DML, так и DDL утверждения. *Что-то написано об активности базы данных, но я этого пока не понял. Далее перечислены функции навигации по записям. Далее показаны конкретный функции и примеры кода. особенно важен пример, который показывает, как получить доступ к отдельному полю.* Данный класс поддерживает выполнение подготовленных выполнений запроса и связывание значений параметра с указателями места. *Далее помещена некоторая информация, которая касается базы данных Oracle. Далее приводятся примеры связывания значений для запросов.*

Вы должны вначале загрузить SQL драйвер и открыть соединение прежде создания объекта данного класса. Также соединение должно оставаться открытым, пока соединение существует; иначе поведение данного класса не определено.

#### QSqlQueryModel

<http://qt-project.org/doc/qt-5.1/qtsql/qsqlquerymodel.html#details>

QSqlQueryModel класс обеспечивает модель данных только для чтения для результирующего набора записей SQL. Это высокоуровневый класс. *Далее описываются некоторые его функции.*

#### QSqlRecord

<http://qt-project.org/doc/qt-5.1/qtsql/qsqlrecord.html#details>

QSqlRecord класс инкапсулирует запись базы данных. Он инкапсулирует функциональность и характеристики записи базы данных. Он поддерживает удаление полей, также как установку и получение значений поля. Значение полей записи могут быть установлены либо при помощи имени, либо при помощи позиции. Когда запросы генерируются для выполнения, то только те поля, для которых функция [isGenerated](http://qt-project.org/doc/qt-5.1/qtsql/qsqlrecord.html#isGenerated)() возвращает истину, включаются в генерируемый SQL. *Также рассказано о других функций. Есть функция для добавления поля.*

#### QSqlRelation

<http://qt-project.org/doc/qt-5.1/qtsql/qsqlrelation.html#details>

QSqlRelation класс сохраняет информацию о SQL внешних ключах. Это вспомогательный класс для класса [QSqlRelationalTableModel](http://qt-project.org/doc/qt-5.1/qtsql/qsqlrelationaltablemodel.html).

#### QSqlRelationalDelegate

<http://qt-project.org/doc/qt-5.1/qtsql/qsqlrelationaldelegate.html#details>

QSqlRelationalDelegate класс обеспечивает делегат, который используется для отображения и редактирования данных из класса [QSqlRelationalTableModel](http://qt-project.org/doc/qt-5.1/qtsql/qsqlrelationaltablemodel.html). Это делегат в виде выпадающего списка для поиска внешних ключей.

#### QSqlRelationalTableModel

<http://qt-project.org/doc/qt-5.1/qtsql/qsqlrelationaltablemodel.html#details>

QSqlRelationalTableModel класс обеспечивает редактируемую модель данных для одной таблицы базы данных с поддержкой внешних ключей. *Далее описываются некоторые функции данного класса, а также требования по его использованию.*

#### QSqlResult

[*http://qt-project.org/doc/qt-5.1/qtsql/qsqlresult.html#details*](http://qt-project.org/doc/qt-5.1/qtsql/qsqlresult.html#details)

QSqlResult класс обеспечивает абстрактный интерфейс для доступа к данным из определённых sql баз данных. Обычно вам вместо данного класса следует использовать класс [QSqlQuery](http://qt-project.org/doc/qt-5.1/qtsql/qsqlquery.html), который является обёрткой для специфических для конкретной базы данных реализаций данного класса. Класс нужен при реализации собственных драйверов баз данных.

#### QSqlTableModel

<http://qt-project.org/doc/qt-5.1/qtsql/qsqltablemodel.html#details>

QSqlTableModel класс обеспечивает редактируемую модель данных для одной таблицы базы данных. *Далее идёт описание данного класса, которое не следует изучать подробно, так как я очень хорошо знаком с данным классом. Теперь переходим к модулю тестирования.*

# QT ТЕСТ

<http://qt-project.org/doc/qt-5.1/qttestlib/qttest-index.html>

qt тест обеспечивает классы для модульного тестирования qt приложений и библиотек. Все публичные методы находятся в пространстве имён QTest. [QSignalSpy](http://qt-project.org/doc/qt-5.1/qttestlib/qsignalspy.html) класс обеспечивает лёгкий анализ для qt сигналов и слотов. *Далее показан импорт данного класса.*

## ОБЗОР КТ ТЕСТ

[*http://qt-project.org/doc/qt-5.1/qttestlib/qtest-overview.html*](http://qt-project.org/doc/qt-5.1/qttestlib/qtest-overview.html)

qt тест – это для модульного тестирования основанных на qt приложениях и библиотек. Qt тест обеспечивает всю функциональность, которая обычно имеет место для модульных тестов, так же как и расширения для тестирования графических интерфейсов пользователя. Qt тест разработан для облегчения написания модульного тестирования для основанных на qt приложений и библиотек.

|  |  |
| --- | --- |
| **Особенность** | **Детали** |
| Легковесность | Qt тесты состоят из 6000 линий кода и 60 экспортированных символов |
| Само-содержание | Qt тест требует только мало символов из qt ядра модуля для не GUI тестирования |
| Быстрое тестирование | Qt тест не требует специальных движков тестирования; а также специальной регистрации для тестов. |
| Тестирование, движимое данными | Тест можно выполнять много раз с разными данными тестов. |
| Базовое GUI тестирование | Qt тест предоставляет функциональность для симуляции мыши или клавиатуры |
| Бенчмаркинг | Qt тест поддерживает бенчмаркинг и обеспечивает несколько измерений бэкендов (*что это такое???*) |
| IDE дружественность | Qt тест выводит сообщения, которые могут быть интерпретированы при помощи Visual Studio, так и KDevelop. |
| Потокобезопасность | Сообщение об ошибках является поточно безопасным и атомарным. |
| Безопасность типа | Расширенное использование шаблонов предотвращает ошибки, вводимые при помощи неявного приведения типов. |
| Легко расширяемы | Частные типы могут легко быть добавлены к данным теста и выводу теста. |

Для создания теста специализируйте QObject и добавьте один или более приватных слотов к нему. Каждый приватный слот – это тестовая функция в вашем тесте. [QTest::qExec](http://qt-project.org/doc/qt-5.1/qttestlib/qtest.html#qExec)() может быть использован для выполнения всех тестовых функций в объекте теста. Вдобавок есть четыре приватных слота, которые не обрабатываются как тестовые функции. Они будут выполнены во время тестирования и могут быть использованы для инициализации или очистки как полного теста, так и текущей тестовой функции.

* initTestCase() будет вызываться перед выполнением первой тестовой функции.
* cleanupTestCase() будет вызываться после выполнения последней тестовой функции.
* init() будет вызываться перед выполнением каждой тестовой функции.
* cleanup() будет вызываться после каждой тестовой функции

*есть пример кода. далее приводятся примеры запуска теста в командной строке. Очень полезный пример кода. далее перечислены команды для получения разных типов вывода результатов тестирования.*

Если используется первая версия вывода, то не может быть использована ни одна из других версий вывода. Если не задано никаких требований к выводу, то результаты тестирования выводятся в виде простого текста стандартным образом.

*Далее перечислены команды, которые указывают на детализацию сообщений в тестовых логах. Далее перечисляются опции командной строки, которые влияют на то, как тесты выполняются. Далее перечисляются команды для управления тестированием производительности. Это очень полезные для тестирования команды!*

Для создания теста для производительности следуйте инструкциям для создания теста и затем добавьте макрос QBENCHMARK для функций тестирования, которые вы желаете протестировать на производительность. *Есть пример кода.* Время выполнения данной функции будет измерено. Можно задавать некоторые аргументы командной строки для определения того, как это будет измерено. *Они перечислены в данной части.*

Вкратце,

* walltime всегда доступен, но требует много повторений для получения полезного результата.
* Тактовый счётчик процессора обычно доступен и может обеспечить результат с меньшим количеством повторений, но может быть восприимчив к вопросам масштабирования частоты центрального процессора.
* Valgrind обеспечивает точные результаты, но не принимает во внимание ожидание ввода/вывода и только доступен на ограниченном количестве платформ.
* Счёт событий доступен на всех платформах и он обеспечивает количество событий, которые были получены циклом обработки событий перед тем, как они были высланы их соответствующим целям (это может включать и не qt-события).

## РУКОВОДСТО ПО ТЕСТИРОВАНИЮ

<http://qt-project.org/doc/qt-5.1/qttestlib/qtest-tutorial.html>

теперь рассмотрим руководство по тестированию.

### НАПИСАНИЕ МОДУЛЬНОГО ТЕСТИРОВАНИЯ

<http://qt-project.org/doc/qt-5.1/qttestlib/tutorial1.html>

В данной Главе рассматривается, как написать простой модульный тест для функции и как его выполнить. Вначале следует создать подкласс QObject, а затем написать в нём в качестве приватных слотов функции для тестирования. *Есть пример того, как это пишется.* [QVERIFY](http://qt-project.org/doc/qt-5.1/qttestlib/qtest.html#QVERIFY)() макрос оценивает выражение, переданное ему в качестве аргумента. Если оно оценивается в правду, то выполнение тестовой функции продолжается. Иначе, сообщение, описывающее крах, добавляется в тестовый лог, а функция прекращает выполнение. Но если вы хотите более многословный вывод в тестовым логе, вам следует использовать макрос [QCOMPARE](http://qt-project.org/doc/qt-5.1/qttestlib/qtest.html#QCOMPAREx)() вместо этого. Наконец, чтобы сделать наш тест автономно исполняемым, следует добавить две линии, *которые показаны в данной части кода.* [QTEST\_MAIN](http://qt-project.org/doc/qt-5.1/qttestlib/qtest.html#QTEST_MAIN)() макрос расширяет к простому методу main() метод, который выполняет все тестовые функции. Заметьте, что если и объявление, и реализация наших тестовых классов являются .cpp файлом, мы также должны включить генерируемый moc файл, чтобы сделать работу по анализу qt. Далее тест помещается в некоторую директорию, и при помощи qmake строится проект, который генерирует make-файл. В Windows следует использовать nmake вместо qmake. После этого следует запустить исполняемый файл и получить вывод. Можно просто создать файл проекта для тестирования. *Он показан среди файлов, которые предоставлены к первой Главе.*

### ТЕСТИРОВАНИЕ, УПРАВЛЯЕМОЕ ДАННЫМИ

<http://qt-project.org/doc/qt-5.1/qttestlib/tutorial2.html>

В данной Главе будет продемонстрировано, как выполнять тестирование много раз с разными данными для тестов. До сих пор мы имели жёстко закодированные данные, которые мы желали тестировать в нашей тестовой функции. Если мы добавим больше тестовых данных, функция может выглядеть следующим образом. *Показан пример кода.*

Qt тест поддерживает добавление тестовых данных в функцию для тестирования. Всё, что для этого нужно, так это добавить другой приватный слот <name>\_data. В этом слоте определяются элементы для тестов: наборы колонок. Затем к ним добавляют некоторые данные при помощи формирования строк, которым присваивают имена. *Есть пример кода. Есть замечательный пример определения тестовой функции.*

[QFETCH](http://qt-project.org/doc/qt-5.1/qttestlib/qtest.html#QFETCH)() макрос используется для выбора данных для тестирования функции. Данный макрос содержит два аргумента:

* тип данных элемента и
* имя элемента.

*Есть пример кода.* Данный подход позволяет легко добавить данные для тестирования, не изменяя тест.

### МОДЕЛИРОВАНИЕ GUI СОБЫТИЙ

<http://qt-project.org/doc/qt-5.1/qttestlib/tutorial3.html>

qt тест также имеет несколько механизмов для тестирования GUI. Вместо симулирования событий нативной системы окон, qt тест посылает внутренние qt события. Это означает, что нет побочных эффектов на машине, на которой выполняются тесты. *Есть пример кода.* Вначале создаётся необходимый графический элемент. Затем при помощи специальных функций симулируются некоторые события ввода. При этом виджет также следует показать, чтобы корректно выполнить тест. *В данной части перечислены функции для симулирования ввода/вывода. Полезные функции для тестирования. Виджеты следует показывать для тестирования горячих клавиш.*

### ПЕРЕИГРЫВАНИЕ СОБЫТИЕ GUI

[*http://qt-project.org/doc/qt-5.1/qttestlib/tutorial4.html*](http://qt-project.org/doc/qt-5.1/qttestlib/tutorial4.html)

В данной Главе мы покажем, как симулировать GUI события и как сохранять наборы GUI событий также, как и воспроизводить их в виджете. *Есть примеры кода. В данном случае используется класс* [QTestEventList](http://qt-project.org/doc/qt-5.1/qttestlib/qtesteventlist.html). Здесь снова следует определить ещё один частный слот. Затем следует написать функцию для предоставления данных. Затем переписывается тестовая функция. Это похоже на то, что было в Главе 2.

### НАПИСАНИЕ ТЕСТОВ НА ПРОИЗВОДИТЕЛЬНОСТЬ

<http://qt-project.org/doc/qt-5.1/qttestlib/tutorial5.html>

В этой последней Главе будет продемонстрировано, как писать тесты на производительность с использованием данного модуля. Для написания теста на производительность используется макрос QBENCHMARK. Тестовая функция на производительность обычно состоит из кода установки и данного макроса.

Код внутри данного макроса будет измерен, а также он может быть повторён несколько раз для получения более точного результата. Можно выбрать несколько бэк-ендов. *Что такое бэкенд, я не понимаю до сих пор.*

Также для тестов на производительность можно обеспечивать различный набор данных и смотреть, например, зависимость производительности от набора данных. Каждая тестовая функция на производительность может иметь один активный макрос. *Q\_UNUSED() макрос используется для показа, что не нужно использовать некоторую величину внутри тестовой функции.* Средства для обработки и визуализации тестовых данных доступны как часть проекта [qtestlib-tools](http://qt.gitorious.org/qt-labs/qtestlib-tools). *Есть ссылка на данный пример. Рассмотрим теперь тестовые классы.*

## КЛАССЫ ТЕСТОВ

<http://qt-project.org/doc/qt-5.1/qttestlib/qttest-module.html>

#### QTest

<http://qt-project.org/doc/qt-5.1/qttestlib/qtest.html#details>

QTest пространство имён содержит все функции и объявления, которые относятся к qt тест.

#### QSignalSpy

<http://qt-project.org/doc/qt-5.1/qttestlib/qsignalspy.html#details>

QSignalSpy класс предоставляет возможность анализа испускания сигналов. Он может присоединиться к любому сигналу любого объекта и сообщить о его испускании. Это список списков QVariant. Каждое испускание сигнала будет содержаться в списке один раз, показывая также и аргументы сигнала. *Есть примеры кода использования данного класса.* Нестандартные типы данных необходимо регистрировать с использованием функции [qRegisterMetaType](http://qt-project.org/doc/qt-5.1/qtcore/qmetatype.html#qRegisterMetaType)(), прежде чем вы сможете создавать QSignalSpy. *Есть пример кода.*

#### QTestEventList

<http://qt-project.org/doc/qt-5.1/qttestlib/qtesteventlist.html#details>

QTestEventList обеспечивает список событий GUI. *Есть пример кода использования данного класса.*

#### QTouchEventSequence

<http://qt-project.org/doc/qt-5.1/qttestlib/qtest-qtoucheventsequence.html#details>

QTouchEventSequence класс используется для симулирования последовательности событий прикосновений. *Данный класс пока мне не интересен. Переходим к изучению дополнительных модулей qt. Класс со скриптом qt является устаревшим, но вместо него следует использовать классы с расширением QJS из модуля qt qml.*

# КОНКАРЕНТ

[*http://qt-project.org/doc/qt-5.1/qtconcurrent/qtconcurrent-index.html*](http://qt-project.org/doc/qt-5.1/qtconcurrent/qtconcurrent-index.html)

[QtConcurrent](http://qt-project.org/doc/qt-5.1/qtconcurrent/qtconcurrent-module.html) пространство имён обеспечивает высокоуровневый программный интерфейс, который делает возможным написание многопоточных программ без использования низкоуровневых потоковых примитивов, таких как мьютексы, блокировки чтения/записи, семаформы или условия ожидания. Также данное пространство имён автоматически приводит в порядок использование доступных ядер для обработки разных потоков. Приложения, развёрнутые в будущем на многоядерных системах, будут успешно распределяться между ядрами. Данное пространство имён включает функциональное программирование для списка параллельных процессов, включая MapReduce and FilterReduce реализации для систем с разделяемой памятью, и классы для управления асинхронными вычислениями в GUI приложениях. *далее они перечислены.* Qt конкарент поддерживает несколько стл-совместимых контейнеров и типов итераторов, но работает лучше с qt контейнерами, которые имеют итераторы произвольного доступа, такие как QList и QVector. *Далее приводится таблица поддержки различных типов итераторов стл. Теперь рассмотрим некоторые классы данного модуля.*

## Qt Concurrent модуль

<http://qt-project.org/doc/qt-5.1/qtconcurrent/qtconcurrent-module.html>

Qt Concurrent модуль расширяет базовую поддержку многопоточности из модуля ядра и упрощает написание кода, который может быть использован параллельно на всех доступных ядрах центрального процессора. *Переходим к следующему модулю.*

## КЛАССЫ КОНКАРЕНТ

#### QFuture

<http://qt-project.org/doc/qt-5.1/qtcore/qfuture.html#details>

QFuture класс представляет результат асинхронных вычислений. Для начала вычисления следует использовать программный интерфейс qt конкарент. Данный класс позволяет потокам быть синхронизированными по отношению к одному или нескольким результатам, которые будут готовы в некоторый будущий момент времени. Результатом может быть любой класс, который имеет конструктор копирования и конструктор по умолчанию. Если результат не доступен во время вызова некоторых функций данного класса, то данный класс будет ждать, пока результаты не станут доступными. Можно использовать [isResultReadyAt](http://qt-project.org/doc/qt-5.1/qtcore/qfuture.html#isResultReadyAt)() функцию, чтобы определить, готов результат или нет. Для класса, который сообщает о более чем одном результате, есть функция [resultCount](http://qt-project.org/doc/qt-5.1/qtcore/qfuture.html#resultCount)(), которая возвращает число продолжающихся результатов. Данный класс предоставляет java итератор и стл итератор. При помощи них можно получить доступ к результатам. Класс также предоставляет способ взаимодействия с выполняющимися вычислениями. Его можно отменить, поставить на паузу. Но будьте внимательны, что не все асинхронные расчёты могут быть отменены или остановлены. Например, нельзя отменить расчёт, возвращаемый функцией QtConcurrent::run(). Также есть функция [progressValue](http://qt-project.org/doc/qt-5.1/qtcore/qfuture.html#progressValue)(), [progressMinimum](http://qt-project.org/doc/qt-5.1/qtcore/qfuture.html#progressMinimum)(), [progressMaximum](http://qt-project.org/doc/qt-5.1/qtcore/qfuture.html#progressMaximum)(), and [progressText](http://qt-project.org/doc/qt-5.1/qtcore/qfuture.html#progressText)() для обеспечения информации о прогрессе. Также есть функции для получения состояния расчёта. Это очень лёгкий класс, который можно передавать по значению. *Также сказано об удобстве использования* QFuture<void>. *Я не понимаю, как данный класс вычисляет прогресс???* Для взаимодействия с выполняющимися задачами с использованием сигналов и слотов следует применять класс [QFutureWatcher](http://qt-project.org/doc/qt-5.1/qtcore/qfuturewatcher.html).

#### QFutureIterator

<http://qt-project.org/doc/qt-5.1/qtcore/qfutureiterator.html#details>

QFutureIterator класс обеспечивает java итератор для класса [QFuture](http://qt-project.org/doc/qt-5.1/qtcore/qfuture.html). *Далее следует подробное описание данного класса с примерами кода и иллюстрациями.*

#### QFutureWatcher

[*http://qt-project.org/doc/qt-5.1/qtcore/qfuturewatcher.html#details*](http://qt-project.org/doc/qt-5.1/qtcore/qfuturewatcher.html#details)

QFutureWatcher класс позволяет следить за [QFuture](http://qt-project.org/doc/qt-5.1/qtcore/qfuture.html) с использованием сигналов и слотов. Он предоставляет информацию и уведомления о QFuture. *Есть очень хороший пример кода, который показывает работу данного класса. Именно так можно отслеживать прогресс вычислений.* QFutureWatcher<void> является специализированным классом, который не содержит функций выбора результата. Любые классы можно просмотреть с использованием данной функции. Это полезно, если нужна только информация о статусе прогресса, но не действительные результаты данных.

#### QFutureSynchronizer

<http://qt-project.org/doc/qt-5.1/qtcore/qfuturesynchronizer.html#details>

QFutureSynchronizer класс является удобным классом, который упрощает синхронизацию [QFuture](http://qt-project.org/doc/qt-5.1/qtcore/qfuture.html). Это шаблон, который упрощает синхронизацию одного или более объектов [QFuture](http://qt-project.org/doc/qt-5.1/qtcore/qfuture.html). Также в данном классе есть функции для добавления и удаления QFuture. Есть функции ожидания того, когда все QFuture завершатся. Эту функцию возвращает и деструктор данного класса, чтобы гарантировать, что все будущие будут завершены прежде удаления данного объекта. *Насколько я понимаю, основное предназначение данной функции состоит в том, чтобы ожидать, пока завершатся все будущие, которые в него входят. Нужны для лучшего понимания конкретные примеры использования данного класса.*

#### [QtConcurrent](http://qt-project.org/doc/qt-5.1/qtconcurrent/qtconcurrent-module.html)

<http://qt-project.org/doc/qt-5.1/qtconcurrent/qtconcurrent.html#details>

[QtConcurrent](http://qt-project.org/doc/qt-5.1/qtconcurrent/qtconcurrent-module.html) пространство имён обеспечивает высокоуровневый программный интерфейс, который делает возможным написание многопоточных программ без использования низкоуровневых поточных примитивов.

# ФОРМАТЫ ИЗОБРАЖЕНИЙ КТ

<http://qt-project.org/doc/qt-5.1/qtimageformats/qtimageformats-index.html>

Библиотека qt GUI по умолчанию поддерживает чтение и запись файлов изображений большинства общих форматов: PNG, JPEG, BMP, GIF и другие. Qt форматы изображений обеспечивает поддержку других форматов файлов изображений. Поддержка форматов обеспечивается прозрачно через систему плагинов qt системы ввода/вывода изображений. *Далее перечислены поддерживаемые форматы файлов.* При построении данные плагины локализуются как динамические библиотеки в директории выполнения плагинов, вместе с плагинами для форматов изображений по умолчанию. Так как эти форматы файлов используются реже, кодеки могут быть менее тщательно отлажены по отношению к потенциальным дырам безопасности. Как всегда, следите за созданием приложений, которые могут быть использованы для декодирования неподконтрольных файлов данных.

# КТ ПОСЛЕДОВАТЕЛЬНЫЙ ПОРТ

<http://qt-project.org/doc/qt-5.1/qtserialport/qtserialport-index.html>

Qt Serial Port обеспечивает базовую функциональность, которая включает конфигурирование, операции ввода/вывода, сигналы управления геттерами и сеттерами RS-232 pinouts. Данный модуль не обеспечивает:

* функции терминала, такие как echo, control CR/LF, and so on.
* Текстовую моду.
* Конфигурирование таймаутов и задержек при чтении.
* Отслеживание и уведомление того, когда сигнал RS-232 pinouts изменится.

RS-232 – это специальный тип порта, который постепенно вытесняется usb. *Далее показано, как импортировать данный модуль.*

<http://qt-project.org/doc/qt-5.1/qtserialport/qtserialport-module.html>

*рассмотрим классы для данного модуля.*

<http://qt-project.org/doc/qt-5.1/qtserialport/qserialport.html#details>

QSerialPort обеспечивает функции для доступа к последовательным портам. О доступных последовательных портах можно получить информацию с использованием класса [QSerialPortInfo](http://qt-project.org/doc/qt-5.1/qtserialport/qserialportinfo.html), которые предоставляет перечисление всех последовательных портов системы. Он полезен для получения правильного имени последовательного порта, который вы используете. Вначале следует установить порт, а затем открыть его для в одной из мод. Последовательный порт всегда открыт с экSQLюзивным доступом (ни один другой процесс или поток не может получить доступ к открытому порту). Есть функции для настройки конфигурации порта. *Далее показан пример использования данного класса, который в будущем следует изучить более подробно.* В графических приложениях блокирующие последовательные порты следует использовать только в неграфических потоках, чтобы избежать замораживания интерфейса пользователя. *Для более подробного изучения данного приближения следует изучить предоставляемые примеры к данному модулю.* Данный класс также можно использовать вместе с [QTextStream](http://qt-project.org/doc/qt-5.1/qtcore/qtextstream.html) and [QDataStream](http://qt-project.org/doc/qt-5.1/qtcore/qdatastream.html)'s операторами потока. При этом следует всегда быть уверенным, что достаточно данных доступна перед попыткой считывания с использованием оператора >>.

<http://qt-project.org/doc/qt-5.1/qtserialport/qserialportinfo.html#details>

QSerialPortInfo класс обеспечивает информацию о существующих последовательных портах. Следует использовать статические функции для генерирования списка объектов данного класса. Каждый объект в списке представляет отдельный последовательный порт, а также может быть запрошен по имени порта, системной локации, описанию или производителю. Он также может быть использован как параметр ввода для метода setPort() класса [QSerialPort](http://qt-project.org/doc/qt-5.1/qtserialport/qserialport.html). *Вот почему в кт нет поддержки для последовательных портов для usb??? Или как это сделать???*

# QT SVG

<http://qt-project.org/doc/qt-5.1/qtsvg/qtsvg-index.html>

Scalable Vector Graphic (SVG) – это основанный на xml язык для описания двумерной векторной графики. Qt предоставляет классы для рисования и отображения SVG в виджетах и на других устройствах рисования. *Далее показан импорт данного модуля.*

## РИСОВАНИЕ SVG ФАЙЛОВ

<http://qt-project.org/doc/qt-5.1/qtsvg/svgrendering.html>

Наиболее лёгким способом рисования SVG является построение [QSvgWidget](http://qt-project.org/doc/qt-5.1/qtsvg/qsvgwidget.html), который может быть использован напрямую, чтобы обеспечить SVG поддержку для частных виджетов. Для загрузки SVG файлов строится [QSvgRenderer](http://qt-project.org/doc/qt-5.1/qtsvg/qsvgrenderer.html) с именем файла и содержанием файла, или вызывается функция [QSvgRenderer::load](http://qt-project.org/doc/qt-5.1/qtsvg/qsvgrenderer.html#load)() для существующего рисовальщика. Если SVG был успешно загружен, то [QSvgRenderer::isValid](http://qt-project.org/doc/qt-5.1/qtsvg/qsvgrenderer.html#isValid)() возвратит истину. После успешной загрузки SVG его можно успешно нарисовать с использованием [QSvgRenderer::render](http://qt-project.org/doc/qt-5.1/qtsvg/qsvgrenderer.html#render)(). *Есть также ссылка на некоторый пример. Далее приводятся некоторые лицензионные соглашения.*

## КЛАССЫ SVG

<http://qt-project.org/doc/qt-5.1/qtsvg/qtsvg-module.html>

#### QGraphicsSvgItem

<http://qt-project.org/doc/qt-5.1/qtsvg/qgraphicssvgitem.html#details>

QGraphicsSvgItem класс является [QGraphicsItem](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsitem.html), которая может быть использована для рисования содержания SVG файлов. Данный класс предоставляет способ рисования SVG файлов на [QGraphicsView](http://qt-project.org/doc/qt-5.1/qtwidgets/qgraphicsview.html). Данный класс может использоваться для отображения лишь некоторой части данного файла. *Далее описываются некоторые важные функции данного класса и приводится пример кода.*

#### QSvgGenerator

[*http://qt-project.org/doc/qt-5.1/qtsvg/qsvggenerator.html#details*](http://qt-project.org/doc/qt-5.1/qtsvg/qsvggenerator.html#details)

QSvgGenerator класс обеспечивает устройство рисования, которое используется для создания SVG рисования. *Далее приводится пример кода для использования данного класса.*

#### QSvgRenderer

<http://qt-project.org/doc/qt-5.1/qtsvg/qsvgrenderer.html#details>

QSvgRenderer класс используется для рисования содержания SVG файлов на устройстве рисования. *Далее описываются некоторые функции данного класса.*

#### QSvgWidget

<http://qt-project.org/doc/qt-5.1/qtsvg/qsvgwidget.html#details>

QSvgWidget класс обеспечивает виджет, который используется для отображения содержания SVG файлов. Класс позволяет отображать SVG файлы отдельно от остальных виджетов. *Пока с данным модулем всё. Следует почитать более подробно, что такое SVG. Возможно, это мне пригодится.*

# ЭКСТРА ВОЗМОЖНОСТИ ДЛЯ Х11

<http://qt-project.org/doc/qt-5.1/qtx11extras/qtx11extras-index.html>

Qt X11 Extras предоставляет возможность qt программисту написать приложения для Linux/Х11 платформы. Приложения, разрабатываемые при помощи qt, могут также разворачиваться на других десктопах и встроенных операционных системах без переписывания кода. *Далее показывается импорт данного модуля.*

## КЛАССЫ ЭКСТРА ПОДДЕРЖКИ Х11

<http://qt-project.org/doc/qt-5.1/qtx11extras/qtx11extras-module.html>

#### QX11Info

<http://qt-project.org/doc/qt-5.1/qtx11extras/qx11info.html#details>

QX11Info обеспечивает информацию о конфигурации дисплея Х. Класс обеспечивает два программных интерфейса:

* набор нестатических функций, которые обеспечивают информацию об отдельном виджете или о растре;
* и набор статических функций, которые обеспечивают информацию по умолчанию для приложений.

Данный класс доступен только на Х11. Для запроса информации об экране переносимым способом используйте класс [QDesktopWidget](http://qt-project.org/doc/qt-5.1/qtwidgets/qdesktopwidget.html). *Также есть модуль для облегчения печати, но этот модуль я пока не буду изучать. Если понадобится его использование, то я изучу его.*

*УСЁ!!!!!*

*Наступная задача – вывучыць лінукс і павывучаць усе кт прыклады!!!*

*Сёння 08.03.13 і я скончыў структурыраванне бібліятэкі Qt.*