

سطح مقدماتی



برنامه‌نویسی مدرن C++ همراه با کتابخانه کیوت

نویسنده : کامبیز اسدزاده
شماره شابک : 978-600-04-4451-8

عنوان کتاب	برنامه‌نویسی مُدرن C++ همراه با کتابخانه Qt
مؤلف	کامبیز اسدزاده
ناشر	ناشر مولف (کامبیز اسدزاده)
نوبت و سال چاپ	۹۵/۰۴/۲۱
قطع و تیراژ	استاندارد در قالب A4 در ۱۱۸ صفحه
نوبت چاپ	اول - سال ۱۳۹۵
شابک	978-600-04-4451-8
رده بندی کنگره	۹ / QA76/۵الف۹۸ ک ۱۳۹۶
رده بندی دیویی	۵۰۰/۱۳۳
آخرین به روز رسانی	۱۳۹۷/۱۲/۱۲
نسخه ویرایشی	۵.۰
پیش نیازها	تسلط متوسط به بالا از C++ مُدرن
شماره کتاب شناسی ملی	۴۷۴۳۱۳۵
قیمت	۲۰.۰۰۰ تومان

فهرست

فصل اول

۶ مقدمه‌ی کتابخانه Qt
۱۳ نسخه‌های کیوت
۱۶ مجوزهای موجود در این کتابخانه
۱۸ محیط‌های توسعه کیوت
۱۹ ویژگی‌های کیوت
۲۴ پشتیبانی از انواع سیستم‌عامل‌ها
۲۷ نصب و پیکربندی Qt

فصل دوم

۳۳ انواع پروژه و ایجاد آن
۳۴ انواع پروژه‌ها
۳۵ ایجاد پروژه

فصل سوم

۴۳ ساده‌ترین برنامه
۴۴ معرفی و کار با Signal و Slot و Event
۴۷ معرفی و کار با نمایش Windows
۵۴ معرفی و کار با لایه‌ها، زبانه‌ها و بدن‌های در طراحی

۵۵ معرفی و کار با قابلیت‌های HTML و CSS در طراحی

فصل چهارم

۵۸ معرفی و کار با لایه‌های افقی و عمودی

۶۰ معرفی و کار با لایه‌های Grid در طراحی فرم

۶۲ معرفی و کار با جدا کننده‌ها Splitter

فصل پنجم

۶۳ معرفی و کار با دایرکتوری ها

۶۵ معرفی و کار با فایل‌ها، خواندن و نوشتan در آن‌ها

فصل ششم

۷۰ معرفی و کار با برچسبها Label

۷۰ معرفی و کار با دکمه‌ها Button

۷۰ معرفی و کار با کنترل ورودی LineEdit

۷۱ معرفی و کار با چک باکس CheckBox

۷۳ معرفی و کار با RadioButton

۷۵ معرفی و کار با Combobox

۷۷ معرفی و کار با ListWidget

۸۰ معرفی و کار با لیست‌های درختی TreeWidget

۸۳ معرفی و کار با Actionها

۸۵ معرفی و کار با Progress Slider و Sliderها

۸۶ معرفی و کار با Statusbar ها

فصل هفتم

۸۸ معرفی و کار با MessageBox

۹۲ معرفی و کار با Timer

۹۴ معرفی و کار با Thread ها

فصل هشتم

۹۷ معرفی و کار با Map

۹۹ معرفی و کار با Hash

۱۰۰ معرفی و کار با QList رشته‌ای

فصل نهم

۱۰۴ معرفی و کار با الگوریتمها

۱۱۰ معرفی و کار شبکه / دانلود فایل بر اساس پروتکل‌های HTTP و FTP

۱۱۹ معرفی و کار با باینری و سریالیز کردن آبجکت‌ها

۱۲۳ معرفی و کار با QTextStream ها

فصل دهم

۱۲۴ مقایسه انواع حالت‌های کامپایل در Qt

۱۲۶ نحوه افزودن دیگر کتابخانه‌های C++ در محیط Qt Creator و استفاده همراه با کتابخانه Qt

۱۳۰ نحوه خروجی گرفتن، گسترش (Deployment) در Qt

۱۳۶	مقایسه و پیکربندی دو موتور قدرتمند OpenGL و ANGLE در پروژه
۱۳۷	دراایور دیتابیس هایی که تحت این کتابخانه پشتیبانی می‌شوند
۱۴۰	ساخت راهانداز دیتابیس در پلفترم‌های Windows، macOS و Linux
۱۴۳	حق نشر کتاب و اهداف در نسخه‌ی بعدی کتاب

مقدمه‌ی کیوت (Qt 5.12.1)

کیوت (Qt) مجموعه‌ای از کتابخانه‌ها و سرآیندهای نوشته شده به زبان C++ است که به برنامه‌نویس امکان توسعه آسان نرم‌افزارهای کاربردی را می‌دهد. اگر به فکر این هستید که در این زمینه یک حرف‌ای محسوب شوید بهتر است به جزئیات آن دقت کنید و از نام آن شروع کنید! نام کیوت در لاتین به صورت Qt نوشته می‌شود و حرف دوم آن یعنی t بهتر است به صورت lowercase یا همان حرف کوچک نوشته شود. طبق این قانون شکل صحیح آن **Qt** است نه **QT**. و اما در رابطه با تلفظ این نام بر خلاف انتظار آوای صحیح آن به صورت "کیوت" مشابه تلفظ "Cute" است و تلفظ **کیوت** اشتباه است.

علاوه بر آن **توجه داشته باشید کیوت یک زبان نیست! لذا در تمامی مراحل تولید یک محصول شما به زبان C++ در حال توسعه خواهد بود.**

کیوت شامل کلاس‌هایی برای کار با واسط گرافیکی، چندرسانه، ابزارهای پایگاه داده، شبکه و ... است. کیوت همانند دیگر کتابخانه‌های سی‌پلاس‌پلاس بوده و یک کتابخانه غیر STL محسوب می‌شود. نرم‌افزارهای نوشته شده با ابزار کیوت قادر هستند تا با استفاده از یک مترجم زبان C++ برای طیف وسیعی از سیستم‌عامل‌ها از جمله گنو/لینوکس (نسخه‌های رومیزی و وسیله‌های قابل حمل)، ویندوز، ویندوز CE، مک او اس، آی او اس، اندروید و ... همگرданی شوند. بدین ترتیب حمل نرم‌افزار نوشته شده بدون تغییر در متن کد نوشته شده امکان‌پذیر است که همانند کتابخانه‌های دیگر C++, STL, Boost, POCO, wxWidgets و ... مورد استفاده قرار می‌گیرد با تفاوت اینکه در زمینه طراحی رابط گرافیکی نسبت به کتابخانه‌های دیگر قدرتمندتر عمل می‌کند.

امروزه توسعه‌ی نرم‌افزار و بهروزرسانی‌های آن در انواع بسترهای (سکوها) از قبیل Windows، Linux، macOS و همچنین بسترهای موبایل و تبلت از قبیل Andoird، iOS، Backberry و ... با سرعت بسیار زیادی دنبال می‌شود همچنین آرزوی اکثر برنامه‌نویسان این است که یک زبان ویژه با تمامی قابلیت‌ها و مهمتر از همه پشتیبانی از شیء‌گرایی (OOP) و Performance (کارایی) بالا را همراه با یک IDE (محیط توسعه) همه‌کاره و جذاب در اختیار داشته باشند که در این صورت به جای تجربه کردن تمامی محیط‌های برنامه‌نویسی در این زمینه‌ها پیشنهاد می‌کنیم خیلی راه دوری نزدیک زیرا با استفاده از محیط و کتابخانه‌ی برنامه‌نویسی Qt که توسط زبان قدرتمند C++ مورد استفاده قرار می‌گیرد تقریبا همه آرزوهای شما در برنامه‌نویسی فراهم خواهد شد.

چرا باید از کیوت استفاده کنیم؟

زبان سی‌پلاس‌پلاس در حال توسعه و رشد بسیار زیادی است که بر اساس آخرین نظریه‌ی توسعه‌دهندگان بیش از ۹۵ درصد برنامه‌نویسی مدرن در حوزه‌اینترنت اشیاء تا سال ۲۰۲۰ نزدیک به ۳۰ میلیارد دستگاه را تحت سلطه‌ی کتابخانه Qt توسعه داده خواهد شد؛ طبیعی است که همانند زبان‌های دیگری در طی این سالها پیشرفت کرده‌اند زبان C++ هم خالی از پیشرفت نباشد و نسبت به قبل بسیار توانمند و قدرتمند شده است؛ مستندات نشان می‌دهد که نسخه‌های مُدرن در این میان نه تنها در رابطه با قابلیت‌ها موارد زیادی در استانداردهای ۱۱، ۱۴ و ۱۷ این زبان رفع و توسعه داده شده است در کنار این پیشرفت‌ها محیط توسعه بسیار جذابی به کمک برنامه‌نویسان و مشتقان این زبان آمده است آن چیزی نیست به جز Qt Creator یک محیط توسعه بسیار قدرتمند و کامل؛ همه چیز ساده‌تر، روانتر و جذاب‌تر شده است و سرعت برنامه‌نویسی و طراحی فرم‌ها و قالب‌بندی‌های پیشرفت‌هه که قبلاً نیاز به کدنویسی‌های بسیار بیشتری داشت بسیار بهتر از قبل شده به طوری که به جرأت می‌توان گفت در نگاه اول کار با Qt را می‌توان پسندید، این محیط بر خلاف محیط‌های معروف دیگری مانند Visual Studio و Xcode به هیچ عنوان سیاست انحصاری را ندارد و فقط برای سکوی خاصی محدود نشده‌اند.

از قابلیت‌هایی که نمی‌شود به این راحتی از آن‌ها چشم پوشی کرد می‌توان به ویژگی Cross-platform (چند-سکویی) بودن برنامه‌های تولید شده توسط Qt/C++/Qt اشاره کرد که شما به راحتی می‌توانید خروجی را در سیستم‌عامل مورد نظرتان کامپایل و اجرا کنید. به جای درگیر شدن با محیط‌های توسعه Visual Studio، Xcode، Android Studio و ... کافی است محیط توسعه Qt Creator را در اختیار داشته باشید تا بتوانید برای هر آنچه که در ذهن دارید برنامه‌های خود را طراحی، توسعه و تولید نمایید.

اگر شما طالبِ برنامه‌نویسی مدرنی هستید که برای شما لذت بخش باشد بدون شک سی‌پلاس‌پلاس می‌تواند شما را شبخته‌ی خود کند. چرا که کتابخانه‌های بزرگ و قدرتمند این زبان همچون کیوت با تمام قدرت آمده‌اند تا شما را به یک برنامه‌نویس افسانه‌ای تبدیل کنند. برنامه‌نویسی که بتواند با یک تیر چند نشان را هدف قرار دهد. اگر یکی از آرزوهای شما این است که بتوانید به عنوان چندین برنامه‌نویس خود به تنها‌ی همه کارها را انجام دهید تحت Qt ممکن خواهد شد! چرا که نیازی نیست شما نگران تولید محصول خود با محدودیت بستر یا فناوری‌ها باشید. کافی است خلاقانه فکر کرده و محصول خود را بعد از اعتبارسنجی وارد مرحله توسعه نمایید. البته همه‌ی این حرف‌ها به این معنی نیست که اگر شما تنها یک برنامه‌نویس بکاند یا

فرانت‌اند هستید خواهید توانست به راحتی برنامه‌نویسی کنید، چرا که برای حرفه‌ای شدن باید به فول‌استک شدن تمکز داشته باشید و مدام به فکر خلاقیت باشید.

یکی از بزرگترین مشکلاتی که در صنعت نرم‌افزاری و همچنین عدم موفقیت‌های استارت‌آپ‌های نوپا موجود است، درگیر بودن برنامه‌نویسان با ابزارها و زبان‌های برنامه‌نویسی متعدد است. برای مثال عاملین استارت‌آپ‌ها معمولاً به دنبال انواع برنامه‌نویس‌های متخصص در حوزه‌های ویندوز، لینوکس، مک، اندروید و غیره... می‌باشند و این مشکل را می‌توان با در نظر گرفتن فناوری چند-سکویی تحت یک زبان قدرتمند و یک سازنده رابط سریع به بهترین حالت ممکن مورد استفاده قرار داد. البته توجه داشته باشید برنامه‌نویسی چند-سکویی به صورت بومی تحت C++ قابل مقایسه با برنامه‌نویسی چندسکویی رایج تحت زبان‌های دیگر نیست چرا که چند منظوره بودن یکی از ذات‌های این زبان است.

پیش‌نیازات برای یادگیری کتابخانه کیوت: مطمئنان می‌دانید که جهت یادگیری هر مبحثی از برنامه‌نویسی نیاز بر داشتن دانش و اطلاعات کافی از پیش‌نیازات آن بسیار مهم است. هرچند کیوت به عنوان یک چهارچوب و کتابخانه‌ی کامل‌شفاف به شما در توسعه‌ی برنامه‌های مورد نظرتان کمک می‌کند، اما لازم است توجه داشته باشید که موارد اشاره شده در زیر برای یادگیری بهتر و درک سریع‌تر این کتابخانه مهم هستند.

۱. دانش متوسط به بالا در رابطه با زبان برنامه‌نویسی مُدرن سی‌پلاس‌پلاس نسخه‌های ۱۱ به بعد، (بنابراین

اگر شما هیچ اطلاعی در رابطه با ساختار برنامه‌های سی‌پلاس‌پلاس و نحوه عملکرد آن ندارید، شناس موفقیت شما بسیار پایین خواهد بود و ممکن است برنامه‌ی تولید شده‌ی شما به بدترین شکل ممکن پیاده سازی شود). ما پیشنهاد می‌کنیم قبل از آن با زبان سی‌پلاس‌پلاس آشنا شوید.

۲. آشنایی با مترجم و نحوه عملکرد آن در بسترها مختلف از جمله ویندوز، مک‌اواس، لینوکس، اندروید و آی‌اواس.

۳. آشنایی با برنامه‌نویسی در معماری‌های مختلف مانند x86، x64، Arm و غیره...

۴. آشنایی با معماری سیستم‌عامل‌ها، برای مثال اگر قرار است از خاصیت چند-سکویی کیوت استفاده کنید تا یک برنامه‌ی تحت اندروید را توسعه دهید، در این صورت باید در نظر داشته باشید که شما بدون

درکِ معماری سیستم‌عامل اندروید و پیکربندی برنامه در مراحل توسعه نمی‌توانید به راحتی از پس این کار بر آید.

۵. آشنایی با اصطلاحات و مفاهیم تجربه‌کاربری و رابط‌کاربری جهت طراحی مناسب با فناوری‌های Qt .Qt Widgets و Quick

۶. آشنایی JavaScript و XML برای طراحی رابط گرافیکی تحت فناوری‌های فوق.

آیا کیوت یک زبان برنامه‌نویسی است، چرا نحو (Syntax) آن با سی‌پلاس‌پلاس استاندارد فرق می‌کند؟ خیر، کیوت ابتدا به عنوان یک کتابخانه‌ی رابط گرافیکی کاربر توسعه داده شده است که بعدها برای توسعه اهداف بیشتری شامل کتابخانه‌های شبکه و غیره شده است که در قالب یک چهارچوب (فریم‌وُرک) که تحت زبان برنامه‌نویسی سی‌پلاس‌پلاس برای این زبان توسعه یافته است و در زمینه‌های مختلفی کاربردهای فراوان دارد. دلیل زیبایی و ظاهر ساده‌ی آن ساختار بسیار قدرتمند آن است که موجب شده همانند چهارچوب‌های قدرتمند دیگری خودنمایی کند.

آیا کیوت از سرویس‌ها و قابلیت‌های اختصاصی اپلیکیشن‌های اندروید و آی‌اواس را به طور کامل پشتیبانی می‌کند؟ خیر، هیچ ابزاری به صورت چند-سکویی فعلًاً (تاكید می‌کنیم - فعلًاً تا به این تاریخ) وجود ندارد که تمامی امکانات اختصاصی این بسترهای پشتیبانی کند. اما این به این معنی نیست که جواب کاملاً منفی خواهد بود، لذا شما در برنامه‌نویسی سی‌پلاس‌پلاس به راحتی می‌توانید برای بسترهای فوق با سرویس‌های آن‌ها ارتباط برقرار کنید. این کار، کمی نیاز به دانش فنی بالایی خواهد داشت. برای مثال (دسترسی به سرویس‌های اندروید یا آی‌اواس) با ترکیب گُدهای آبجکتیو-سی و جاوا امکان‌پذیر است. این کار به راحتی قابل انجام خواهد بود و کافی است شما در رابطه با نحوه ترکیب گُدهای این زبان‌ها مطلع باشید.

کیفیت خروجی برنامه‌های تحت کیوت چگونه است؟ اگر شما واقعًا یک برنامه‌نویس ماهر سی‌پلاس‌پلاس باشید می‌توانید برنامه‌ای را تولید کنید که بسیار خوش دست‌تر و سریع‌تر از برنامه‌های پیشفرض بسترهای باشد. اگر غیر از این باشد برنامه‌ای شما بسیار بد و کُند عمل خواهد کرد و بهتر است سراغ

زبان‌های بومی (هر بستر) بروید. هرچند سی‌پلاس‌پلاس یک زبان بومی برای تمامی بسترهای محسوب می‌شود اما این ریسک برای افراد مبتدی پیشنهاد نمی‌شود. معمولاً برنامه‌های گسترده و عظیم توسط سی‌پلاس‌پلاس توسعه داده می‌شوند که برنامه‌نویسان آن واقعاً حرفه‌ای و با قوانین این زبان آشنا هستند.

آیا برای تولید برنامه‌های مک و آی‌اواس نیاز به سیستم خاصی داریم؟ بله، شما بدون وجود سیستم‌عامل مک نمی‌توانید برنامه‌ای را بر روی دستگاه‌های اپل کامپایل کنید. این امر مستلزم سیستم‌عامل اختصاصی این شرکت بوده و باید توسعه دهنده دارای حساب کاربری معتبر سالانه باشد تا بتواند برنامه‌ی خود را بر روی دستگاه‌های مورد نظر اجرا کند. در غیر این صورت تنها می‌تواند برنامه‌ی خود را بر روی سیستم خود مورد آزمایش و خطا قرار دهد.

آیا کیوت از بستر وب پشتیبانی می‌کند؟ کیوت از تمامی بسترهای پشتیبانی می‌کند، مخصوصاً با پشتیبانی از مازول‌های QtWebAssembly و QtWebEngine این امر امکان‌پذیر است که برنامه‌های خود را تحت فناوری‌های وب نیز توسعه دهید.

آیا من حتماً باید به زبان سی‌پلاس‌پلاس مسلط باشم؟ ما بر این تاکید نمی‌کنیم که شما حتماً باید سی‌پلاس‌پلاس را کامل یاد بگیرید، این زبان به قدری گسترده است که نمی‌توان درباره‌ی آن به این راحتی مدعی شد. اما قوانین و اصول اولیه و عامیانه‌ی زبان بسیار نیاز است. کتاب‌ها و مقالات مربوط به این زبان را بیاموزید و تا جایی که می‌توانید اطلاعات خود را ارتقاء دهید.

آیا واقعاً حجم برنامه‌های کیوت نسبت به Java بیشتر است؟ خیر، به طور ذاتی برنامه‌های توسعه داده شده توسط سی++ دارای کمترین حجم برنامه هستند، معمولاً کتابخانه‌های استاندارد این زبان به صورت پیشفرض بر روی سیستم‌عامل‌ها تعییه شده و در دسترس قرار دارند. اما شما در ویندوز زمانی که با دات‌نت برنامه‌نویسی می‌کنید، فریمورک مربوطه از قبل بر روی سیستم‌عامل ویندوز نصب بوده و بدون آن هیچ برنامه‌ی نوشته شده توسط دات‌نت قابل اجرا است. اما چون از قبل این کتابخانه بر روی سیستم‌عامل تعییه شده است شما تنها نیاز به داشتن فایل اجرایی دارید و نیازی نیست فریمورک دات‌نت

را در کنار فایل اجرایی خود مستقر کنید. بنابراین حجم مربوط به چهارچوب به چشم نیامده و اینطور به نظر می‌رسد که برنامه‌های تحت دات نت بسیار سبک‌تر هستند! متاسفانه این تفکری اشتباه است! برنامه‌های تحت سی‌پلاس‌پلاس نسبت به زبان‌های دیگر کم حجم‌تر و سبک‌تر بوده و شما کافی است فایل‌های مربوط به کتابخانه را در کنار برنامه‌ی خود داشته باشید. در رابطه با کتابخانه‌ی Qt نیز باید گفت کیوت به عنوان یک چهارچوب مانند دات نت شامل کلاس‌ها و مازول‌هایی است که باید همانند دات نت بر روی سیستم عامل تعییه شود اما چون اینکار به صورت جداگانه در کنار برنامه‌ی شما قرار می‌گیرد اینگونه تصور می‌شود که برنامه‌های مبتنی بر کیوت نسبت به دات نت از حجم بیشتری برخوردار هستند. این کاملاً طبیعی بوده و به عنوان نکته ضعف نیست.

آیا لیستی برای مشاهده‌ی برنامه‌های توسعه یافته توسط Qt داریم؟ بسیاری از برنامه‌های قدرتمند و خارق‌العاده‌ای توسط سی‌پلاس‌پلاس توسعه داده می‌شوند که می‌توان لیست عظیمی از آن‌ها را نام برد. اما در این میان لیستی از برنامه‌هایی که تحت کیوت توسعه داده شده‌اند در [این بخش](#) آمده است. علاوه بر این که شما می‌توانید در مرجع به دنبال آموزش‌های مرتبط با این زبان باشید، پیشنهاد ما این است که از مراجع رسمی آن نیز استفاده کنید. برخی از آن‌ها به صورت زیر آمده‌اند:

- cppreference.com
- <http://www.cplusplus.com/>
- [Learn C++](http://LearnCPlusPlus.com)
- [C++ Tutorial | SoloLearn: Learn to code for FREE!](http://SoloLearn.com/CPP)
- [Learn C++ \(Introduction and Tutorials to C++ Programming\)](http://www.learn-cpp.com/introduction-to-cpp-programming)

قابلیت‌ها در طراحی : قابلیت طراحی فوق العاده با استفاده از فناوری QML و همچنین پشتیبانی از CSS و HTML یکی دیگر از مزایای Qt است که به کمک آن می‌توان طراحی منحصر بفرد و قابل توجهی را خلق کرد.

QML چیست؟ یک زبان بر پایه‌ی جاواسکریپت (JavaScript) و برنامه‌نویسی اعلانی برای ایجاد واسطه‌های کاربری برای برنامه‌های کاربردی است. این زبان بخشی از فناوری Qt Quick (کیت آن توسعه یافته به وسیله شرکت DIGIA) است که البته در حال حاضر توسط شرکت کیوت پشتیبانی و توسعه داده می‌شود. کیو-آم-آل به طور عمده برای نرم‌افزارهای موبایل و دسکتاپ از نوع رابطه‌های خلقانه استفاده می‌شود و بر سه پایه استوار است.

به عنوان حامل، اشیا (مستطیل، دایره، مثلث، عکس و...) و رفتارها (حالت‌ها، انتقال‌ها، اینیمیشن‌ها). این المان‌ها می‌توانند برای ساخت قطعات پیچیده از قطعات ساده مانند کلیدها و لغزندگان به منظور استفاده در برنامه‌های کاربردی و قابل دسترس از طریق اینترنت استفاده شود.

این المان‌ها همچنین می‌توانند با استفاده‌ی درون برنامه‌نویسی جاوا اسکریپت و همچنین فریمورک کیوت بر پایه‌ی زبان برنامه‌نویسی C++ توسعه پیدا کند.



استفاده از قابلیت‌های HTML یکی از بهترین و جذابترین موارد می‌تواند باشد که در برنامه‌نویسی Desktop و Mobile بسیار جذاب خواهد بود همه‌ی این قابلیت‌ها دست به دست هم می‌دهند تا نویسنده در کمترین زمان بدون نیاز به صرف زمان بیشتر طراحی رابطکاربری را با بهترین فناوری فراهم سازد. بنابراین با داشتن دانش C++ و اطلاعات کافی در رابطه با زبان‌های رابطکاربری همچون "HTML,CSS,JavaScript,QML" و آشنایی با محیط Qt که یک نوع فریم ورک ویژه‌ای برای این زبان است منجر به یک خروجی خارق العاده‌ای می‌شود. در کنار این محیط با کیفیت بالا و همچنین طراحی مدرن همه و همه در خروجی نهایی برنامه‌ی شما قابل حس خواهند بود.

طراحی سنتی تحت Widgets: در این روش شما از رابطکاربری سنتی تحت دسکتاپ استفاده خواهید کرد، شما می‌توانید توسط کشیدن و رها کردن کنترل‌ها و ابزارهای لازم به صورت بومی رابطکاربری خود را بر پایه‌ی Qt Widgets خلق کنید که در این کتاب به آن اشاره شده است.

طراحی مدرن با ترکیب QML و HTML5

همزمان تحت موتور **WebView** امکانپذیر است و به شما اجازه می‌دهد تا HTML و QML را همزمان با یکدیگر ترکیب کرده و اقدام به خلق رابطکاربری مورد نظر نمایید.

طراحی مدرن تحت JavaScript و QML

فناوری **Qt Quick** که در نهایت ظاهر سفارشی و حسی متفاوت را منعکس خواهد کرد. این شیوه در کتاب برنامه‌نویسی پیشرفته تحت کیوت آموزش داده شده است.

*نکته: این شیوه بیشتر برای خلق رابطهای خلاقانه و زیبا تحت دستتاب و موبایل پیشنهاد می‌شود.

نسخه‌های Qt

در رابطه با نسخه‌های کیوت باید گفت خوشبختانه این کتابخانه در تمامی سکوهای موجود قابل استفاده است که در زیر به آن‌ها اشاره شده است:

- (Unix / Linux) برای خانواده‌ی Qt/X11
- iOS برای مکینتاش قابلیت پشتیبانی در Qt/Mac macOS و
- Windows برای ویندوز Qt/Windows
- Qt/Embedded وسایل همراه (PDA, تلفن هوشمند و غیره)
- Windows CE برای Qt/WinCE
- Java برای Qt Jambi
- Qt Extended برای سیستم‌عامل لینوکس نسخه وسایل همراه

تمامی این موارد برای دریافت موجود است و شما می‌توانید نسخه مربوط به هرکدام را به صورت آنلاین و آفلاین دریافت نمایید که در زیر لینک آن‌ها آمده است.

- [نسخه‌ی ۳۲ بیتی آنلاین بستر ویندوز](#)
- [نسخه‌ی ۶۴ بیتی آنلاین بستر مک اواس](#)
- [نسخه‌ی ۳۲ بیتی آنلاین بستر لینوکس](#)

- [نسخه ۶۴ بیتی آنلاین ستر لینوکس](#)

نکته: نسخه‌های ۳۲ بیتی به این معنی نیستند که شما تنها می‌توانید نسخه‌ی ۳۲ بیتی محصول خود را کامپایل کنید. منظور از آن معماری سکوی قابل توسعه‌ی سیستم شما است.

نسخه‌های آفلاین، مستقل کیوت (Offline Installers "stand-alone") بر اساس نوع معماری و نوع سیستم‌عامل قابل دریافت است.

- نسخه‌ی مربوط به سکوی لینوکس "Unix/Linux"

- [دریافت کیوت برای ۶۴ بیتی ستر لینوکس \(ابونتو\)](#)

- نسخه‌ی مربوط به سکوی یونیکس "Unix/OSX" بر اساس نوع سیستم‌عامل

- [دریافت کیوت برای ۶۴ بیتی ستر مک‌اواس](#)

- نسخه‌ی مربوط به سکوی ویندوز "DOS/WINDOWS" بر اساس نوع سیستم‌عامل

- [ویرایش ۳۲ بیتی کیوت \(تحت ویندوز\)](#)

همچنین شما می‌توانید خودتان اقدام به کامپایل Qt نمایید.

توجه: نیازی نیست شما برای هر کدام از بسترهای دوباره نویسی و کدنویسی مجدد انجام دهید شما می‌توانید برای شروع برای ویندوز یا لینوکس یک نسخه مورد نظر را دریافت کنید و برنامه خود را بنویسید در نهایت که اگر نیازی برای کامپایل بر روی سیستم‌های دیگری مانند iOS و Mac یا Android بود تنها کافی است نسخه مربوط به آن سیستم‌عامل را دریافت و توسط آن پروژه خود را Import نموده و سپس مجددا کامپایل نمایید.



مجوزهای موجود در این کتابخانه

نوع مجوز و ویژگی‌ها	LGPLV3	GPLV2/GPLV3	مجوز تجاری
هزینه	رایگان	رایگان	شروع هزینه از ۴۵۹ دلار در هر ماه
حقوق و خدمات			
پشتیبانی جامعه	✓	✓	✓
پشتیبانی رسمی از طرف شرکت کیوت	✗	✗	✓
شما می‌توانید نرم‌افزارتان را به صورت خصوصی نزد خود نگه‌دارید.	✗	✗	✓
در صورتی که اپلیکیشن خود را به صورت لینک-پویا (Dynamic-link) کامپایل کرده باشد، می‌توانید آن را خصوصی برای خود حفظ کنید.	✓	✗	✓
بدون ارائه مکانیزم خاصی برای کتابخانه‌های کیوت (می‌توانید همیشه از کامپایل استاتیک) بهره‌مند شوید.	✗	✗	✓
نیازی برای ارائه کپی از مجوز و صراحةً از استفاده کیوت نست است.	✗	✗	✓
نیازی برای ارائه یک کپی (نسخه) از سورس گذ کیوت برای مشتری را ندارید.	✗	✗	✓

حقوق کامل برای تغییرات در کد منبع کیوت و سفارشی آن	✗	✗	✓
اجبار و تاکید بر (مدیریت حقوق دیجیتال)	<u>مشاهده جزئیات در LGPv3 مجوز</u>	<u>مشاهده جزئیات در LGPv3 مجوز</u>	✓
اجبار و تاکید بر (پیاده سازی پروتکل های نرم افزاری)	<u>مشاهده جزئیات در LGPv3 مجوز</u>	<u>مشاهده جزئیات در LGPv3 مجوز</u>	✓
ماژول ها			✓
Qt Essentials	✓	✓	✓
افزونه های عمومی کیوت	✓	✓	✓
Qt Charts	✓	✓	✓
Qt Data Visualization	✓	✓	✓
ابزارها			✓
محیط توسعه یکپارچهی نرم افزاری Qt با تمامی قابلیت ها	✓	✓	✓
ابزارهای مستند سازی	✓	✓	✓
ابزارهای سفارشی	✓	✓	✓

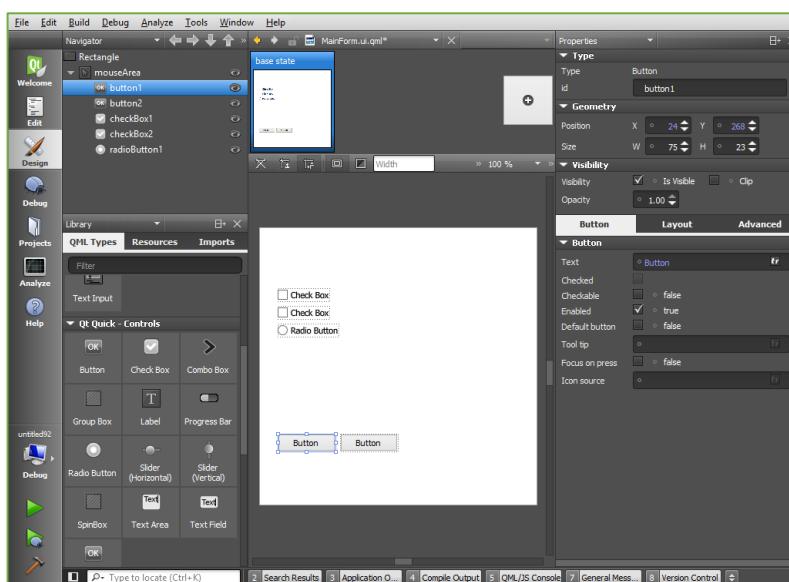
ابزار طراحی Qt Quick Designer	✓	✓	✓
ابزار Qt Quick Profiler	✓	✓	✓
ابزار توسعه و پیشرفت در Visual Studio	✓	✓	✓
ابزار Qt Quick 2D Renderer ، استفاده بدون OpenGL	✓	✓	✓
ابزار کامپایل Qt Quick Compiler	✓	✓	✓
ابزارها و راه حل‌های کیوت برای ساخت دستگاه‌های امید		✓	✓
ابزار Qt Virtual Keyboard	✗	✓	✓
خطایابی (دیباگینگ مستقیم) بر روی دستگاه	✗	✗	✓
ابزار Boot to Qt	✗	✗	✓
نصب و استقرار با یک کلیک در دستگاه و بستر هدف	✗	✗	✓
دستورالعمل‌های پروژه Yocto برای سفارشی سازی ایمیج‌های از پیش ساخته شده	✗	✗	✓

توجه داشته باشید که معمولاً شرکت کیوت بعضی از ویژگی‌های موجود را که مختص نسخه تجاری است بعضاً در نسخه‌های جدید تحت مجوزهای LGPV3 منتشر می‌کند. در این اواخر نمونه‌ای از آن‌ها Qt Quick Compiler بود که در نسخه ۵.۱۱.۰ منتشر شد.

قرداد نامگذاری در Qt: کلاس‌ها و کتابخانه‌ها در Qt با Q شروع می‌شوند به طور مثال QPushButton کلاس دکمه است و تقریباً اکثر نرم‌افزارهایی که با Qt توسعه داده می‌شوند از همین قانون نامگذاری برای نامگذاری نرم‌افزارشان استفاده می‌کنند؛ مانند QDevelop بنابراین برخی از برنامه‌نویسان مبتدی با دیدن کلمه Q در بین کدهای نوشته شده چنین می‌اندیشنند که Qt یک زبان است و هیچ ربطی به C++ ندارد! بنابراین باید به این موضوع اشاره‌ای داشته باشیم، که تمامی دستورات و نام‌گذاری‌ها بر پایه و اساس استانداردهای موجود در زبان C++ مدیریت می‌شوند.

محیط‌های توسعه Qt

محیط‌های توسعه مختلفی برای ابزار Qt وجود دارد که اکثراً توسعه برنامه‌نویسان علاقه‌مند به این ابزار ایجاد شده‌اند. جدیدترین محیط توسعه این ابزار Nokia Qt Creator نام دارد که قبلًاً توسعه Nokia ایجاد و سپس توسعه داده شد که در حال حاضر نیز تحت مدیریت شرکت کیوت توسعه و پشتیبانی می‌شود. Digia همچنین توسعه افزونه‌هایی که برای این کتابخانه نوشته شده است شما می‌توانید از محیط‌هایی مانند Visual Studio نیز استفاده نمایید.



در کنار محیط طراحی این محیط توسعه‌ی یکپارچه‌ی نرم‌افزار می‌توان به محیط منحصر بفرد ویرایشگر یا همان محیط کدنویسی (Editor) اشاره کرد که در نوع خود از قابلیت‌های بسیاری در سفارشی سازی برخوردار است.

```

File Edit Build Debug Analyze Tools Window Help
Projects pathstroke.cpp* PathStrokeRenderer::PathStrokeRenderer(QWidget*, bool) # Line: 389, Col: 25
application application.pro
Sources Headers
pathstroke shared Headers
Sources main.cpp mainwindow.cpp
Resources
pathstroke pathstroke.pro
shared Headers
Sources main.cpp pathstroke.cpp
Resources
Edit Design Debug Projects Analyze Help
pathstroke Debug
Type to locate (Ctrl+K) Search Results Application O... Compile Output QML/JS Cons... General Mess... Version Control

```

The screenshot shows the Qt Creator IDE interface. The left sidebar contains project navigation and tool icons. The main area is the code editor with the following code:

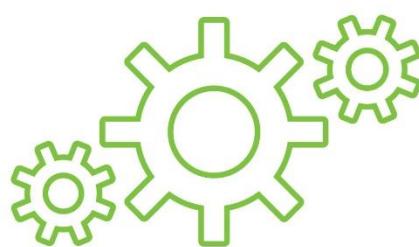
```

369 void PathStrokeWidget::setStyle( QStyle * style )
370 {
371     QWidget::setStyle(style);
372     if (m_controls != 0)
373     {
374         m_controls->setStyle(style);
375     }
376     QList<QWidget * > widgets = m_controls->findChildren<QWidget *>();
377     foreach (QWidget * w, widgets)
378         w->setStyle(style);
379 }
380
381 PathStrokeRenderer::PathStrokeRenderer(QWidget *parent, bool smallScreen)
382     : ArthurFrame(parent)
383 {
384     m_smallScreen = smallScreen;
385     mPointSize = 10;
386     m_activePoint = -1;
387     m_capStyle = Qt::FlatCap;
388     m_joinStyle = Qt::BevelJoin;
389     m_pathMode = CurBeginNativeGesture intBeginNativeGesture
390     m_penWidth = 1; BevelJoin
391     m_penStyle = Qt::BottomEdge
392     m_wasAnimated = true;
393     setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
394    setAttribute(Qt::WA_AcceptTouchEvents);
395 }
396
397 void PathStrokeRenderer::paint(QPainter *painter)
398 {
399     if (m_points.isEmpty())
400         initializePoints();
401
402     painter->setRenderHint(QPainter::Antialiasing);
403
404     QPalette pal = palette();
405     painter->setPen(Qt::NoPen);
406 }
407

```

ویژگی‌های Qt

همانطور که در توضیحات تقریباً اشاره شده است کیوت به عنوان یک کتابخانه قدرتمند ویژگی‌های فراتر از ملزومات را در خود جای داده است بنابراین کیوت فراتر از یک کتابخانه همراه با افزونه‌ها و مازول‌های خاص خود به کمک شما می‌آید؛ برای مثال کلاس‌ها و توابع پیشفرض همراه با کیوت برای تمامی سیستم‌عامل‌ها منتشر می‌شود.



در کل دو نوع مازول بندی در این کتابخانه وجود دارد که به صورت "مازول‌های ضروری" و مازول‌های اضافی و انحصاری تعریف می‌شوند که در جداول زیر به آنها اشاره می‌کنیم.

توضیحات	ماژول
کلاس است اصلی و هسته‌ی کتابخانه که توسط دیگر ماژول‌ها مورد استفاده قرار می‌گیرد.	Qt Core
کلاس پایه برای رابطکاربری که شامل OpenGL است.	Qt GUI
کلاس‌های صوتی و تصویری، رادیو و قابلیت‌های دوربین.	Qt Multimedia
کلاس‌های مبتنی بر ویجت برای پیاده سازی قابلیت‌های چند رسانه‌ای.	Qt Multimedia Widgets
کلاسی برای ایجاد برنامه‌نویسی شبکه‌ای راحت‌تر و قابل حمل‌تر است.	Qt Network
کلاس‌هایی که شامل QML و JavaScript هستند.	Qt QML
چهارچوب اعلانی برای ساخت برنامه‌های کاربردی بسیار پویا با رابطه‌ای کاربری سفارشی و مدرن.	Qt Quick
قابل استفاده مجدد در Qt تحت فناوری کیوت کوئیک و ایجاد رابطکاربری (UI) بر اساس کنترل برای ایجاد رابطکاربری دسکتاپ به سبک کلاسیک.	Qt Quick Controls
انواع دیالوگ‌ها برای استفاده در برنامه‌های کاربردی.	Qt Quick Dialogs
طرح بندی آیتم‌های مورد استفاده به بر اساس فناوری 2.0	Qt Quick Layouts
کلاسی برای یکپارچه سازی پایگاه داده با استفاده از SQL.	Qt SQL
کلاسی برای آزمایش واحد برنامه‌های کاربردی در کتابخانه Qt.	Qt Test
کلاس‌هایی برای ایجاد وب‌کیت ۲ و همچنین استفاده از API‌های QML	Qt WebKit
کلاس‌هایی برای Webkit و Widgets مبتنی بر کیوت 4.x	Qt WebKit Widgets
کلاس‌هایی برای توسعه رابطکاربری توسط ویجت‌های C++	Qt Widgets

به عنوان مثال برای کار با **Bluetooth** کدهای بسیاری می‌توان نوشت. اما در این میان کیوت برای راحتی کار ماژولی با نام **Qt Bluetooth** را فراهم کرده است تا بتوانید به راحتی برای ارتباط با سخت افزار و بولتوث پردازید.

برخی از مازول‌های موجود در کتابخانه به صورت زیر مشخص شده است:

توضیحات	سکوی مورد استفاده	سکوی توسعه	ماژول
کلاس برای برنامه‌های کاربردی است که استفاده از اکتیو ایکس و COM		مم	Active Qt
یک راه حل مدیریت به عنوان یک سرویس برای سهولت توسعه بخش مدیریت برای برنامه‌های کاربردی متصل شده و مبتنی بر داده.	مم	مم	Enginio
رابطهای برنامه کاربردی پلت فرم خاص برای اندروید را فراهم می‌کند.	Android	مم	Qt Android Extras
دسترسی به سخت افزار بلوتوث را فراهم می‌کند.	Android, Linux, BlackBerry	مم	Qt Bluetooth
کلاس برای نوشتن برنامه‌های چند رشته ای بدون نیاز به برنامه‌نویسی از سطح پایین		مم	Qt Concurrent
کلاس‌های برای برقراری ارتباط بین فرایند D-BUS از طریق پروتکل‌های		مم	Qt D-Bus
تأثیرات گرافیکی برای استفاده در کیوت کویک ۲,۰		مم	Qt Graphical Effects
افزونه برای فرمات‌های تصویری اضافی: WBMP, TGA, MNG, TIFF		مم	Qt Image Formats
رابطهای برنامه کاربردی پلت فرم خاص برای سیستم‌عامل OSX را فراهم می‌کند.	OS X	مم	Qt Mac Extras

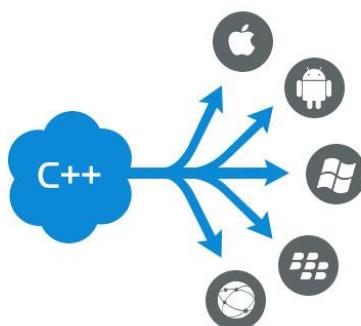
دسترسی به ارتباطات نزدیک میدان سخت افزارهای مجهز به (NFC) را فراهم می‌کند.	<u>Blackberry</u>	مم	Qt NFC
کلاس پشتیبانی از موتور OpenGL توجه برنامه‌های که توسط نسخه ۴ نوشته شده اند پشتیبانی می‌شوند بنابراین برای استفاده در کدهای جدید از کلاس استفاده در <u>QtGUI</u> در QOpenGL استفاده کنید.		مم	Qt OpenGL
کلاسی برای کپسوله‌سازی اطلاعات خاص را در انواع سیستم‌عامل‌ها فراهم می‌کند.		مم	Qt Platform Headers
دسترسی به کلاس‌های موقعیت، ماهواره‌ای و نظارت بر منطقه را فراهم می‌کند.		مم	Qt Positioning
کلاسی که امکان چاپ را به صورت قابل حمل‌تر فراهم می‌کند.		مم	Qt Print Support
کلاس Qt Declarative برای نسخه ۴ فراهم شده است که برای استفاده در Qt Quick و در نسخه‌های جدید نیز پشتیبانی می‌شود.		مم	Qt Declarative
کلاسی برای فراهم ساختن برنامه‌های کاربردی اسکریپتی است که در نسخه ۴ نیز قابل پشتیبانی است بنابراین برای استفاده از کلاس QJS استفاده نمایید.		مم	Qt Script

اجزای اضافی برای برنامه‌های کاربردی که با استفاده از Qt Script نوشته می‌شوند.		مه	Qt Script Tools
دسترسی به سخت‌افزار سنسور و تشخیص اشاره و حرکت را فراهم می‌کند.	Android , Blackberry , Qt for iOS و WinRT .	مه	Qt Sensors
دسترسی به سخت‌افزار و پورت سریال مجازی را فراهم می‌کند.	Windows , Linux و OS X .	مه	Qt Serial Port
کلاس برای نمایش محتويات فایل‌های SVG پشتیبانی از زیر مجموعه‌ای از SVG 1.2 استاندارد است.		مه	Qt SVG
دسترسی به اشیاء QObject یا QML را از طرف HTML توسط کاربر را فراهم می‌کند.	مه	مه	Qt WebChannel
امکان دسترسی و اجرای API‌های برنامه‌های کاربردی در موتور مرورگر کرومیوم را فراهم می‌کند.	Windows , Linux و OS X .	مه	Qt WebEngine
امکان دسترسی و اجرای API‌های C++ را در برنامه‌های کاربردی در موتور مرورگر کرومیوم را فراهم می‌کند.	Windows , Linux و OS X .	مه	Qt WebEngine Widgets
ارتباطات WebSocket سازگار با RFC 6455 را فراهم می‌کند.	مه	مه	Qt WebSockets
امکان ارتباط با API‌های خاص بستر ویندوز را فراهم می‌کند.	Windows	مه	Qt Windows Extras
امکان ارتباط با API‌های خاص بستر X11 را فراهم می‌کند.	Linux/X11	مه	Qt X11 Extras

پیاده سازی C++ از SAX و DOM را که مختص است را فراهم XML			Qt XML
پشتیبانی از XQuery و XPath ، XSLT و اعتبار سنجی مدل XML			Qt XML Patterns

در صورتی که شما از ابزار **qmake** جهت ساخت (بیلد) پروژه استفاده می‌کنید کافی است کد زیر را در فایل **.pro** قرار دهید، در این حالت به صورت خودکار ماژول **Qt GUI** به پروژه‌ی شما افزوده خواهد شد.

```
QT += gui
```



طراحی، کدنویسی، اشکال‌زدایی و گسترش سریع

پشتیبانی از انواع سیستم‌عامل‌ها

به طور طبیعی برنامه‌نویسی با C++ در تمامی سیستم‌عامل‌ها و تجهیزات امکان‌پذیر است، حال که برای طراحی و پیاده سازی یک برنامه در محیط Qt فراهم شده است به قابلیت‌های این کتابخانه در این زمینه می‌پردازیم.

باید توجه داشته باشیم که، بدون هیچ بازنویسی و طراحی مجدد شما می‌توانید برنامه‌ای را که طراحی کردید در نسخه‌های مختلف رومیزی، موبایل و سیستم‌های تعییه شده (Embedded) از آن‌ها اجرا کنید و این یکی از بهترین امتیازها برای یک برنامه‌نویس است که یک بار کد نوشته و در سیستم‌های مختلف بدون بازنویسی آن را اجرا کند.

معماری قابل پشتیبانی: معماری‌های ۳۲ و ۶۴ بیتی به صورت کامل در این کتابخانه پشتیبانی می‌شوند که معمولاً در هر بستر تحت **GCC** می‌توان آن را ساخت و در این میان باید به این نکته اشاره کنیم که قابلیت

پشتیبانی از (Qt Quick) در فناوری Open GL (ES) 2.0, DirectX 9 (with ANGLE) نیز پشتیبانی می‌شود که علاوه بر آن در حالت Widgets استفاده از شتاب دهنده‌های سخت افزاری نیاز نیست. بسترهاي که پشتیبانی می‌شوند به صورت زیر است.

نسخه‌های دسکتاپی:

- ویندوز (Windows)
- لینوکس (Linux)
- مکیناش (Mac OS X)

نسخه‌های Embedded:

- ویندوز (Windows)
- لینوکس (Linux) تمامی سیستم‌عامل‌های بر پایه Unix
- اندروید(Android)
- کیو ان ایکس (QNX)
- وی ایکس ورکس (VxWorks)

نسخه‌های موبایلی:

- ویندوز فون (Windows Phone)
- آی او اس (iOS)
- اندروید(Android)

از دیگر سیستم‌عامل‌ها می‌توان به sailfishos و Tizen هم اشاره کرد.

در این جدول لیستی از انواع سیستم‌عامل‌ها و مترجم‌های سازگار با این کتابخانه موجود است.

بستر	مترجم
Ubuntu Linux 14.04, X11 (64-bit)	As provided by Ubuntu
RedHat 6.6, X11 (64-bit)	As provided by Red Hat
openSUSE 13.1, X11 (64-bit)	As provided by SUSE
Microsoft Windows 7 (32-bit)	MSVC 2010 SP1

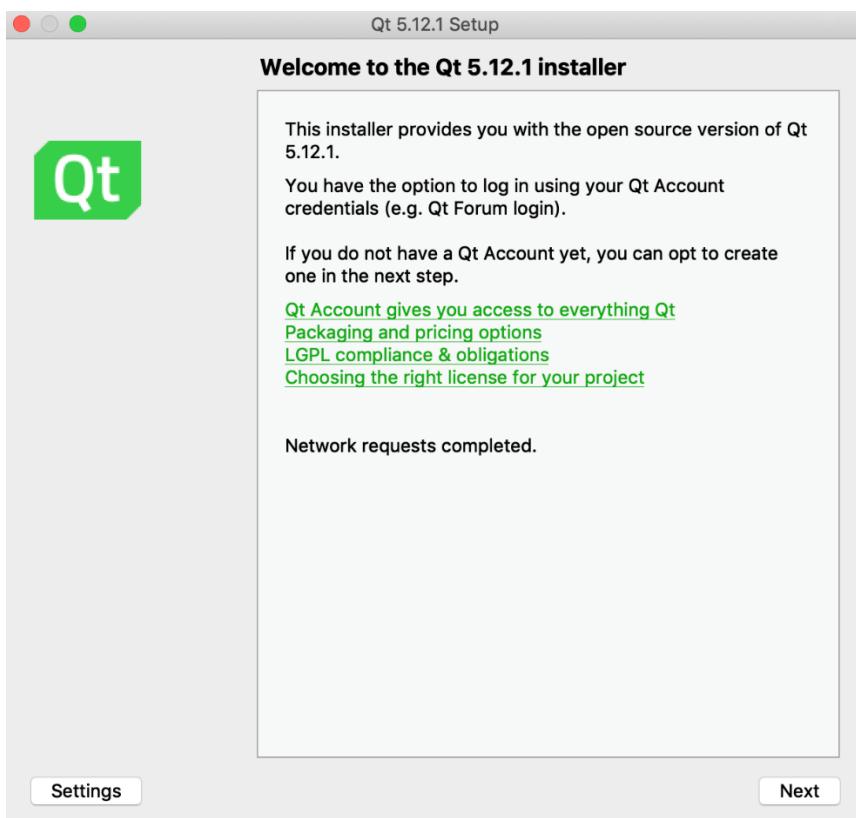
پستر	مترجم
Microsoft Windows 7 (32-bit)	MinGW-builds gcc 4.9.1 (32-bit)
Microsoft Windows 8.1, 10 (32-bit and 64-bit)	MSVC 2012, 2015, 2017 SP2
Microsoft Windows 8.1 (32-bit and 64-bit)	MSVC 2013 SP3
Microsoft Windows Phone 8.1 (64-bit)	MSVC 2013 SP3
Microsoft Windows Runtime 8.1 (64-bit)	MSVC 2013 SP3
Apple OS X 10.8 "Mountain Lion", Cocoa (64-bit)	Clang as provided by Apple
Apple OS X 10.9 "Mavericks", Cocoa (64-bit)	Clang as provided by Apple
Apple OS X 10.10 "Yosemite", Cocoa (64-bit)	Clang as provided by Apple
Apple iOS 8 (64-bit)	Clang as provided by Apple
Google Android 4.4	As provided by Google (Android NDK Revision 10c)

نصب و پیکربندی محیط Qt

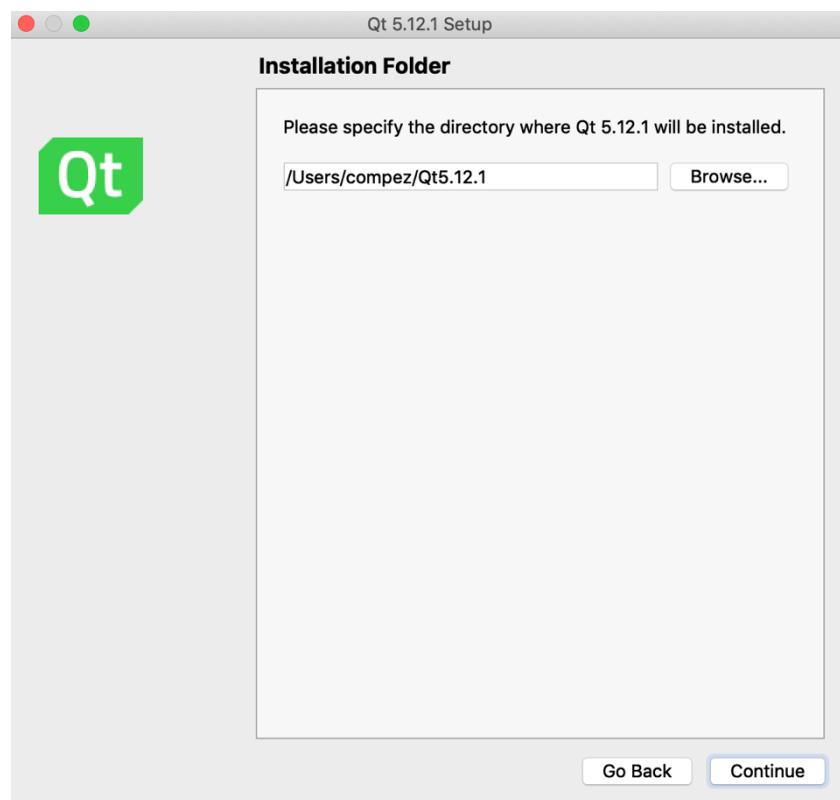
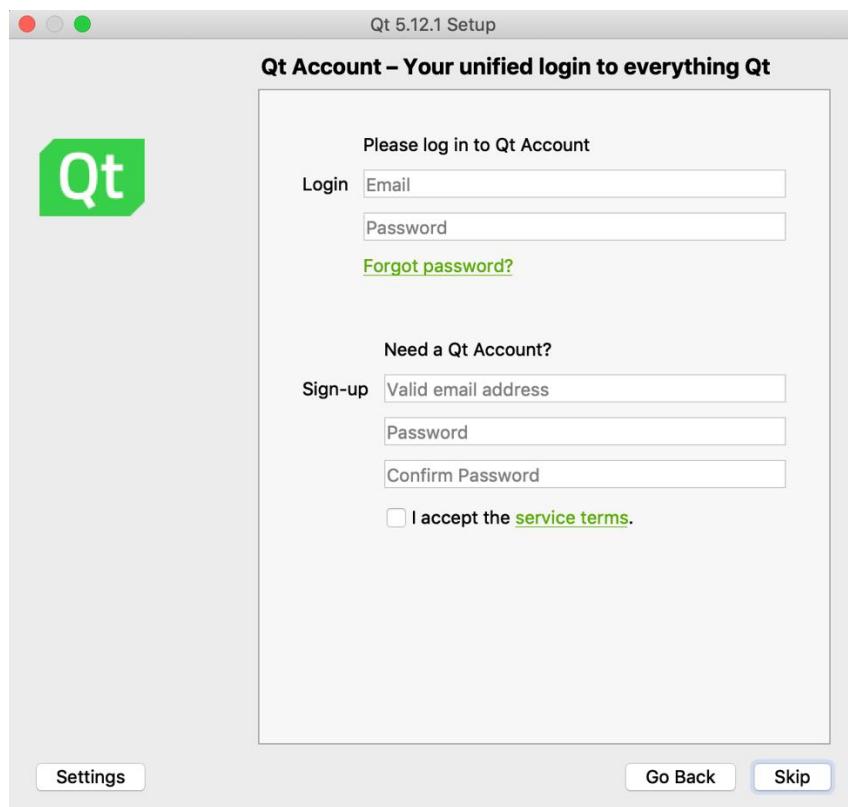
همانطور که می‌دانید معمولاً برای استفاده کتابخانه‌های C++ نیاز است تا آن‌ها را کامپایل و سپس در محیط توسعه خود آن‌ها را اضافه کنیم؛ این روش در این کتابخانه نیز صدق می‌کند با این تفاوت که توسعه‌دهنده رسمی آن نسخه‌های کامپایل شده آن را در اختیار کاربران قرار می‌دهند.

بنابراین نیازی نیست این کتابخانه را کامپایل نمایید مگر اینکه در موارد خاص ترجیح دهید با پیکربندی خاصی از آن خروجی بگیریم که در این صورت از سایت رسمی آن می‌توانید سورس (منبع) آخرین نسخه و حتی نسخه‌های پیشین آن را دریافت کنید.

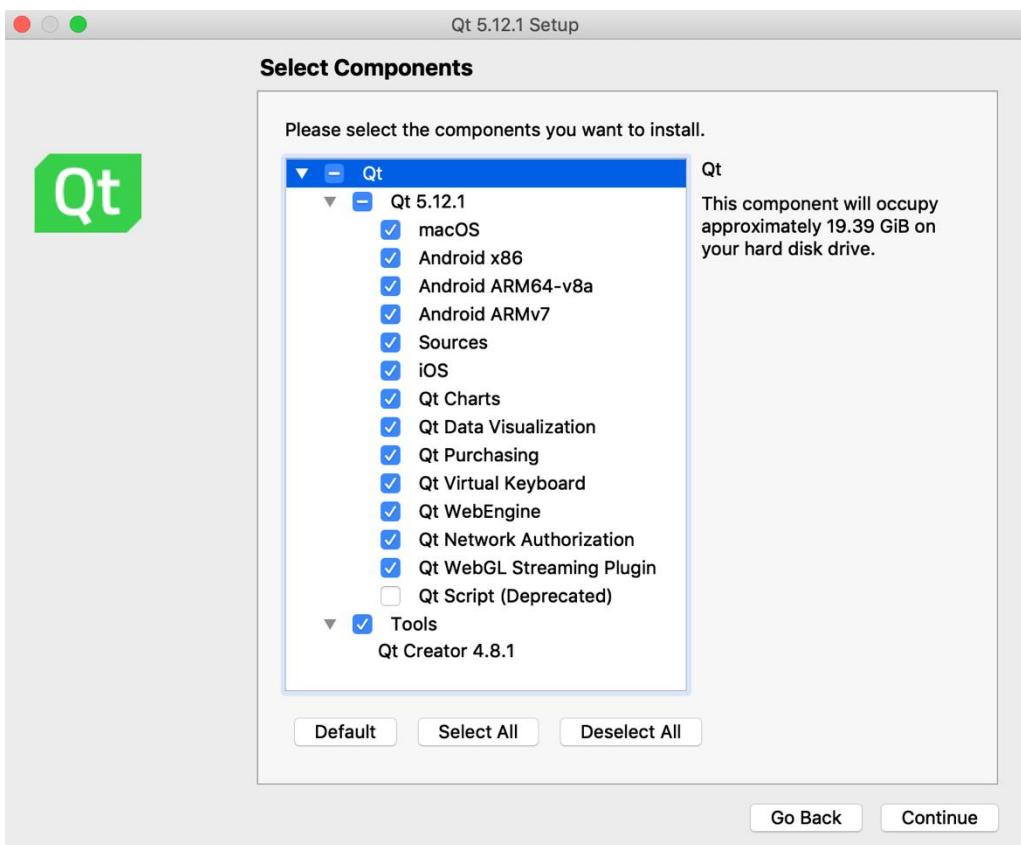
مرحله‌ی اولیه جهت نصب کتابخانه به صورت زیر خواهد بود:



در این بخش گزینه‌ی **Next** را زده و وارد مرحله بعد خواهیم شد، اگر نام کاربری و رمز عبور کیوت خود را دارید می‌توانید وارد کنید، در غیر این صورت نیازی به آن نیست.



بعد از مشخص کردن مسیر نصب کیوت به مهمترین بخش آن می‌رسیم، این بخش نیاز به توضیحاتی دارد که در ادامه برای مطالع شدن از آن‌ها اشاره شده است.

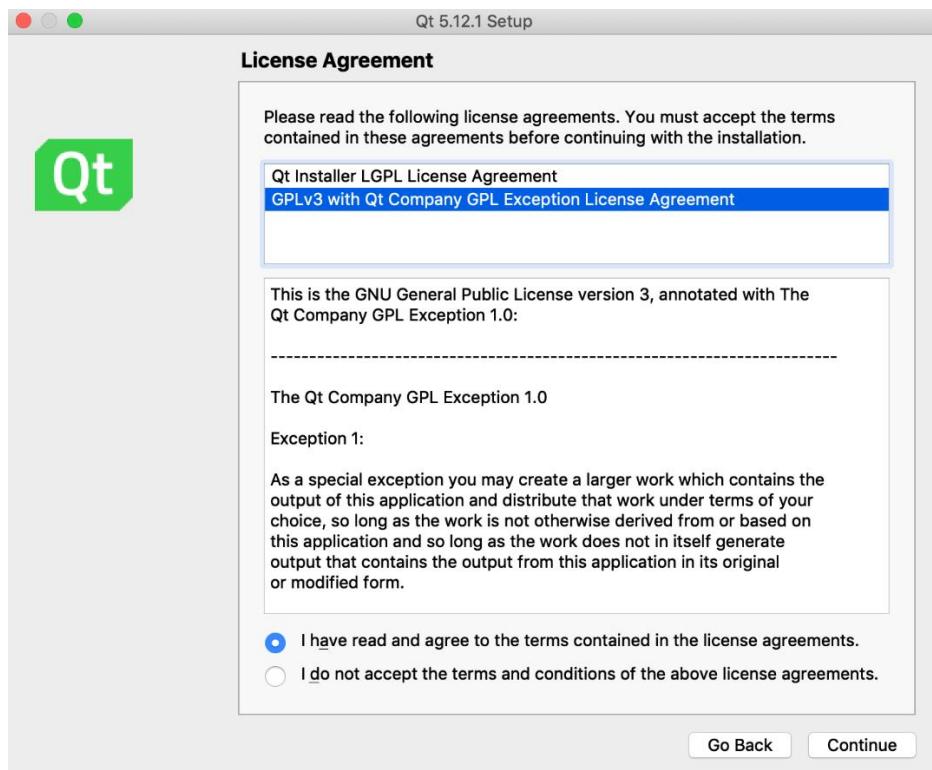


*توجه داشته باشید که در بسترهای ویندوز و لینوکس گزینه‌های مربوط به iOS موجود نمی‌باشند. بنابراین، مهمترین سوال در این مرحله است که چه گزینه‌هایی را باید برای نصب انتخاب شوند؟ در این بخش تمامی گزینه‌ها را انتخاب کنید به جز گزینه‌هایی که منسوخ شده‌اند که با واژه‌ی (Deprecated) مشخص شده‌اند و منظور از آن‌ها این است که در نسخه‌های بعدی حذف خواهند شد.

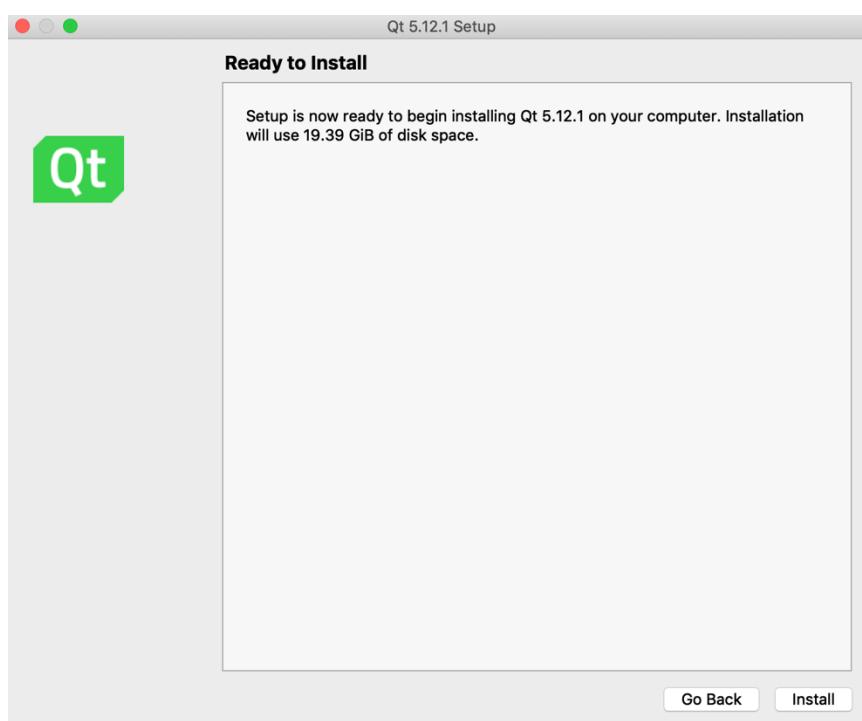
همچنین گزینه‌هایی که مقابل آن‌ها TP نوشته شده است مخفف Technology Preview است به معنی اینکه این مازول یا افزونه به صورت آزمایشی فعلاً در این پکیج قرار گرفته اما نهایی نشده است. در بخش Tools تمامی گزینه‌ها را انتخاب کنید همه‌ی آن‌ها نیاز است. همچنین برای اینکه بتوانید از بانک اطلاعاتی (Database) QMySQL استفاده کنید باید گزینه‌ی Sources را انتخاب کنید تا بعداً امکان ساخت این مازول فراهم شود.

مواردی که شامل گزینه‌های UWP هستند به خاطر آن است که شما بتوانید تحت کیوت برنامه‌های Universal Windows Platform را استقرار و اجرا کنید. همچنین جهت امکان تولید برنامه‌های اندروید و ویندوز فون گزینه‌های Android ARMv7 و UWP ARMv7 نیاز هستند. دقت کنید که گزینه‌ی x86 برای اندروید معمولاً برای نسخه‌ی مجازی دسکتاپ مورد استفاده قرار می‌گیرد.

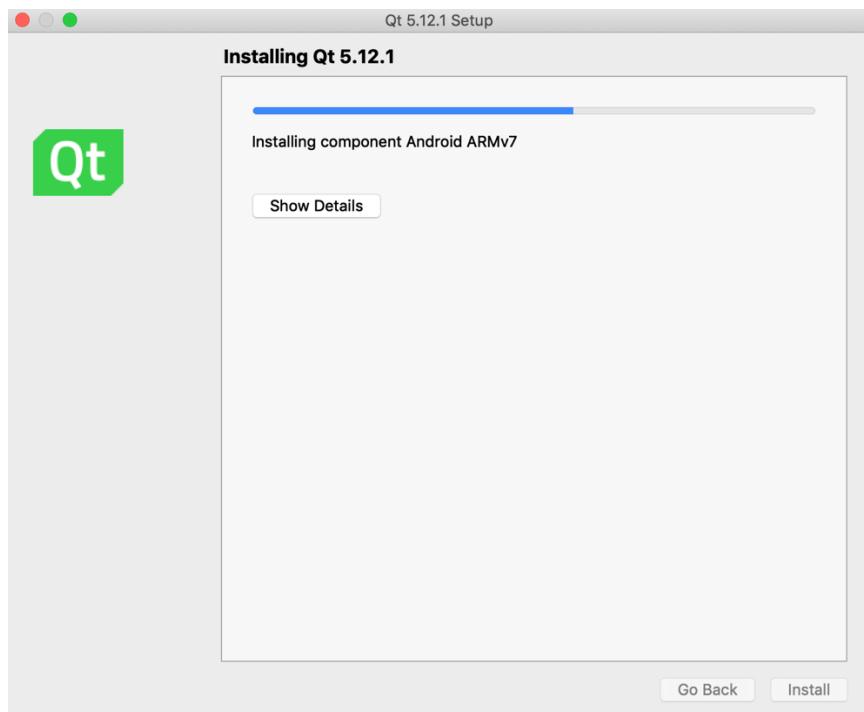
نکته: در بسترهای لینوکس و مک اواس گزینه‌های مرتبط با MSVC وجود ندارد. تنها با این تفاوت که در بستر iOS نیز موجود هستند و شما می‌توانید با انتخاب گزینه‌ی iOS آن را به محیط توسعه خود اضافه کنید. در نهایت شرایط و مجوزهای کیوت را تایید کنید که در ادامه آمده است:



توجه داشته باشید که حتماً فضای کافی جهت نصب و پیاده سازی این کتابخانه را داشته باشید، برای مثال نسخه‌ی ۵.۱۲.۱ کیوت بر روی بستر مک اواس فضایی به مقدار ۱۹.۳۹ گیگابایت نیاز خواهد داشت.



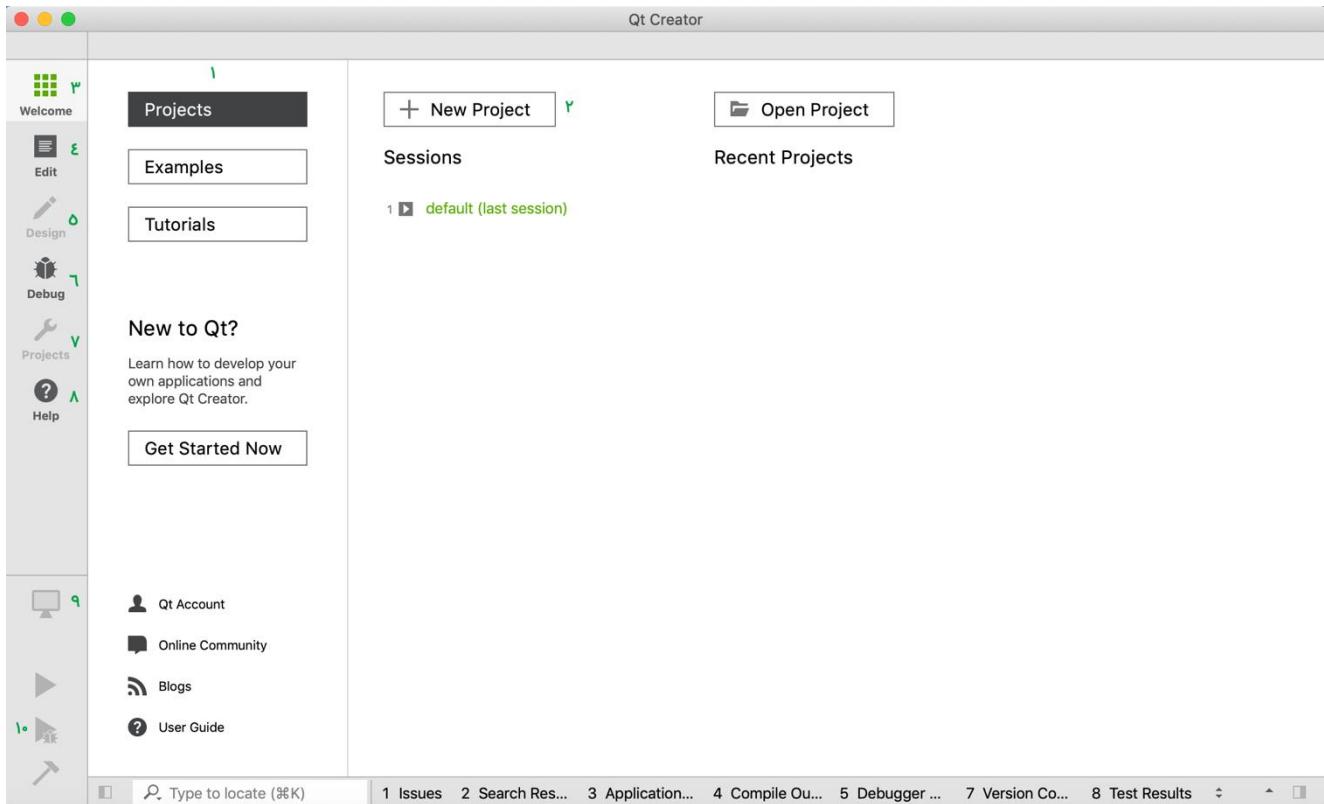
در نهایت برای شروع نصب کیوت با تنظیمات مربوطه کافی است بر روی install کلیک کنید، بد نیست تا نصب کیوت را با یک فنجان قهوه یا چای از نوع دَمنوش تجربه کنید ☺



در نهایت بعد از نصب کیوت محیط توسعه و کتابخانه بر روی بستر توسعه‌ی شما مستقر شده است.



حال محیط توسعه‌ی کیوت آماده استفاده است که با اجرای آن به صورت زیر نمایان خواهد شد.

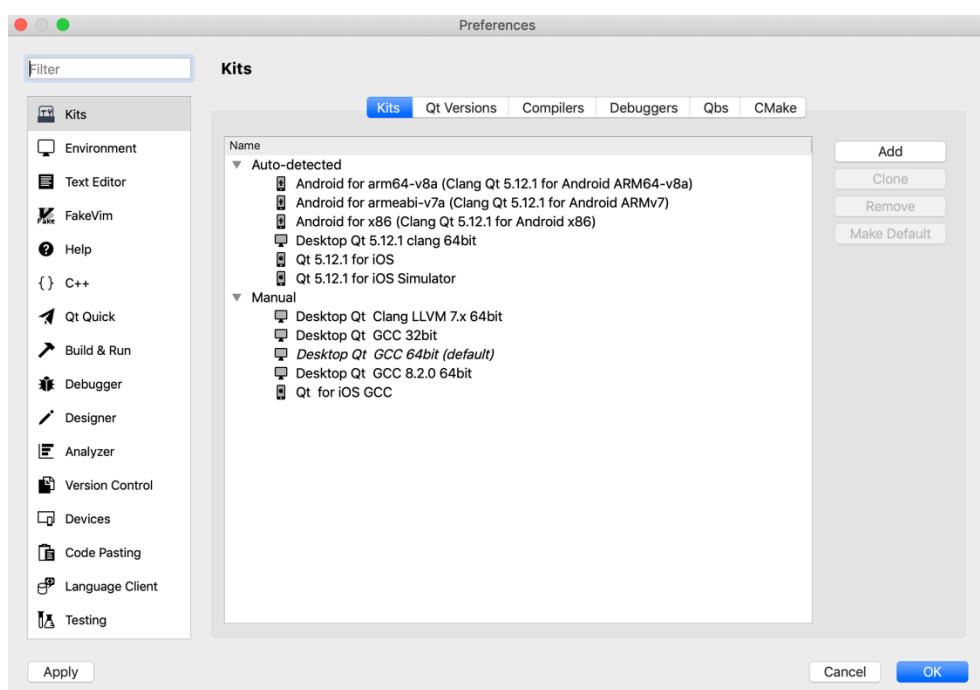


طبق تصاویر خواهید دید که تقریباً ۱۰ بخش مختلفی وجود دارد که هر کدام به صورت زیر دارای وظایف و کاربردهایی هستند.

۱. منوی اصلی، نوار ابزارها (شامل منوی‌های استانداردی است که اکثرا در هر محیط برنامه‌نویسی یافت می‌شود) در محیط مک با حرکت ماوس به بالاترین نقطه‌ی محیط ظاهر خواهند شد و در محیط ویندوز منوها به صورت پیش‌فرض نمایان هستند.
۲. شامل گزینه **Open project** و **New project** جهت دسترسی و ایجاد پروژه‌ها و همچنین اجازه دسترسی برای شروع کار طبق آخرین جلسات کاری و جلسات ذخیره شده به صورت پیش‌فرض را در اختیار قرار می‌دهد.
۳. قسمت ویرایش یا همان **Edit** در صورت وجود پروژه شامل محتویات پروژه شما می‌شود و آن را به صورت ساختار درختی نشان خواهد داد که در کل امکان انتخاب و ویرایش فایل‌های موجود در پروژه را می‌دهد.
۴. بخش طراحی و **Design** شامل محیط طراحی، ابزارها و موارد مرتبط با طراحی پروژه است.
۵. بخش **Debug** مربوط به عملیات دیباگینگ (اشکال‌زدایی) است.

۶. این بخش مربوط به تنظیمات پروژه مثل تنظیمان نوع ساخت (Build) و غیره است.
۷. در صورتی که به صورت همزمان در حال توسعه‌ی چندین پروژه هستید، این بخش امکان انتخاب پروژه را برای شما فراهم می‌کند.
۸. یکی از بهترین قسمت‌های پرکاربرد برای مبتدیان بخش Help خواهد بود که می‌توانید با مراجعه با این قسمت و جستجوی دستورات و عبارات مورد نظر آموزشی را در رابطه با موضوع مورد نظر دریافت نمایید.
۹. اجرای برنامه همراه با کامپایل، در صورتی که برنامه بیلد (ساخته) شده باشد برنامه فقط اجرا خواهد شد در غیر اینصورت بر اساس تنظیمات پیش فرض کامپایل و سپس اجرا خواهد شد.
۱۰. نوع مترجم (مترجم) و خروجی را تنظیم می‌کند و عملیاتی مثل Build، Compile، Run، Rebuild و ... را انجام می‌دهد.
۱۱. مورد آخر شامل نوار وضعیت‌های پروژ است که شامل موارد Output و ... برای مشاهده انواع رخدادها و خروجی‌ها به صورت کنسول است.

تمامی محیط‌های برنامه‌نویسی شامل گزینه‌های زیادی است که در پروژه‌ها و اهداف خروجی تنظیمات را اعمال خواهیم کرد که در نهایت تمامی این موارد را توضیح خواهیم داد. به عنوان مثال تصویر نشان دهنده این است که چه نوع مترجم و کیت‌هایی برای ساخت پروژه‌ی شما فعال است.



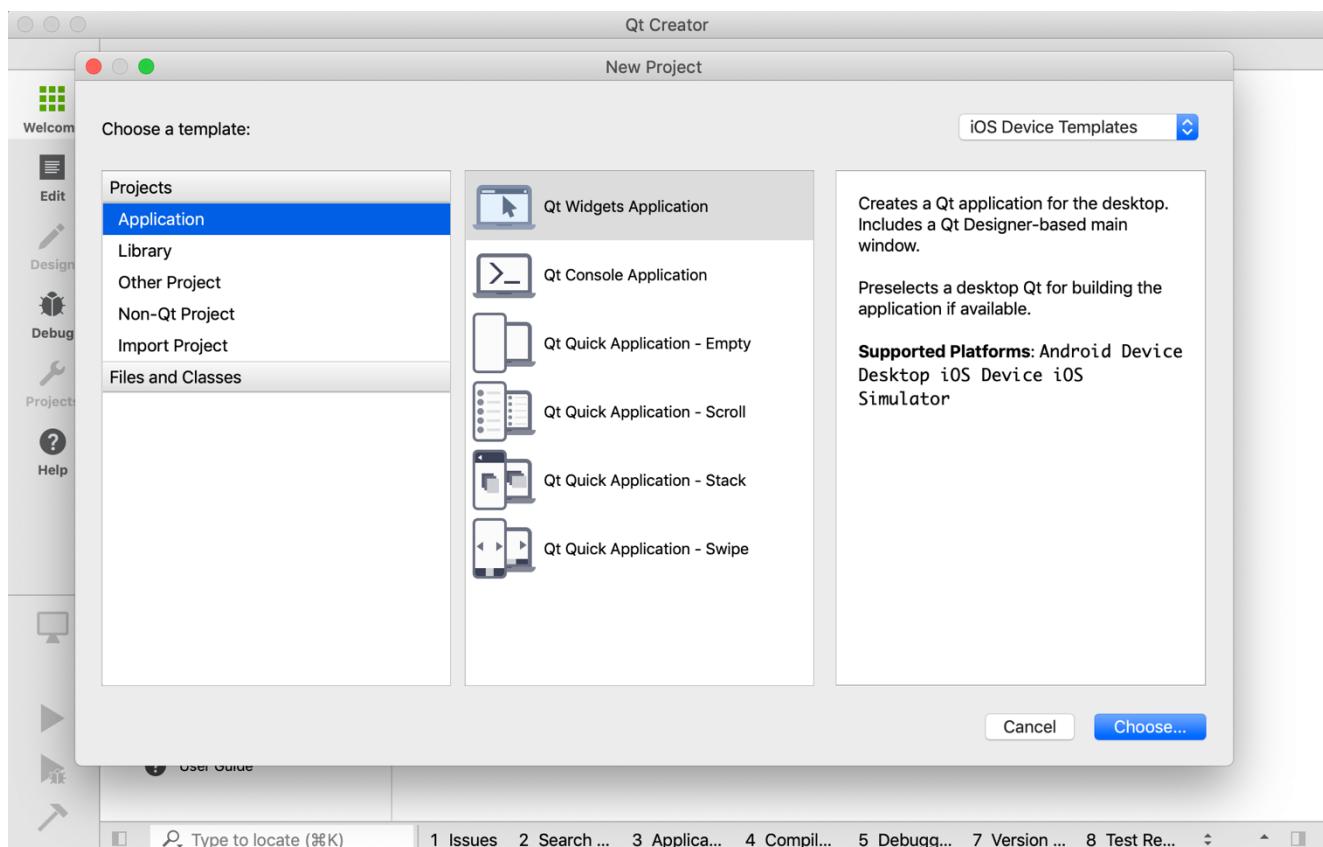
انواع پروژه‌ها

در رابطه با انواع پروژه‌ها در کیوت می‌توان به نوع بستر و ساختار آن اشاره کرد، بنابراین قبل از ایجاد آن می‌توانید نوع آن را مشخص نمایید که معمولاً شامل Qt Quick Application و Qt Widgets Application است که Qt Canvas 3D Application و Qt Console Application، Qt Quick Controls Application ما در این کتاب تنها به دو نوع آن یعنی Qt Console Application و Qt Widgets Application اشاره خواهیم کرد.

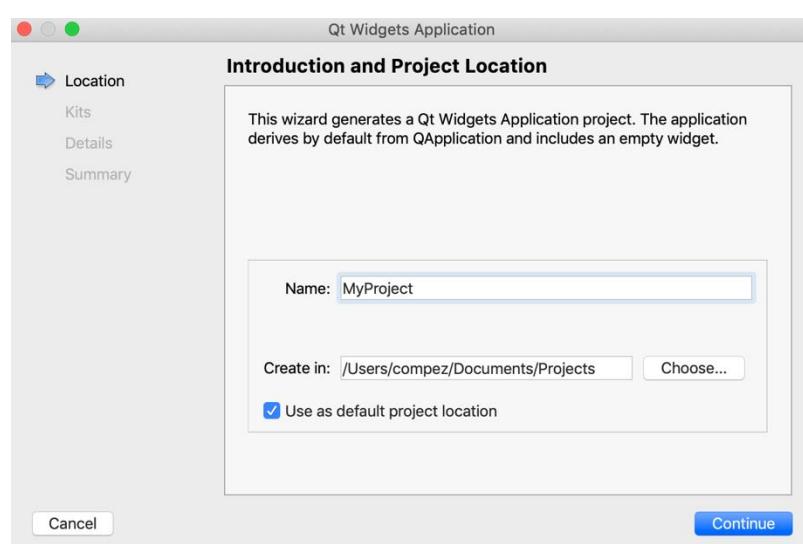
- این نوع پروژه‌ها بر اساس **Widgets** تولید می‌شوند و به طور کلی کدنویسی در محیط C++ همراه با Widgets پیاده سازی خواهد شد.
- این نوع پروژه‌ها بر اساس فناوری Quick و با استفاده از زبان QML با C++: ترکیب می‌شوند (آموزش‌های مربوط به این محیط در کتاب پیشرفته موجود هستند).
- این نوع پروژه‌ها بر اساس فناوری Quick و با استفاده از زبان QML با C++: ترکیب می‌شوند که به صورت پیشفرض مازول‌های Controls در آن موجود است.
- این نوع پروژه تنها در محیط کنسول اجازه کدنویسی را می‌دهد و خالی از هر گونه محیط GUI خواهد بود بنابراین به صورت پیش فرض مازول Widgets در آن غیر فعال است.
- این نوع پروژه‌ها بر پایه‌ی فناوری 3D پیاده سازی شده و توسط آن می‌توانید پروژه‌های ۳ بعدی را ایجاد نمایید.

ایجاد پروژه

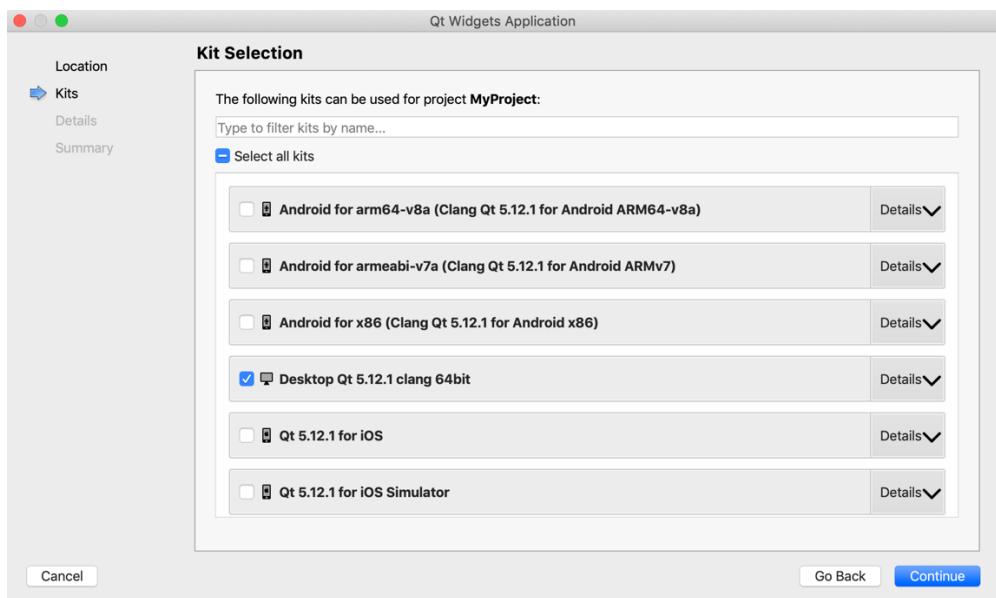
برای شروع به منوی **File** و گزینه **New File or Project** مراجعه کنید. حالا در این قسمت شما انواع پروژه‌ها و فایل‌هایی که می‌توانید توسط Qt ایجاد کنید با توجه به نسخه‌ای که نصب کردیدن فعال و قابل انتخاب است که ما در این آموزش از پروژه **Qt Widgets Application** استفاده می‌کنیم.



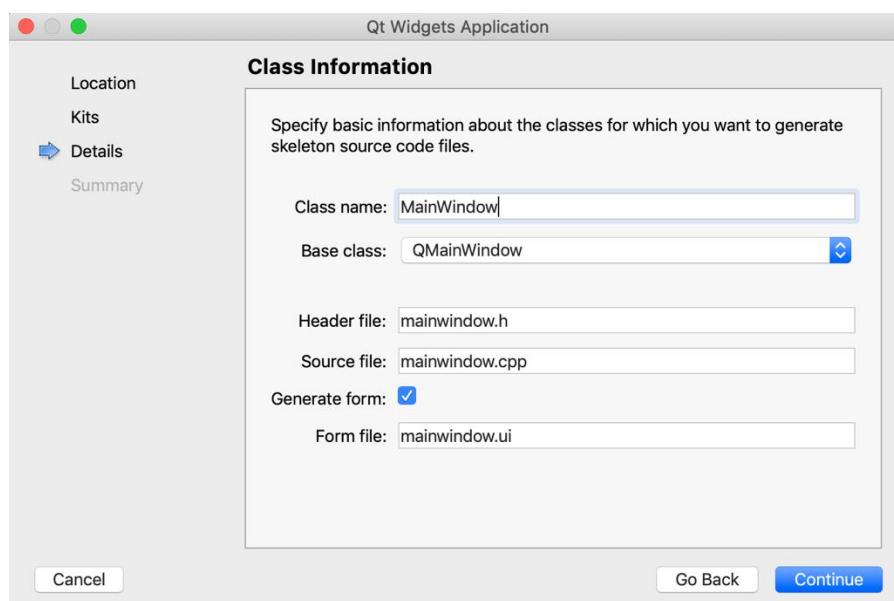
حال با ادامه این مرحله به صورت زیر نام پروژه و مسیری که مایل هستید پروژه در آن مکان ذخیره شود را انتخاب کنید:



در مرحله بعدی نوع مترجم در صورت نصب بودن تایید و انتخاب می‌شود به صورت زیر:



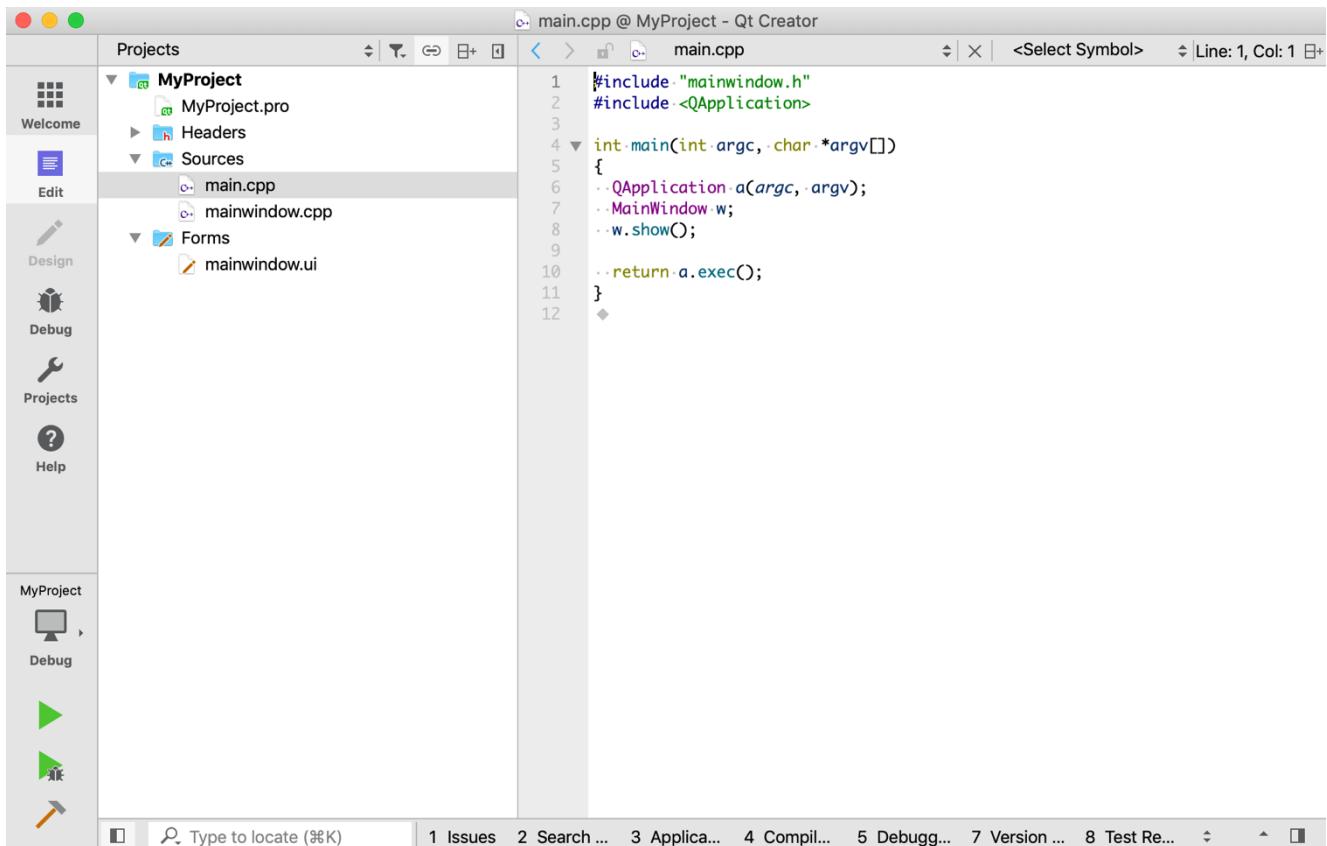
در این مرحله شما می‌توانید نوع کلاس و همچنین سرآیند (Header) آن را مشخص کنید طبق تصویر زیر ادامه دهید:



در ادامه، اگر شما نیاز به مدیریت نسخه‌ی پروژه داشته باشید، می‌توانید آن را با انتخاب گزینه‌ی Git اعمال کنید. در این آموزش نیازی به آن نخواهیم داشت. بنابراین تنظیم پروژه به صورت زیر به اتمام خواهد رسید.

تا کنون ما تنها یک پروژه از نوع **Qt Widgets** ایجاد کرده‌ایم که شامل **Main Window** و کلاس و سرآیندهای استاندارد برای شروع کار و برنامه‌نویسی است. در پروژه‌های Qt چند نوع فایل پروژه‌ای موجود هستند که با

پسوندهای .pri و .pro. که هر دوی این فایل‌ها توسط Qt قابل شناسایی هستند و به عنوان فایل اصلی پروژه شما در نظر گرفته می‌شوند.



اگر بر روی فایل MyProject.pro دو بار کلیک کنید کدهایی را به صورت زیر خواهید دید:

```
#
# Project created by QtCreator 2019-02-23T11:24:13
#
QT      += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = MyProject
TEMPLATE = app

# The following define makes your compiler emit warnings if you use
# any feature of Qt which has been marked as deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if you use deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version of
# Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000      # disables all the APIs
deprecated before Qt 6.0.0
```

```

CONFIG += c++11

SOURCES += \
    main.cpp \
    mainwindow.cpp

HEADERS += \
    mainwindow.h

FORMS += \
    mainwindow.ui

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

```

در این بخش محتوای موجود در فایل pro. مهمترین قسمت پروژه می‌تواند باشد که در فراخوانی فایل‌ها و مراجع (References) مهم است؛ برای مثال اگر نیاز است از بانک اطلاعاتی و دستورات SQL استفاده کنیم ابتدا باید ماثول آن در این فایل فراخوانی شود که توضیحاتی در رابطه با تاریخ ساخت پروژه توسط محیط Qt آورده شده است که در این پروژه به صورت زیر آمده است:

```

#-----
#
# Project created by QtCreator 2019-02-23T11:24:13
#
#-----
```

در کیوت برای حذف یا افزودن ماثول پیشفرض و یا سفارشی از متغیر QT استفاده می‌شود که مثال در این پروژه به صورت زیر آمده است:

```
QT += core gui
```

عملگرهای += به منظور افزوده شدن ماثول‌های مربوطه است. برای مثال در اینجا ماثول‌های core و gui در پروژه مربوطه افزوده شده‌اند. در صورتی که نیاز باشد ماثول خاصی را حذف کنیم می‌توانید دستور مربوطه را با عملگرهای -= اعمال کنید.

مثال:

```
QT += core
QT -= gui
```

در ادامه اگر نیاز باشد ماثول خاصی را که نسبت به نسخه‌های پیشین کیوت فرق داردن (به صورت شرطی اعمال کنید) آن به صورت دستور زیر می‌تواند در دسترس باشد.

```
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
```

همچنین، برای تعیین نوع پروژه و هدف ساخت گزینه‌های Target و Template مقدار دهی می‌شوند که در زیر آمده است:

```
TARGET = MyProjectName  
TEMPLATE = app
```

نکته: دقت کنید که مقدار TARGET بهتر است لاتین باشد (نامی که در پورسنه‌ی برنامه‌های اجرایی توسط سیستم‌عامل قابل شناسایی خواهد بود).

در صورتی که مقدار TEMPLATE را بر روی app تنظیم کنید، پروژه‌ی شما از نوع اجرایی (برنامه‌ی کاربردی) ساخته خواهد شد. در صورتی که مقدار آن را از نوع lib قرار دهید خروجی پروژه‌ی شما از نوع کتابخانه‌ای ساخته خواهد شد.

در خط بعد توسط متغیر QT_DEFINES مشخصه‌ی جهت اعمال اخطار برای ویژگی‌های منسوخ شده در نسخه‌ی فعلی کتابخانه استفاده شده است.

```
DEFINES += QT_DEPRECATED_WARNINGS
```

خط بعدی مشخصه‌ی CONFIG یکی از پرکاربردترین متغیرهایی است که در فایل .pro کاربرد دارد. توسط این مشخصه می‌توانید متغیرهای مربوط به ویژگی‌های مورد نیاز را مشخص کنید.

```
CONFIG += c++11
```

در مثال فوق، پروژه‌ی مربوطه از استاندارد سی‌پلاس‌پلاس ۱۱ پشتیبانی می‌کند که در ادامه لیستی از ویژگی‌های مربوط به آن آورده شده اند:

توضیحات	گزینه
این مقدار مشخص می‌کند که نوع پروژه به صورت انتشار نهایی ساخته خواهد شد.	release
این مقدار مشخص می‌کند که نوع پروژه به صورت دیباگ (اشکال‌زدایی) ساخته خواهد شد.	debug
این مقدار پروژه را برای ساخته شدن در حالت‌های اشکال‌زدایی (debug) و انتشار (release) آماده می‌کند.	debug_and_release
در صورتی که متغیرهای debug و release هر دو باهم تنظیم شده باشند هر کدام از آن‌ها در مسیر خود ساخته خواهند شد.	debug_and_release_target
اگر مشخصه‌ی debug_and_release تنظیم شده باشد، پروژه در دو حالت اشکال‌زدایی و انتشار باهم ساخته خواهند شد.	build_all

توضیحات	گزینه
این مشخصه باعث می‌شود به صورت خودکار یک فایل از نوع .cpp که شامل هدرهای از پیش کامپایل شده هستند را شامل شود.	autogen_precompile_source
هنگام استفاده از قالب subdirs، این گزینه مشخص می‌کند که دایرکتوری‌های لیست شده باید در ترتیبی که داده می‌شوند پردازش شوند.	ordered
پشتیبانی از استفاده از سرآیندهای پیش ترجمه شده در پروژه را فعال می‌کند.	precompile_header
پشتیبانی از استفاده از هدرهای قبل از ترجمه برای فایلهای C را فعال می‌کند.	precompile_header_c MSVC
با مشخص شدن این گزینه، مترجم (مترجم) هشدارهایی که ممکن است تولید شوند را نمایش خواهد داد.	warn_on
با مشخص سازی این گزینه، مترجم همان هشدارهای ضروری را نمایش خواهد داد.	warn_off
پشتیبانی از استثناءها فعال می‌شود.	exceptions
پشتیبانی از استثناءها غیر فعال می‌شود.	exceptions_off
پشتیبانی از RTTI به صورت پیشفرض توسط مترجم استفاده می‌شود که فعال است.	rtti
پشتیبانی از RTTI غیر فعال می‌شود.	rtti_off
پشتیبانی از STL به صورت پیشفرض فعال می‌شود.	stl
پشتیبانی از STL به صورت پیشفرض غیر فعال می‌شود.	stl_off
پشتیبانی از چندنخی فعال می‌شود. این ویژگی زمانی فعال می‌شود که پروژه شامل کیوت باشد.	thread
پشتیبانی از C99 فعال می‌شود. در صورتی که مترجم از استاندارد C99 پشتیبانی نکند، این گزینه هیچ تاثیری بر آن نخواهد داشت.	c99
پشتیبانی از C11 فعال می‌شود. در صورتی که مترجم از استاندارد C11 پشتیبانی نکند، این گزینه هیچ تاثیری بر آن نخواهد داشت.	c11

توضیحات	گزینه
پشتیبانی از ویژگی‌های اضافی مترجم برای C را غیرفعال می‌کند. این گزینه به صورت پیش‌فرض فعال است.	strict_c
پشتیبانی از استاندارد C++11 را فعال می‌کند. در صورتی که مترجم از استاندارد ۱۱ پشتیبانی نکند، این گزینه تاثیری بر آن نخواهد گذاشت. به صورت پیش‌فرض پشتیبانی از این استاندارد فعال است.	c++11
پشتیبانی از استاندارد C++ ویرایش ۱۴ را فعال می‌کند. در صورتی که مترجم از استاندارد ۱۱ پشتیبانی نکند، این گزینه تاثیری بر آن نخواهد گذاشت. به صورت پیش‌فرض پشتیبانی از این استاندارد فعال است.	c++14
پشتیبانی از استاندارد C++ ویرایش ۱۷ را فعال می‌کند. در صورتی که مترجم از استاندارد ۱۷ پشتیبانی نکند، این گزینه تاثیری بر آن نخواهد گذاشت. به صورت پیش‌فرض پشتیبانی از این استاندارد فعال است.	c++1z
این گزینه نیز همانند مشخصه <code>c++1z</code> عمل می‌کند.	c++17
پشتیبانی از ویژگی‌های اضافی مترجم برای C++ را غیرفعال می‌کند. این گزینه به صورت پیش‌فرض فعال است.	strict_c++
اضافه کردن مقدار <code>INCLUDEPATH</code> به <code>DEPENDPATH</code> که به <code>embed_translations</code> تنظیم شده است.	depend_includepath
اجرای <code>lrelease</code> برای تمامی فایل‌های ذکر شده در <code>TRANSLATIONS</code> و <code>EXTRA_TRANSLATIONS</code> فراهم می‌شود. در صورتی که <code>embed_translations</code> تنظیم نشده باشد، فایل‌های ساخته شده <code>.qm</code> را در <code>QML_FILES_INSTALL_PATH</code> نصب کنید. همچنین از گزینه‌ی <code>QMAKE_LRELEASE_FLAGS</code> جهت افزودن گزینه‌ها برای <code>lrelease</code> استفاده می‌شود که به صورت پیش‌فرض فعال نیستند.	<code>lrelease</code>
قرار دادن ترجمه‌های ساخته شده از <code>lrelease</code> در فایل اجرایی، تحت <code>QM_FILES_RESOURCE_PREFIX</code> که نیاز به مشخص سازی <code>lrelease</code> را خواهد داشت. همچنین به صورت پیش‌فرض مشخص نشده است.	<code>embed_translations</code>

گزینه‌ی Headers وظیفه نگهداری تمام فایل‌های C++ از نوع h. یا همان سرآیند (Header) را بر عهده دارد. گزینه‌ی Sources وظیفه نگهداری تمام فایل‌های C++ از نوع cpp. یا همان منبع (Source) را بر عهده دارد. گزینه‌ی Forms وظیفه نگهداری تمام فایل‌های مربوط به طراحی را دارد که پسوند فایل‌های طراحی در کیوت ویجت A UI تعریف می‌شوند و به صورت mainwindow.ui که در فایل طراحی پروژه شما به عنوان یک فرم در نظر گرفته شده است قابل دسترس خواهند بود.

گزینه‌های android، unix، qnx نیز نوع دیگری از مشخصه‌ها هستند که شرط‌گذاری برای توسعه در بسترها مربوطه کاربرد دارد. به عنوان مثال شما می‌توانید از دستورات بلاک هر بستر برای خود آن استفاده کنید.

```
win32 {
    HEADERS += windows/winapi.h
}

macos {
    HEADERS += windows/macosapi.h
}

ios {
    HEADERS += windows/iosapi.h
}

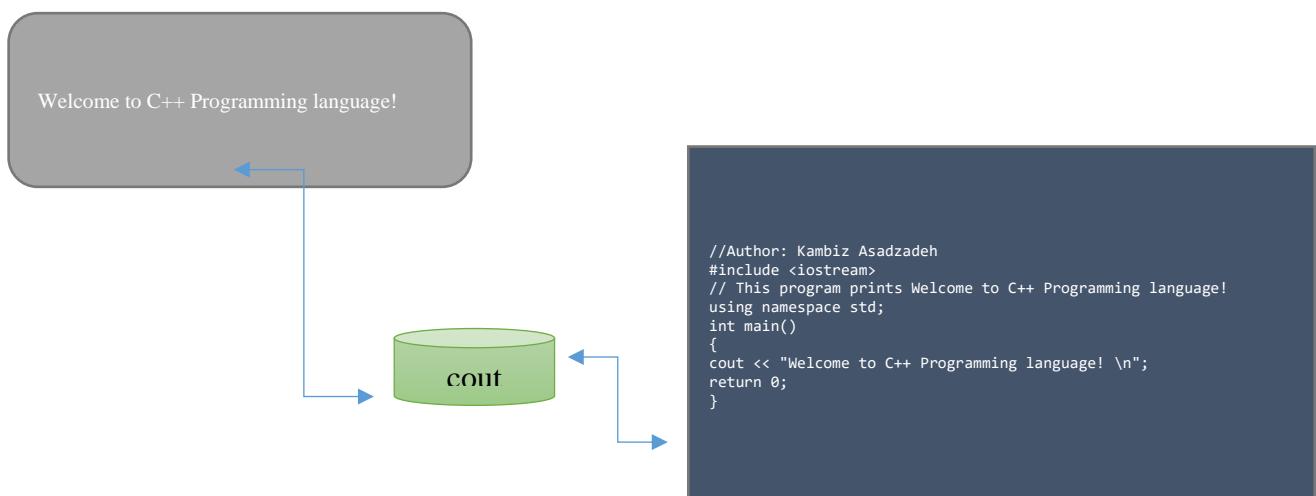
android {
    HEADERS += windows/androidapi.h
}

linux {
    HEADERS += windows/linuxapi.h
}
```

ساده ترین برنامه

همانطور که می‌دانید در هر زبان برنامه‌نویسی می‌توان ساده ترین برنامه را در چند خط کوتاه ایجاد و نتیجه را نمایش داد برای اینکه بهتر و از مراحل ابتدائی ساختار تولید برنامه در C++ را مشاهده کنیم به صورت زیر ساده ترین کدنویسی به زبان C++ و البته در نسخه جدید را توضیح می‌دهیم.

به شکل زیر توجه کنید :



این یک برنامه ساده که فقط عمل نمایش یک متن را در خروجی نمایش داده است که به صورت زیر به تجزیه و تحلیل خط به خط آن می‌پردازیم اگر ما بخواهیم توسط کتابخانه Qt آن را بازنویسی کنیم به صورت زیر خواهد بود :

```
//Author: Kambiz Asadzadeh

#include <QCoreApplication>
#include <QDebug>

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

    qDebug() << "Welcome to C++ Programming language!\n";

    return a.exec();
}
```

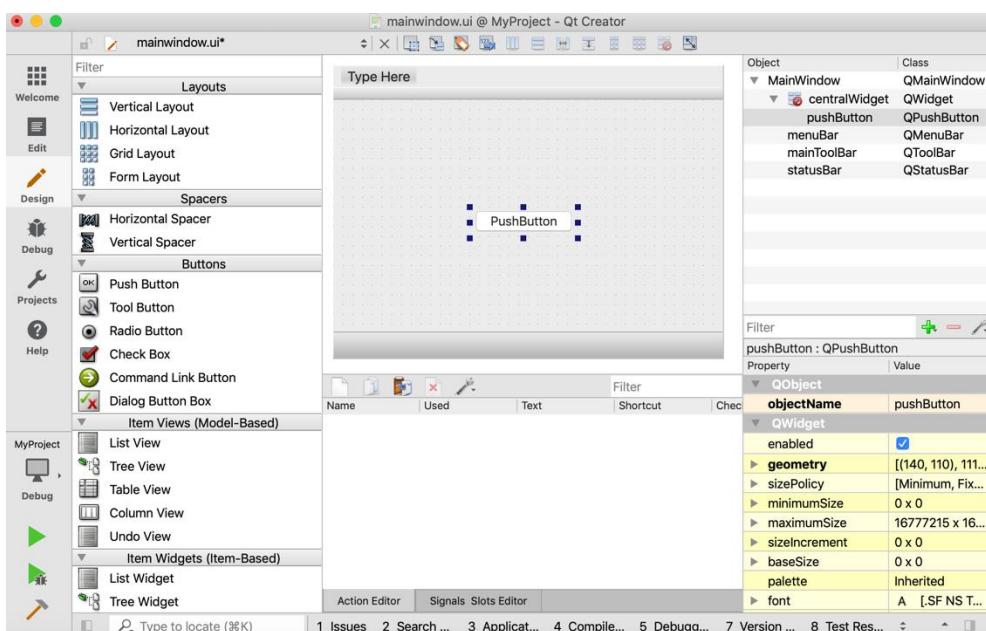
سیگنال یا اسلات چیست؟

مسئلماً شما برای ایجاد یک رویداد در یک زمان با اثرگذاری بر روی یک کنترل یا شیء (Object) انتظار این را دارید که آن شیء در قبال فشرده شدن یا هر رویداد دیگری واکنشی را نسبت به آن نشان دهد.

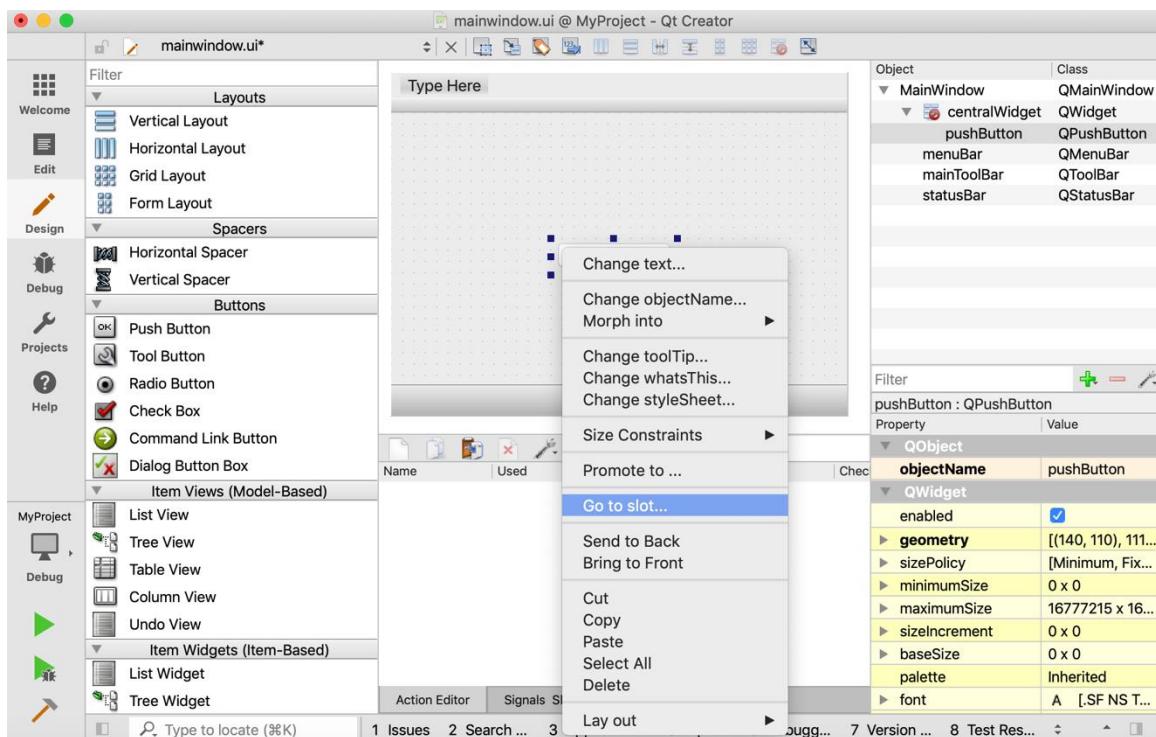
برای مثال بر روی یک دکمه‌ای کلیک می‌کنید انتظار دارید یک پیغامی در رویداد Clicked آن نمایش داده شود منظور از Signal و Slot به ترتیب، سیگنال و اسلات در رابطه با این مسائل است که حال در این میان برای صدا زدن از Signal استفاده می‌کنیم و از Slot برای دریافت دستور و عمل کردن.

*به طور کلی سیگنال و اسلات روش‌های برقراری ارتباط بین اشیاء هستند. این ویژگی به عنوان یک مکانیزم مرکزی از ویژگی‌های اصلی کیوت بشمار می‌رود.

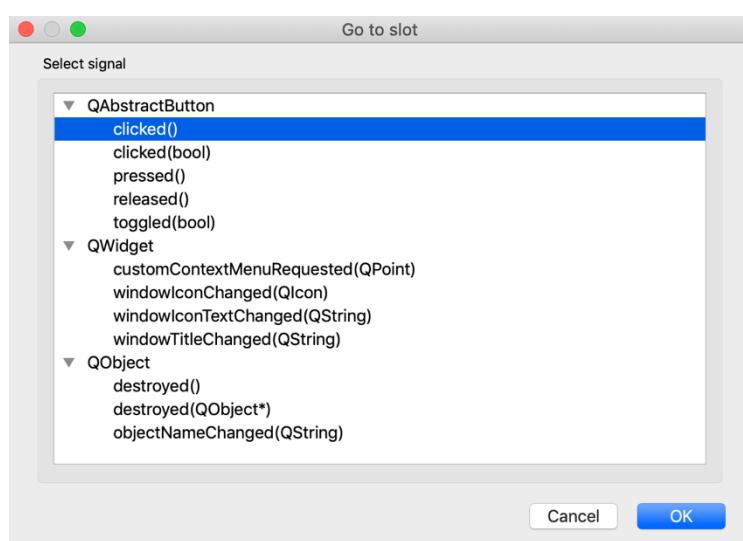
برای شروع کار روی یک فرم شیء ای را ایجاد کنید، قصد داریم تا از رویداد کلیک شدن بر روی یک شیء استفاده کنیم. بر روی ناحیه Forms در پروژه رفته و روی فایل mainwindow.ui دوبار کلیک کنید تا محیط طراحی فرم نمایان شود. یک شیء از منوی سمت چپ کنترل‌ها انتخاب و به عمل کشیدن و رها کردن بر روی فرم آن را ایجاد کنید.



سپس بر روی راست کلیک کرده و گزینه Go to Slot را انتخاب کنید که در ادامه آمده است.



طبق تصویر زیر مشاهده می‌کنید که تمام رویدادهای مربوط به کنترل مورد نظر شما قابل انتخاب هستند ما در این بخش لازم است از رویداد **clicked** استفاده کنیم، بنابراین آن را انتخاب و گزینه‌ی OK را اعمال کنید.



حال در این بخش تابع مربوط به عمل کلیک شدن روی این دکمه ایجاد شده است.

```
void MainWindow::on_pushButton_clicked()
{
}
```

در فایل mainwindow.h کد زیر اضافه شده است که تعریف اسلاتِ مربوط به کنترل است.

```
private slots:  
void on_pushButton_clicked();
```

توجه داشته باشید که بعد از کلمه‌ی **slots** گزینه‌ی **private** نشان‌گر آن است که تابع مربوطه تحت مکانیزم اسلات در کیوت تعریف شده است.

منظور از **pushButton** نام دکمه‌ای است که روی آن کلیک خواهد شد مسلماً نام دکمه بعدی خواهد بود! شما می‌توانید این تابع را بدون عملیاتی که در تصویر دیده‌اید ایجاد و به هر یک از کنترل‌های خود نام و مشخصاتی را اختصاص دهید.

در این بخش مثالی برای نمایش پیغام با قابلیت کلیک شدن آورده شده است، بنابراین طبق قوانین C++ ابتدا **Message** را فراخوانی می‌کنیم که نام هدر و کلاس آن **MessageBox** است:

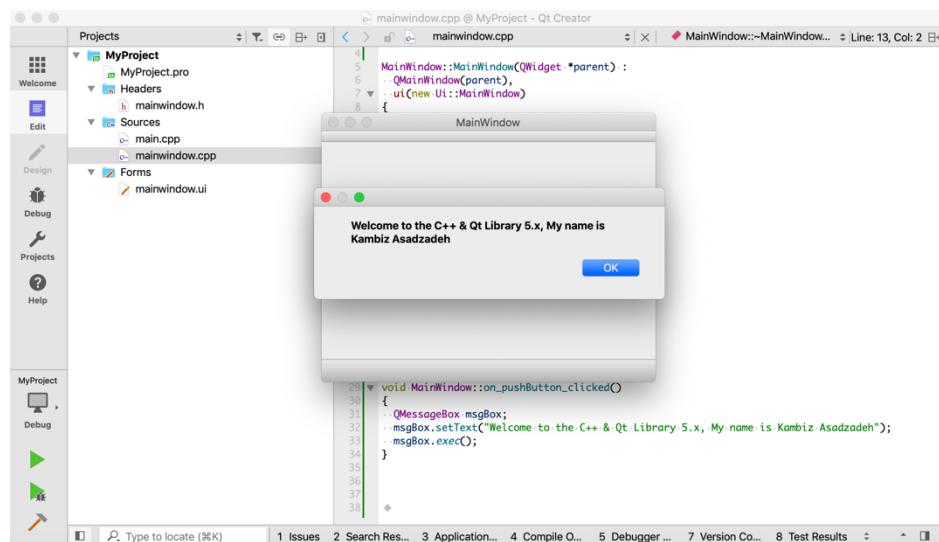
```
#include <QMessageBox>  
  
سپس کد زیر را داخل تابع رویداد مربوط به کلیک شدن اضافه می‌کنیم:
```

```
QMessageBox msgBox;  
msgBox.setText("Welcome to the C++ & Qt Library 5.x, My name is  
Kambiz Asadzadeh");  
msgBox.exec();
```

درواقع کد دامنه‌ی تابع مربوط به رویداد کلیک بر روی دکمه به صورت زیر خواهد بود:

```
void MainWindow::on_pushButton_clicked()  
{  
    QMessageBox msgBox;  
  
    msgBox.setText("Welcome to the C++ & Qt Library 5.x, My name is  
Kambiz Asadzadeh");  
  
    msgBox.exec();  
}
```

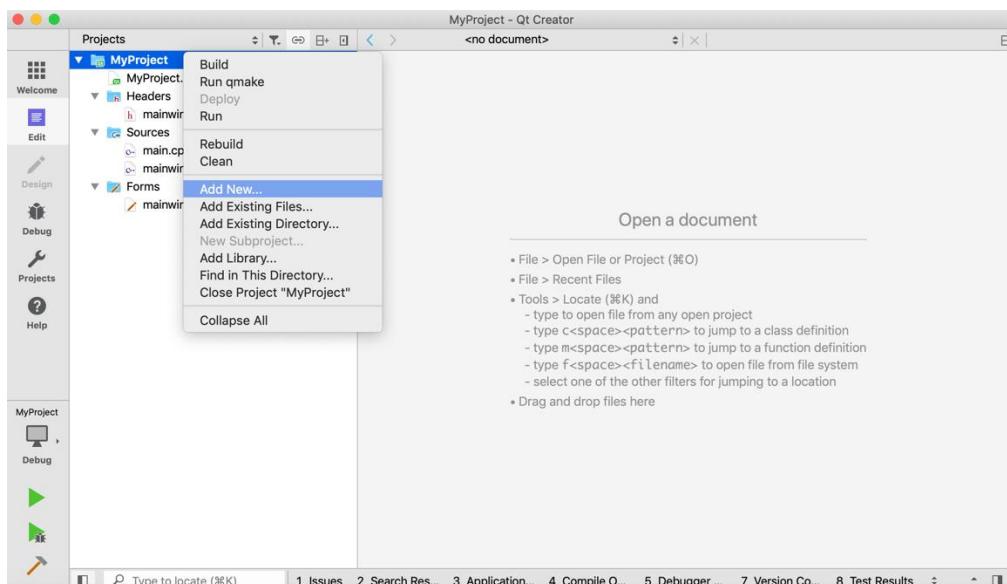
حال بعد از کامپایل و اجرای برنامه و کلیک بر روی دکمه مورد نظر پیغام ما به صورت زیر نمایش داده خواهد شد.



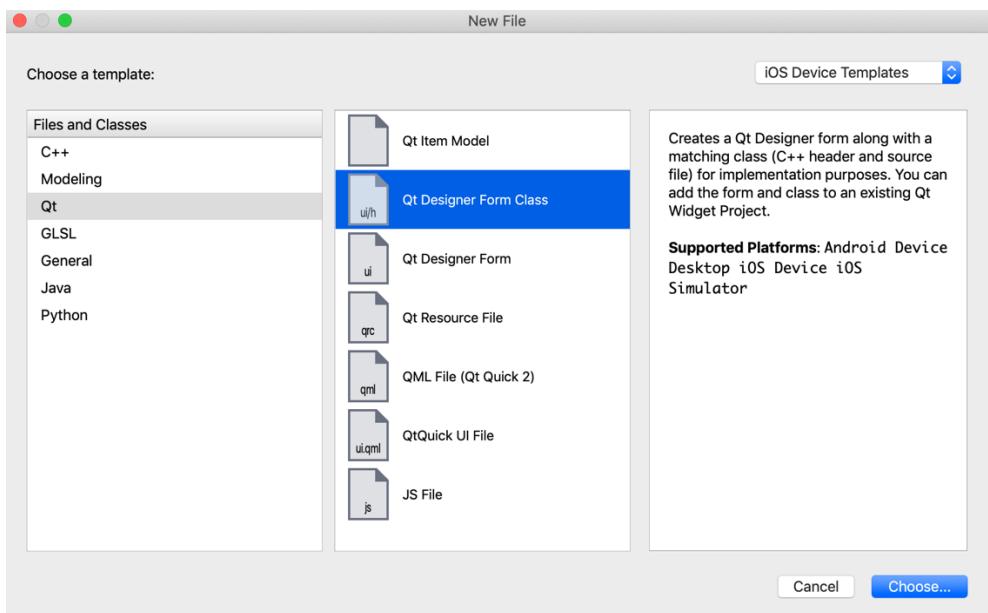
معرفی و کار با نمایش Windows

اگر توجه کرده باشید خواهید دید که معمولاً نیاز است از فرم و پنجره‌هایی در برنامه‌ها استفاده شود که هر کدام از آن‌ها می‌توانند برای اهداف خاصی طراحی و کدنویسی شوند؛ معمولاً از Widgets یا Dialog یا استفاده می‌شود و برای افزودن آن‌ها به پروژه باید روی عنوان پروژه راست کلیک و گزینه Add new گزینه را انتخاب

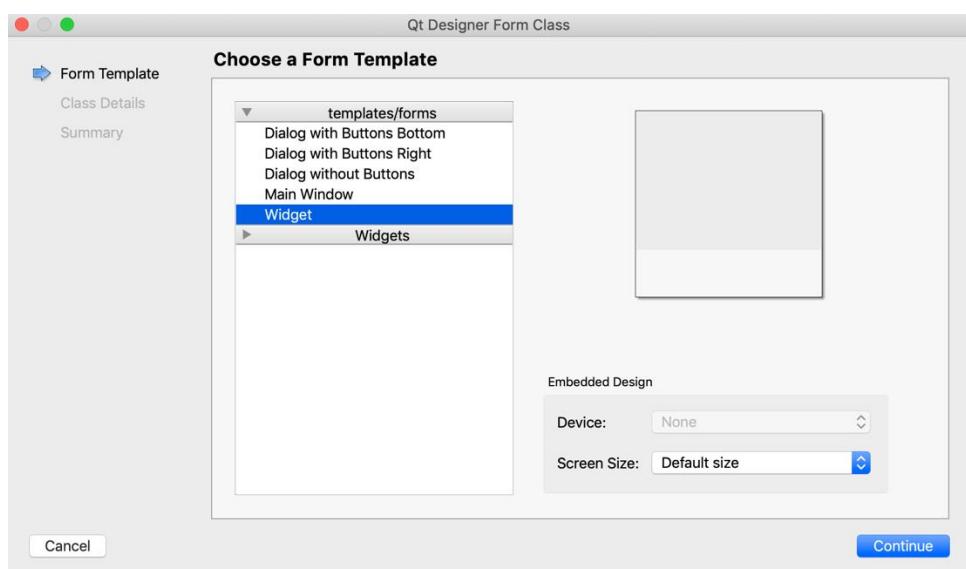
نماییم.



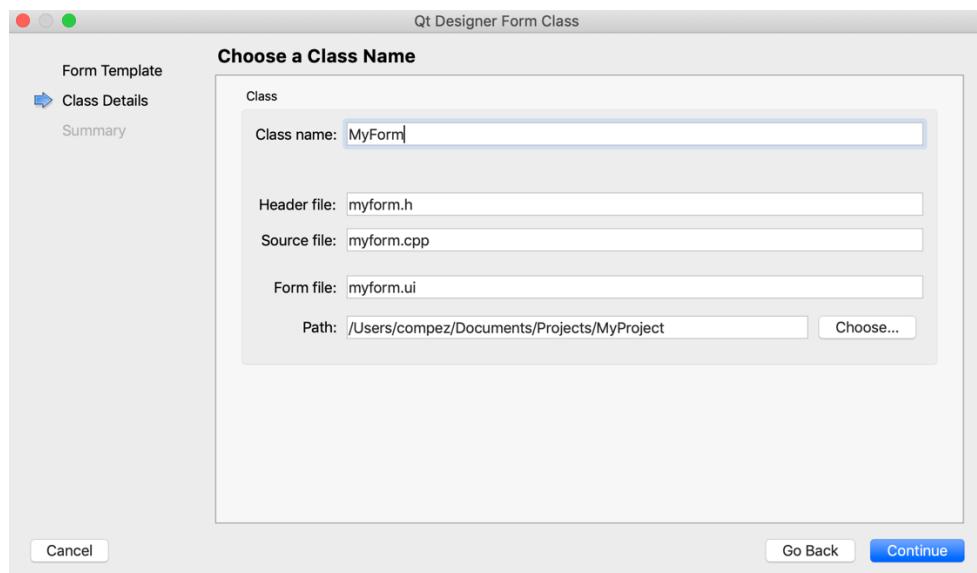
انواع گزینه‌های مختلفی وجود دارد که هر کدام بر اساس گزینه‌های مورد نیاز قابل انتخاب می‌باشند. معمولاً طبق قوانین C++ نیاز است فرم ما شامل ھدر و سورس همراه با فایل Source که در اینجا UI است. برای شروع کار ما از جامعترین یعنی Qt Designer Form class استفاده می‌کنیم.



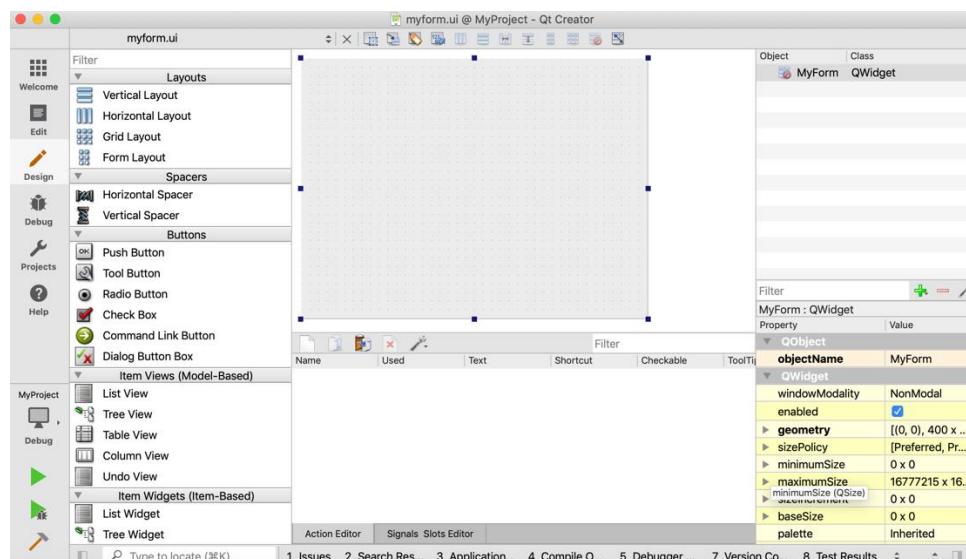
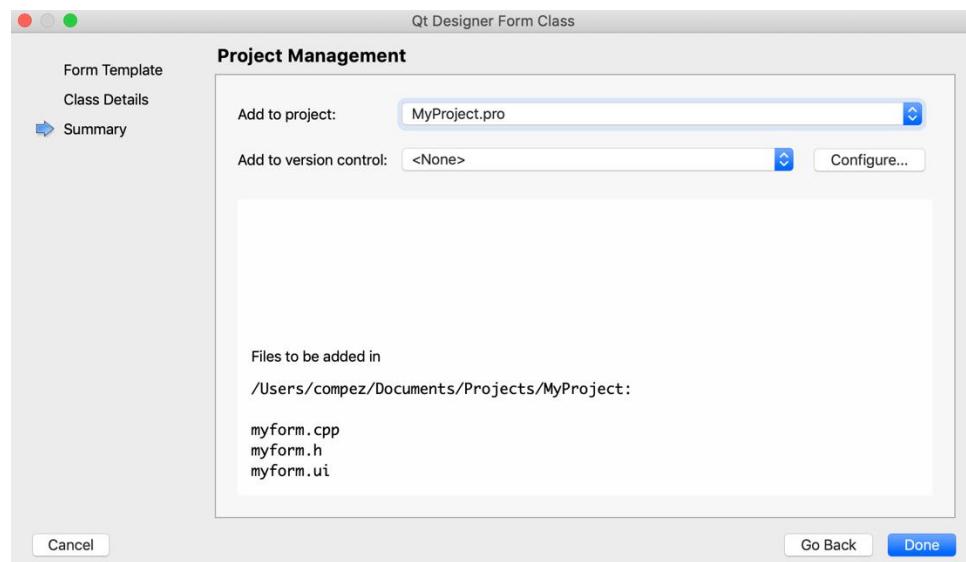
در این بخش شما می‌توانید نوع فرم خود را انتخاب کنید، فرمیدر قالب ویجت با دکمه‌ها و نگه‌دارنده‌های ویژه و یا فرمی در قالب دیالوک‌های ساده! برای معرفی نیاز است ما از نوع ویجت‌های معمولی استفاده کنیم. بنابراین نوع Widget را انتخاب و ادامه دهید.



در مرحله بعد شما می‌توانید به صورت سفارشی نام کلاس، نام هدیر و نام فایل اصلی و فایل طراحی را مشخص کنید که در خروجی بر اساس نام انتخاب شده مورد استفاده قرار می‌گیرد که در این آموزش نام فرم را مشخص و به صورت زیر تایید می‌کنیم.

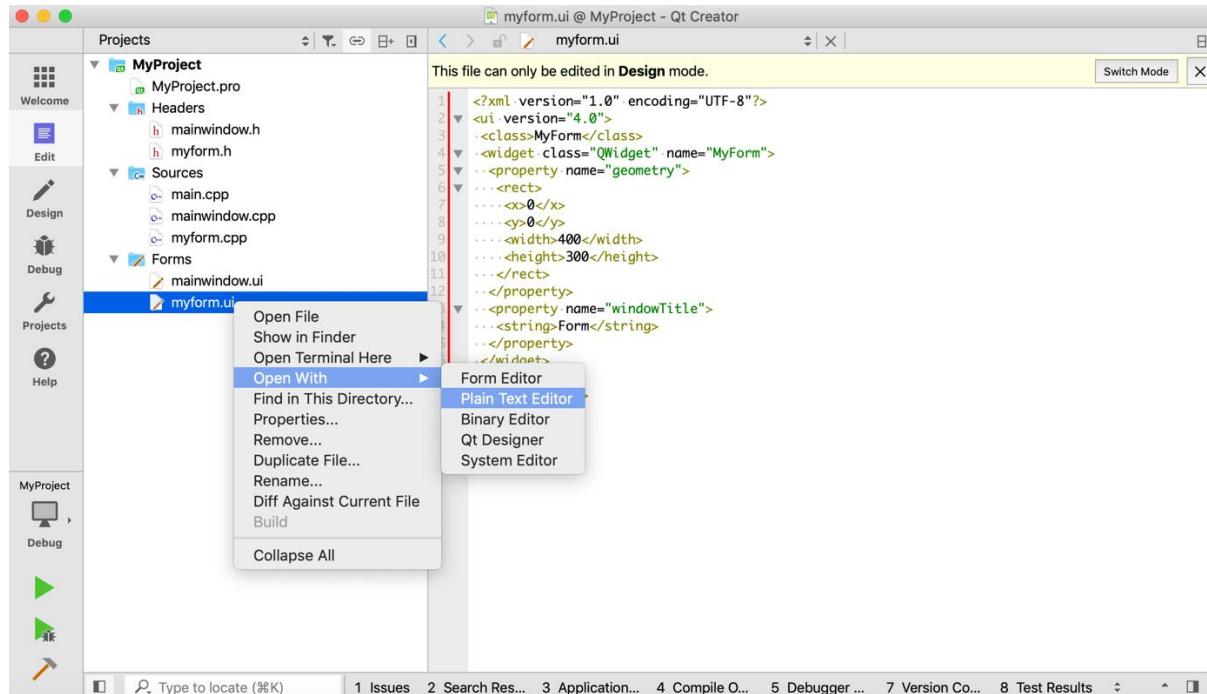


فرم مربوطه ایجاد شده و ما می‌توانیم از آن استفاده کنیم.



فرم مربوطه آماده‌ی طراحی توسط کنترل‌های موجود در محیط کیوت کریتور است.

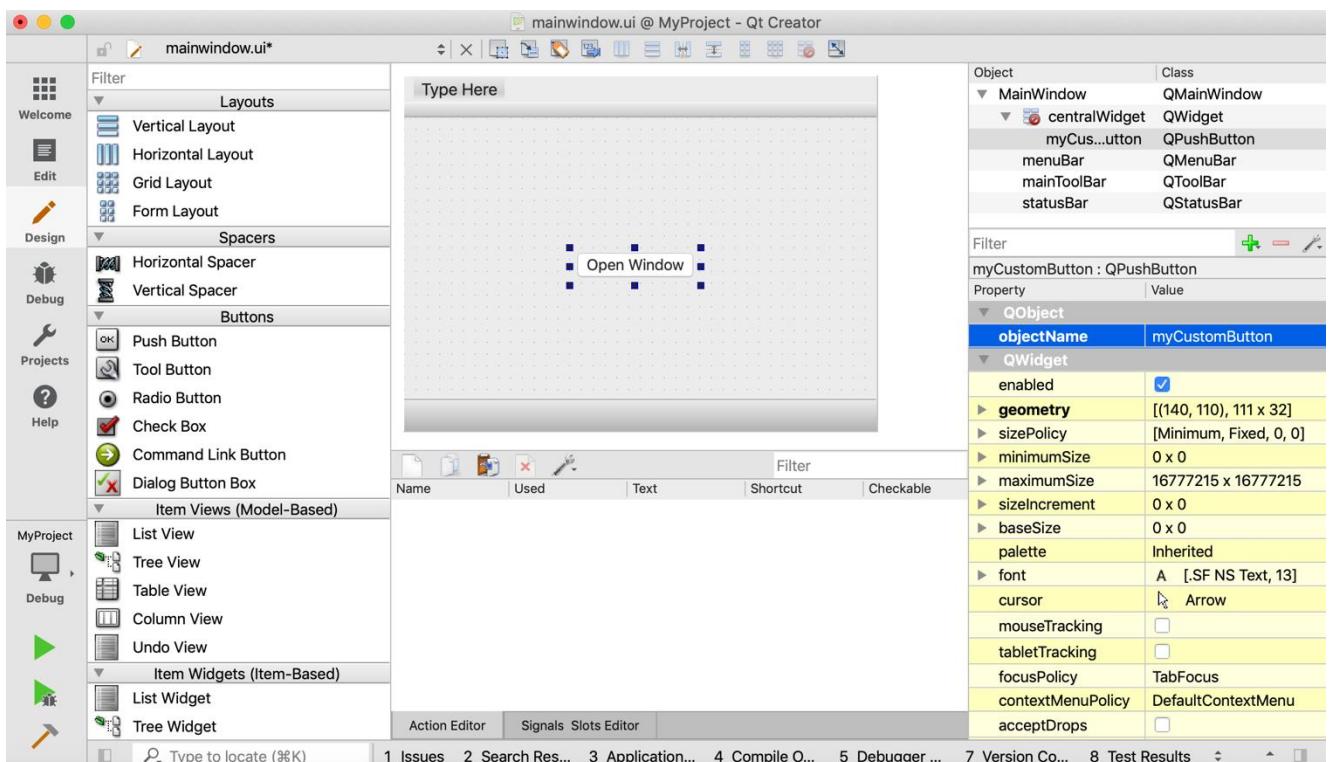
توجه داشته باشید که امکان طراحی در قالب کدهای XML در این محیط وجود دارد، کافی است بر روی فرم خود راست کلیک و گزینه **Open With** را انتخاب کنید تا کدهای مربوط به آن را ویرایش و مشاهده نمایید.



طبق تصویر بالا مشخص است فرم ایجاد شده دارای فایل هدر myform.h و دو فایل دیگر myform.cpp و myform.ui که به ترتیب شامل کدهای C++ و XML است ایجاد شده‌اند. کافی است بر روی فایل myform.ui دوبار کلیک کرده تا محیط طراحی مربوط به آن فرم نمایش داده شود.

حال برای نمایش فرم ساخته شده در فرم اصلی توسط event مربوط به دکمه دوم اقدام می‌کنیم طبق روش گفته شده از طریق **Got to Slot** تابع مربوط به رویداد clicked را می‌سازیم که در فرم MainWindow خواهد بود. بنابراین رویداد باید در فایل mainwindow.cpp ایجاد شود.

جهت سفارشی سازی، نام‌گذاری و اعمال عنوان مشخص به کنترل مورد نظر را انتخاب و از بخش سمت راست محیط توسعه مقدار objectname می‌کنیم که در این مثال مقدار mycustomButton را اعمال کرده و عنوان آن را به **Open Window** تغییر داده‌ایم. همچنین می‌توانید با تغییر مشخصه windowTitle در فرم مورد نظر عنوان مربوطه را برای آن اعمال کنید.



جهت تغییر عنوان می‌توانید بر روی کنترل دوبار کلیک و نام آن را تغییر دهید، و یا از سمت راست گزینه‌ها مقدار آن را تغییر دهید. طبق روال قبل بر روی کنترل راست کلیک کرده و **Go to slot** را انتخاب و رویداد **clicked** آن را باز کنید تا کد مربوط به رویداد کنترل به صورت زیر تولید شود:

```
void MainWindow::on_myCustomButton_clicked()
{
}
```

همچنین کد اعلان آن به صورت زیر خواهد بود که در فایل mainwindow.h تعریف شده است.

```
private slots:
    void on_myCustomButton_clicked();
```

نکته‌ی قابل توجه* اجبار نیست همیشه به صورت Wizard یا همان روش طراحی بصری برای ایجاد رویداد کنترل مورد نظر استفاده کنید، شما می‌توانید کدهای مربوطه را با رعایت کلمات کلیدی **signals** و **slots** آنها را اعلان نمایید.

در نهایت برای اینکه فرم ساخته شده با نام MyForm را در تابع رویداد کلیک اجرا کنیم، ابتدا باید فایل سر آیند (Header) مربوط به آن فرم را فراخوانی نماییم که به صورت زیر خواهد بود.

```
#include "myform.h"
```

با فراخوانی این فایل دسترسی لازم برای ساختن اشاره گر از آن فرم اعمال می‌شود. بنابراین می‌توان با نوشتن کد زیر عمل نمایش فرم را ایجاد کرد.

```
void MainWindow::on_myCustomButton_clicked()
{
    MyForm *dialog = new MyForm();
    dialog->show();
    // ...
}
```

توجه داشته باشید که بر اساس قوانین C++ ابتدا مشخص کنید که به چه شیوه‌ای حافظه‌ای را برای اشیاء مورد نظر خود اختصاص خواهید داد. در استک (Stack) یا پشته (Heap). در کیوت می‌توانید از همان کلمه‌ی کلیدی delete در سی++ و یا تحت کیوت از روش deleteLater حافظه را در موقعیتی که نیاز است آزادسازی کنید. البته ناگفته نماند اشاره‌گرهای هوشمند نیز عملی مشابه GB را انجام خواهند داد که می‌توانید از آن‌ها هم استفاده کنید. این به شما بستگی دارد که چطور آن را مدیریت کنید. و حتی خود کیوت هر کلاسی که از QObject ارث بری کند اشیاء موجود در والد خودش را به عنوان فرزند شناسایی می‌کند و دیگر نیاز نیست شما آن‌ها را به صورت دستی آزاد سازی کنید، زیرا آن‌ها به صورت خودکار در زمان اجرای مخرب (Destructor) از بین خواهند رفت.

بنابراین، کد مربوط به فایلmainwindow.cpp به صورت زیر خواهد بود که کد مربوط به تابع رخداد و نمایش فرم و همچنین فایل سرآیند ضخیم شده است.

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "myform.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::changeEvent(QEvent *e)
{
    QMainWindow::changeEvent(e);
    switch (e->type()) {
        case QEvent::LanguageChange:
```

```

        ui->retranslateUi(this);
        break;
    default:
        break;
    }
}

void MainWindow::on_myCustomButton_clicked()
{
    MyForm *dialog = new MyForm();
    dialog->show();
}

```

لازم است کمی تغییرات در مشخصه‌های فرم خود اعمال کنیم. اما اینبار به شیوه‌ی گذنویسی! بنابراین فایل myform.cpp را باز کرده و کد زیر را در حوزه‌ی دامنه‌ی سازنده اضافه کنید.

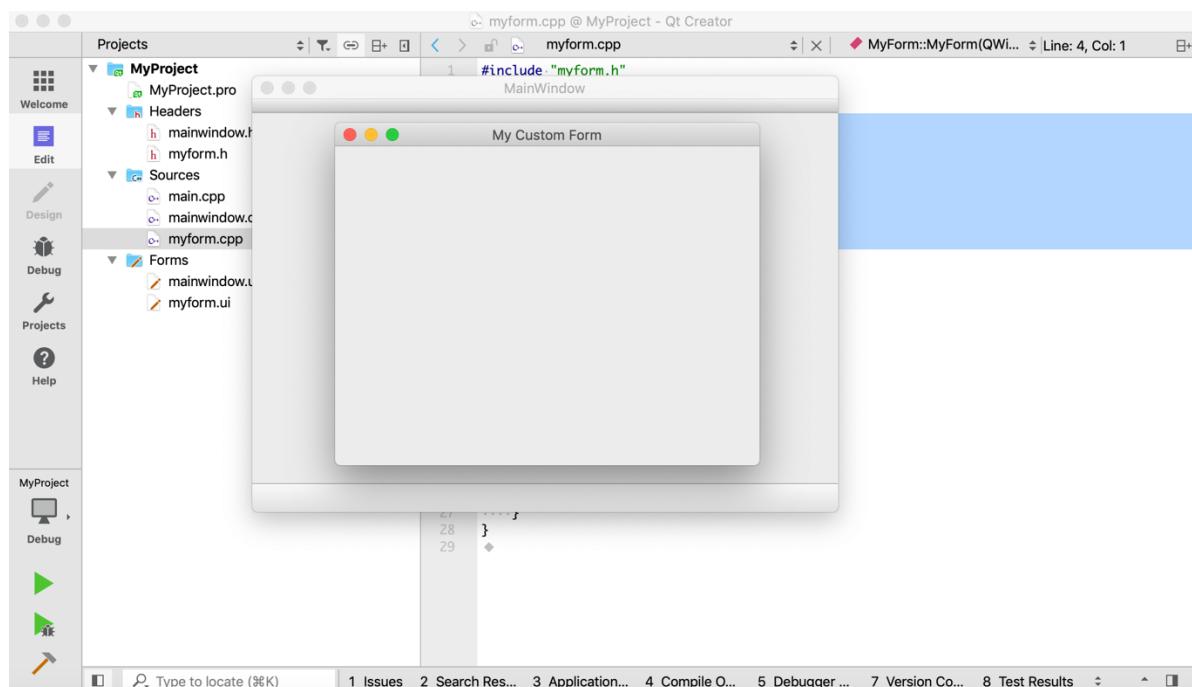
```

this->setWindowTitle("My Custom Form");

MyForm::MyForm(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::MyForm)
{
    ui->setupUi(this);
    this->setWindowTitle("My Custom Form");
}

```

با افزودن کد فوق عنوان فرم سفارشی شما نیز در هنگام نمایش (ساخته‌شدن) تغییر خواهد یافت که با کلید بر روی دکمه‌ی Open Window فرم سفارشی شما به صورت زیر نمایش داده خواهد شد.



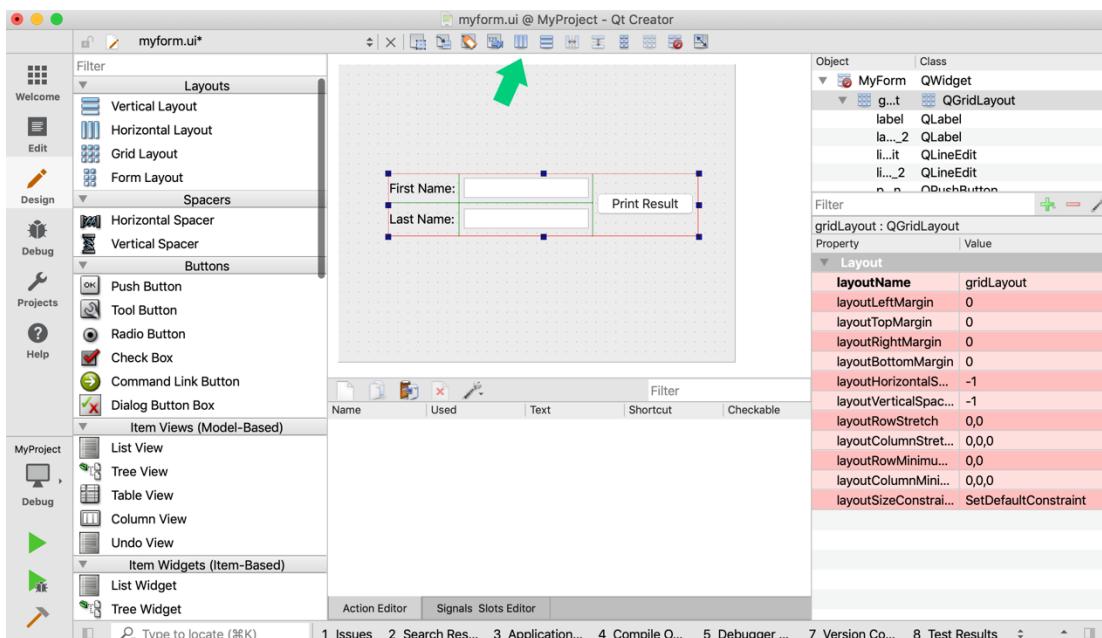
جهت بستن فرم مربوطه می‌توانید کلیدی را تعریف و کد زیر را در آن اعمال کنید.

```
this->close();
```

کد فوق عمل بسته شدن فرم جاری را انجام می‌دهد.

معرفی و کار با لایه‌ها زبانه‌ها و بدن‌های در طراحی

در این بخش توسط نوار ابزار بالا که شامل مواردی جهت تنظیم لایه‌ها برای دسترسی سریع لایه به لایه در طراحی است به کدنویسی و یا تنظیم موقعیت کنترل‌ها و فرم‌ها اقدام کنیم.



- گزینه **Edit Widgets**: در حالت کلی وقتی لازم است ابزارها و کنترل‌هایی را روی فرم قرار دهیم یا به طور ساده شیئی را روی فرم درج نماییم حتماً باید در این لایه کار کنیم.
- گزینه **Edit Signals/Slots**: به صورت کلی این لایه وظیفه در اختیار گذاشتن سریع دسترسی‌های مربوط به Signal‌ها و Slot‌ها را در اختیار ما می‌گذارد که با عمل کشیدن و رها کردن اشیاء روی هم‌دیگر مراحل بعدی آن نمایان می‌شود.
- گزینه **Edit Buddies**: این لایه امکان تنظیم اندازه فرم را فراهم می‌کند.
- گزینه **Edit tab order**: در این لایه از طراحی شما می‌توانید به راحتی شماره ایندекс یا همان index number اختصاصی هر یک از اشیاء را روی فرم مشخص کنید که با عمل Tab نیز بر اساس اولویت انتخاب می‌شوند.

یک نکته مهم در Qt Creator وجود دارد که عملیات کشیدن و رها کردن اشیاء بر خلاف محیط‌های توسعه‌ی دیگر ویژگی‌های بسیار زیادی را برای راحتی کار ایجاد می‌کند.

معرفی و کار با قابلیت‌های HTML و CSS در طراحی

داشتن یک طراحی رابط‌کاربری و تجربه‌کاربری مناسب یکی از فاکتورهای بسیار مهم در تولید محصول است؛ بنابراین دسترسی به ابزارها و امکان استفاده از فناوری‌های مربوط به این حوزه بسیار کارآمد در کیوت بشمار خواهد بود.

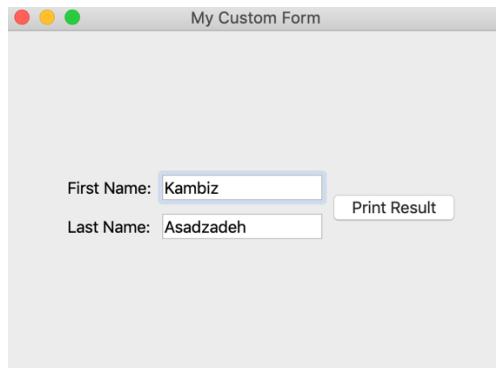
در رابطه با فناوری‌های معروفی چون JavaScript، HTML، CSS که در حوزه‌ی وب نیز بسیار قدرتمند عمل کرده‌اند باید گفت با ورود آن‌ها به کتابخانه Qt پیشرفت چشمگیری را در طراحی رابط‌های کاربری توسط C++ مشاهده می‌کنیم.

فناوری CSS با آوای (سی‌اس‌اس): روشی ساده برای نمایش چیدمان و جلوه‌های تصویری (مانند نوع قلم، رنگ و اندازه‌ها) بر صفحه‌های وب است. شیوه‌نامه آبشاری از جنس زبان‌های نشانه‌گذاری، با ساختار متن ساده رایانه هستند و درون هرکدام، دستورهایی آبشار مانند و پی‌درپی، برای چگونگی نمایش هر صفحه وب افزوده می‌شود. به گفته‌ای ساده‌تر، این دستورها روش نشان داده شدن قلم‌ها و اندازه‌ی آن‌ها، رنگ‌ها و پس زمینه‌ها، روش چیدمان موزائیک‌های دربرگیرنده‌ی داده‌ها (دیواره‌ها)، و بسیاری دیگر از عنصرهای ساختار هر صفحه وب را درون خود جای می‌دهند.

فناوری HTML: زبان امتدادپذیر نشانه‌گذاری فرمت‌نی ای ام ال (Extensible HyperText Markup Language - XHTML) همان اچ‌تی‌ای‌ال است به همراه رعایت دقیق تمامی قواعد و دستورات نحوی نزدیک‌تر به زبان اکس‌ام‌ال که موجبات افزایش اطمینان از عمل‌کرد صحیح سندها در شرایط پیچیده‌تر موجود در اینترنت امروزین را فراهم می‌سازد. XHTML‌ها، نوع‌های سندها و ماژول‌ها در حال حاضر و در آینده هستند که در واقع زیر مجموعه و گسترش یافته HTML4 است. این نوع اسناد بر پایه XML هستند و برای کار در ترکیب با عامل کاربر مبتنی بر XML طراحی شده‌اند. و در حال حاضر نیز جدیدترین آنها HTML5 است.

در زبان C++ قابلیت این وجود دارد که ما از این فناوری‌ها توسط کتابخانه کیوت استفاده نماییم. بنابراین پروژه‌ای را ایجاد کنید که شامل یک فرم و چند کنترل باشد تا "استایل" یا همان ظاهر آن را توسط این فناوری‌ها بازسازی نماییم.

به تصویر زیر توجه نمایید که بدون هیچ کدی رابط‌کاربری به صورت کلاسیک نمایش داده می‌شود.



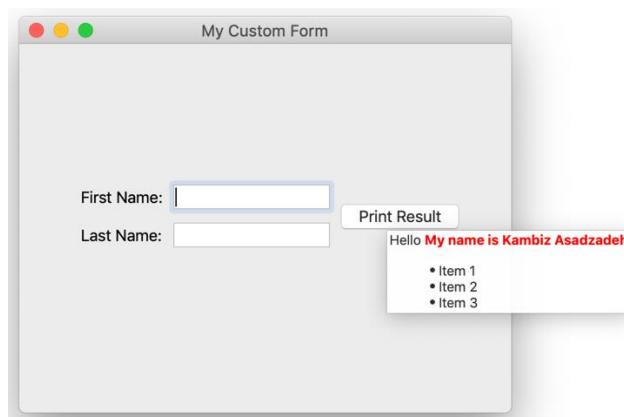
حال برای طراحی رابط گرافیکی زیبا توسط HTML و CSS چگونه باید عمل کنیم؟ برای اینکار ما می‌توانیم به دو روش اقدام نماییم که به صورت روش سریع از طریق کلیک بر روی شیء و انتخاب گزینه‌ی Change انتخاب کرده و یا توسط اشاره‌گر setStyleSheet از آن استفاده می‌کنیم.

```
ui->label->setStyleSheet ("Your HTML/CSS Code");
```

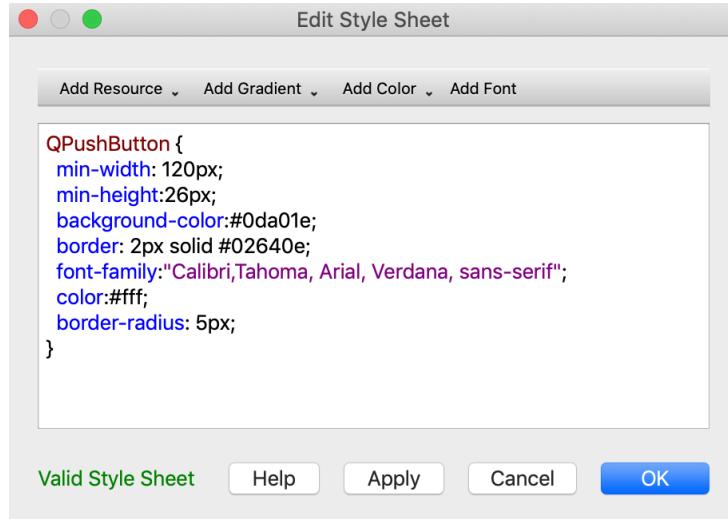
برای مثال یک Tooltip را برای دکمه موجود بر روی فرم در نظر می‌گیریم باید به صورت زیر نوشته شود:

```
ui->pushButton->setToolTip ( "Hello <font color='red'><b>My name is Kambiz Asadzadeh</b></font>" <ul>"<li>Item 1</li>" "<li>Item 2</li>" "<li>Item 3</li>" "</ul>" );
```

نتیجه هنگام حرکت ماوس بر روی آن نمایش داده خواهد شد که به کمک کدهای HTML و CSS اعمال شده است.



حال با کدنویسی‌های CSS در استال شیت رابطکاربری نه چندان ساده به صورت زیر فراهم می‌شود طبق تصویر زیر مشخص است همان فرمی که طراحی شده است با کدنویسی CSS ظاهری سفارشی شده را دارد.



همچنین در بخش کد می‌توان به روش زیر عمل کرد:

```
ui->pushButton->setStyleSheet ("QPushButton {"  
    "min-width: 120px;"  
    "min-height: 26px;"  
    "background-color: #0da01e;"  
    "border: 2px solid #02640e;"  
    "font-family: \"Calibri\";"  
    "color: #fff;"  
    "border-radius: 5px;"  
"}");
```

- برای توسعه‌ی هرچه بهتر با CSS باید طبق قوانین آن عمل کنید.
- برای اطلاعات بیشتر در رابطه با CSS نویسی می‌توانید به آدرس <http://www.w3schools.com/css> مراجعه نمایید.
- جهت اطلاعات بیشتر در رابطه با کلاس‌ها و زیر کلاس‌های مربوط به آبجکت‌های موجود در [کیوت](#) به [مثال‌های آن](#) مراجعه کنید.

معرفی و کار با لایه‌های افقی و عمودی

معمولاً در نرم‌افزارها تنظیمات فرم‌ها و همچنین لایه‌ها تو سط Re-size شدن صورت می‌گیرند و در زمان تغییر اندازه آبجکت‌ها و کنترل‌های موجود بر روی فرم متناسب با تغییرات اندازه‌ی به خوبی تغییر می‌کنند در این حالت می‌گوییم تمامی کنترل‌های موجود روی لایه تنظیم شده است.

در کیوت کلاس‌هایی برای این کار وجود دارد که شامل **QVBoxLayout** و **QHBoxLayout** است؛ برای اینکار ابتدا تو سط کد فرم آزمایشی خود را ایجاد می‌کنیم و قصد داریم به صورت افقی آن را در لایه تنظیم نماییم به صورت زیر:

```
QWidget *window = new QWidget;
window->setWindowTitle("Window Layout");
```

در این کد ما یک شیء جدیدی از کلاس **QWidget** می‌سازیم سپس تو سط اشاره گر به خاصیت **WindowTitle** آن دسترسی پیدا کرده و عنوان آن را **Window Layout** می‌گذاریم. حال نیاز است سه عدد دکمه در این فرم ایجاد کنیم بنا براین باید به صورت زیر از آن‌ها نمونه ایجاد نماییم.

```
QPushButton *button = new QPushButton("Button One");
QPushButton *button1 = new QPushButton("Button Two");
QPushButton *button2 = new QPushButton("Button Three");
```

در این کد ما سه نمونه از کلاس **QPushButton** ایجاد کرده و نام‌های هرکدام را مشخص کرده‌ایم. اگر در این حالت برنامه را اجرا نماییم کنترل‌هایی که تعریف کرده‌ایم نمایش داده نخواهد شد زیرا لایه‌ای که آن‌ها را نگه خواهد داشت فعلًاً تعریف نشده است. بنابراین برای تعریف آن می‌توانیم از لایه‌های افقی یا عمودی استفاده کنیم که به صورت زیر تعیین می‌شوند.

```
QHBoxLayout *hlayout = new QHBoxLayout;
```

در این مرحله لایه مربوطه تعریف شده است ولی هنوز هیچ کنترلی در آن درج نشده است. برای اعمال آن‌ها باید به صورت زیر عمل شود.

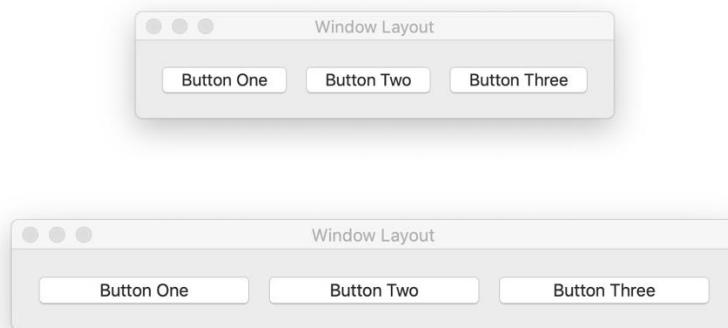
```
hlayout->addWidget(button);
hlayout->addWidget(button1);
hlayout->addWidget(button2);
```

در این بخش کنترل‌ها به ترتیب وارد لایه‌ی نگه‌دارنده شده‌اند که جهت نمایش بر روی لایه‌ی فرم والد به صورت زیر اعمال خواهد شد.

```
window->setLayout(hlayout);
```

در انتهای نمایش لایه و محتویات درونی آن در فرم کد نهایی را وارد می‌کنیم:

```
window->show();
```



کد کامل این بخش به صورت زیر است:

```
#include "mainwindow.h"
#include <QApplication>
#include <QPushButton>
#include <QHBoxLayout>
#include <QVBoxLayout>

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

    QWidget *window = new QWidget;
    window->setWindowTitle("Window Layout");

    QPushButton *button = new QPushButton("Button One");
    QPushButton *button1 = new QPushButton("Button Two");
    QPushButton *button2 = new QPushButton("Button Three");

    QHBoxLayout *hlayout = new QHBoxLayout;

    hlayout->addWidget(button);
    hlayout->addWidget(button1);
    hlayout->addWidget(button2);

    window->setLayout(hlayout);
    window->show();

    return a.exec();
}
```

معرفی و کار با لایه‌های Grid در طراحی فرم

به طور کلی در فرم‌هایی که اشیاء به صورت متراکم و یا تو در تو ایجاد و قرار داده می‌شوند نیاز است برای کنترل موقعیت و خاصیت آن‌ها از لایه‌های توری یا Grid استفاده شود تا به راحتی برای هریک از اشیاء موقعیت انحصاری تعریف شود.

مجدداً طبق کد زیر فرم خود را ایجاد و نام آن را Grid Window می‌گذاریم:

```
QWidget *window = new QWidget;
window->setWindowTitle("Grid Window");
```

اکنون نیاز به لایه‌ی توری (Grid) داریم که به صورت زیر تعریف می‌شود:

```
QGridLayout *Layout = new QGridLayout;
```

در این بخش چند کنترل تعریف می‌کنیم به صورت زیر:

```
QLabel *label1 = new QLabel("First name:");
QLineEdit *txtFirstName = new QLineEdit;
QLabel *label2 = new QLabel("Last name:");
QLineEdit *txtLastName = new QLineEdit;
QPushButton *button = new QPushButton("OK");
```

در این بخش باید به خصوصیاتی که Grid توسط آن اشیاء را نمایش می‌دهد توجه نماییم که این تابع به صورت زیر است :

```
void QGridLayout::addWidget(
    QWidget * widget, int fromRow, int fromColumn, int rowSpan, int
    columnSpan, Qt::Alignment alignment = 0
);
```

با توجه به خصوصیاتی که تابع دارد کد به صورت زیر بر اساس شماره‌ی ایندکس (شاخص)، سلول و ستون مشخص می‌شود.

```
Layout->addWidget(label1,0,0);
Layout->addWidget(txtFirstName,0,1);
Layout->addWidget(label2,1,0);
Layout->addWidget(txtLastName,1,1);
Layout->addWidget(button,2,0,1,2);
```

حال با توجه به کد کلی خروجی آن نیز به صورت زیر خواهد بود:



کد نهایی:

```
#include "mainwindow.h"
#include <QApplication>
#include <QPushButton>
#include <QLineEdit>
#include <QLabel>
#include <QGridLayout>

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

    QWidget *window = new QWidget;
    window->setWindowTitle("Grid Window");

    QGridLayout *Layout = new QGridLayout;

    QLabel *label1 = new QLabel("First name:");
    QLineEdit *txtFirstName = new QLineEdit;
    QLabel *label2 = new QLabel("Last name:");
    QLineEdit *txtLastName = new QLineEdit;
    QPushButton *button = new QPushButton("OK");

    Layout->addWidget(label1,0,0);
    Layout->addWidget(txtFirstName,0,1);
    Layout->addWidget(label2,1,0);
    Layout->addWidget(txtLastName,1,1);
    Layout->addWidget(button,2,0,1,2);

    window->setLayout(Layout);
    window->show();

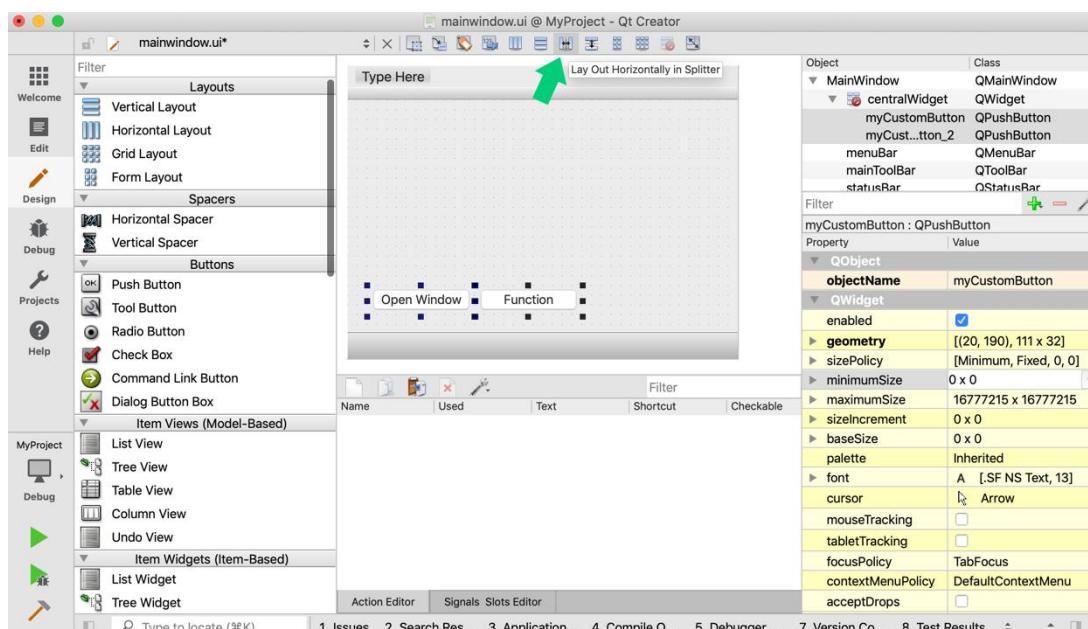
    return a.exec();
}
```

معرفی و کار با جدا کننده‌ها Splitter

در بسیاری از مواقع لازم است بین اشیاء و یا فرم با همیگر ارتباط از لحاظ موقعیت و چیدمان ایجاد شود که در این صورت ابزارهایی با دسترسی سریع در نوار ابزار بالا موجود هستند که در حالت‌های افقی و عمودی اشیاء را به یکدیگر تنظیم می‌کند.

فرض کنید اگر فرمی داریم که فاصله بین حاشیه‌های آن و اشیاء از هم بسیار است یا هنگام تغییر اندازه فرم اشیاء از یکدیگر جدا شده و نامرتب می‌شوند بنابراین برای تنظیم این نوع موارد توسط **Splitter** می‌توان به صورت عمودی و افقی اقدام کرد.

برای حل این مشکل از ابزارهای موجود در بالای نوار ابزار طراحی گزینه‌های مشخص شده را بر حسب موقعیت "افقی" و "عمودی" انتخاب کرده و برای اعمال کافی است اشیاء را انتخاب و گزینه "Layout Vertically in" را فشار دهید.



با استفاده از ابزار Layout Horizontally in Splitter اشیاء به صورت افقی تنظیم خواهند شد. و در حالت به صورت عمودی تنظیم می‌شوند. در نهایت با کشیدن نگه‌دارنده و فرم والد بر اشیاء چیدمان آن‌ها پراکنده نخواهد شد.

معرفی و کار با دایرکتوری‌ها

برای استفاده از دایرکتوری، پوشه، فایل و مشخصات مربوط به آن‌ها کلاسی به نام `QDir` موجود است که در این کلاس می‌توان به کار با فایل‌ها و پوشه‌ها پرداخت.

برای مثال برای بررسی یک مسیر به صورت زیر عمل خواهیم کرد :

```
QDir MyDir("c:/");
qDebug() << MyDir.exists();
```

ساختار این تابع به صورت زیر است :

```
bool QDir::exists(const QString & name) const
```

نوع کلی تابع از نوع "بولین" است. بنابراین در صورت تشخیص وجود مسیر تعریف شده مقدار "true" و در غیر اینصورت مقدار "false" را برگشت خواهد داد.

کد زیر عمل چاپ نتیجه را توسط مجرای `qDebug` بر عهده دارد.

```
qDebug() << MyDir.exists();
```

همان عمل چاپ را انجام می‌دهد دقیقاً مانند `std::cout` که در حالت استاندارد STL کاربرد دارد.

مثال زیر عمل نمایش درایوهای موجود را انجام می‌دهد.

به صورت زیر یه نمونه از `QDir` و یه نمونه برای لیست کردن درایو هام از `QFileInfo` و گرفتن اطلاعات آن‌ها می‌گیریم و در داخل یک حلقه روند را تا جایی که لازم است تکرار می‌کنیم.

```
QDir MyDir;
foreach (QFileInfo MyItem, MyDir.drives())
{
    qDebug() << MyItem.absoluteFilePath();
```

در این بخش وظیفه‌ی `absoluteFilePath` دریافت و ارسال اطلاعات و مشخصات مسیر و فایل است و با اجرای این دستور نتیجه را مشاهده خواهید کرد.

دستور نهایی آن به صورت زیر خواهد بود:

```
#include <QCoreApplication>
#include <QDebug>
#include <QDir>
#include <QFileInfo>

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

    QDir MyDir;
    foreach (QFileInfo MyItem, MyDir.drives())
    {
        qDebug() << MyItem.absoluteFilePath();
    }

    return a.exec();
}
```

معرفی و کار با فایل‌ها / خواندن و نوشتן در آن‌ها (Read)

در این بخش به نحوهی **خواندن** اطلاعات از فایل‌ها می‌پردازیم که برای شروع لازم است کلاس‌های زیر را فراخوانی کنیم:

```
#include <QFile>
#include <QString>
#include <QTTextStream>
```

برای خواندن فایل تابعی به صورت زیر می‌نویسیم:

```
void Read(QString Filename)
{
    QFile MyFile(Filename);

    if (!MyFile.open(QFile::ReadOnly | QFile::Text))
    {
        qDebug() << "could not open file for reading";
        return;
    }

    QTTextStream in(&MyFile);
    QString MyText = in.readAll();

    qDebug() << MyText;

    MyFile.close();
}
```

در این تابع نام آن را **Read** و متغیر ورودی آن را از نوع رشته‌ای با **QString** با نام **Filename** قرار داده‌ایم و برای اجرای این تابع نیاز است آن را به صورت زیر فراخوانی و مقدار دهی کنیم:

```
Read("/Users/compez/Desktop/test.txt");
```

خروجی دستور محتوایی خواهد بود که در داخل فایل مورد نظر یعنی **test.txt** موجود است برای مثال:

```
"My name is Kambiz Asadzadeh ☺"
```

برای دریافت نام و مسیر فایل تابع به صورت زیر تعریف می‌شود:

```
void Read(QString Filename);
```

ابتدا برای نوشتن و یا خواندن در داخل یک فایل باید از کلاس **QFile** یک نمونه گرفته شود تا بتوانیم این کار را انجام دهیم پس به صورت زیر ابتدا یه نمونه گرفته ایم:

```
QFile MyFile(Filename);
```

در خط بعدی یک دستور شرطی آورده شده است که اگر فایل را از نوع متنی و در حالت فقط خواندنی در نظر بگیریم اگر فایل را بر اساس این شرایط تشخیص و قادر به خواندن محتویات آن نشود پیغام عدم توانایی در باز کردن و خواندن فایل را نمایش بدهد بنابراین کد آن در ادامه به صورت زیر خواهد بود:

```
if(!MyFile.open(QFile::ReadOnly | QFile::Text))  
{  
    qDebug() << "could not open file for reading";  
    return;  
}
```

در قسمت بعد برای کار با یک فایل متنی و رشته‌ای لازم است از کلاس **QTextStream** یه نمونه ساخته شود:

```
QTextStream in(&MyFile);
```

اکنون در قسمت بعد تمام محتویاتی که در فایل وجود دارد را دریافت خواهد کرد به صورت زیر:

```
QString MyText = in.readAll();
```

در قسمت بعد دستور چاپ محتوای فایل را ارسال و در نهایت دستور بستن فایل اعمال شده است:

```
qDebug() << MyText;  
MyFile.close();
```

کد کامل برای این بخش:

```
#include <QCoreApplication>  
#include <QDebug>  
#include <QFile>  
#include <QString>  
#include <QTextStream>  
  
void Read(QString Filename)  
{  
    QFile MyFile(Filename);  
    if(!MyFile.open(QFile::ReadOnly | QFile::Text))  
    {  
        qDebug() << "could not open file for reading";  
        return;  
    }  
    QTextStream in(&MyFile);  
    QString MyText = in.readAll();  
    qDebug() << MyText;  
    MyFile.close();  
}
```

معرفی و کار با فایل‌ها / خواندن و نوشتن در آن‌ها (Write)

در این بخش به نحوه **نوشتن** اطلاعات در فایل‌ها می‌پردازیم که برای شروع لازم است کلاس‌های زیر را فراخوانی کنیم:

```
#include <QFile>
#include <QString>
#include <QTTextStream>

void Write(QString Filename)
{
    QFile MyFile(Filename);

    if (!MyFile.open(QFile::WriteOnly | QFile::Text))
    {
        qDebug() << "could not open file for reading";
        return;
    }

    QTextStream out(&MyFile);
    out << "Hello my friend \n";

    MyFile.flush();
    MyFile.close();
}
```

در این تابع نام آن را **Write** و متغیر ورودی آن را از نوع **QString** با نام **Filename** قرار داده‌ایم و برای اجرای این تابع نیاز است آن را به صورت زیر فراخوانی و مقدار دهی کنیم:

```
Write("/Users/compez/Desktop/test.txt");
```

هنگامی که تابع اجرا شود در صورتی که شرط برقرار نشده باشد و بدون مشکل فایل مورد نظر باز شود، بعد از درج اطلاعات در داخل فایل پیغامی به عنوان موفقیت آمیز بودن عمل چاپ خواهد شد: ☺! "The text is written!" و اگر داخل فایل را باز کنید خواهید دید که متن مقابل در داخل ان نوشته شده است: ☺! Hello my friend و ابتدا برای نوشتن و یا خواندن در داخل یک فایل باید از کلاس **QFile** یک نمونه گرفته شود تا بتوانیم این کار را انجام دهیم پس به صورت زیر ابتدا یه نمونه گرفته ایم:

```
QFile MyFile(Filename);
```

در خط بعدی یک دستور شرطی آورده شده است که اگر فایل را از نوع متنی و در حالت فقط نوشتندی در نظر بگیریم اگر فایل را بر اساس این شرایط تشخیص و قادر به نوشتند را نشود پیغام عدم توانایی در باز کردن و نوشتند در فایل را نمایش خواهد داد بنابراین به صورت زیر خواهیم داشت:

```
if (!MyFile.open(QFile::WriteOnly | QFile::Text))
{
```

```

    qDebug() << "could not open file for reading";
    return;
}

```

در قسمت بعد برای کار با فایل متنی و رشته‌ای باید از کلاس **QTextStream** نمونه‌ای ساخته شود:

```
QTextStream out(&MyFile);
```

حال در قسمت بعد متن مورد نظر را در داخل فایل می‌نویسیم به صورت زیر:

```
out << "Hello my friend!";
```

در قسمت بعد دستور چاپ پیغام نوشته شده است و در نهایت دستور بستن فایل را ارسال می‌کنیم به صورت زیر:

```
qDebug() << "The text is written!";
MyFile.flush();
MyFile.close();
```

قبل از بستن فایل مشخص شده است که، هرگونه متنی را که در فایل باید نوشته می‌شود قبل از آن در حافظه‌ی **Buffer** قرار خواهد داد. در این صورت مقدار ۱ را برگشت خواهد داد و فایل را خواهد بست در غیر اینصورت ۰ و خطا ساطع خواهد شد. تا این بخش فایل و مسیر آن را مشخص نکرده‌ایم برای این کار به صورت زیر عمل می‌کنیم:

```
Write("/Users/compez/Desktop/test.txt");
```

ابتدا نام تابعی که نوشته‌ایم و بعد مسیر آن را همراه با نام و پسوند فایل مشخص کرده و سپس برنامه را اجرا کنید.

کد نهایی:

```

#include <QCoreApplication>
#include <QDebug>
#include <QDir>
#include <QFileInfo>

void Write(QString Filename)
{
    QFile MyFile(Filename);

    if (!MyFile.open(QFile::WriteOnly | QFile::Text))
    {
        qDebug() << "could not open file for reading";
        return;
    }

    QTextStream out(&MyFile);
    out << "Hello my friend \n";

    qDebug() << "The text is written!";
    MyFile.flush();
    MyFile.close();
}
```

```
}
```

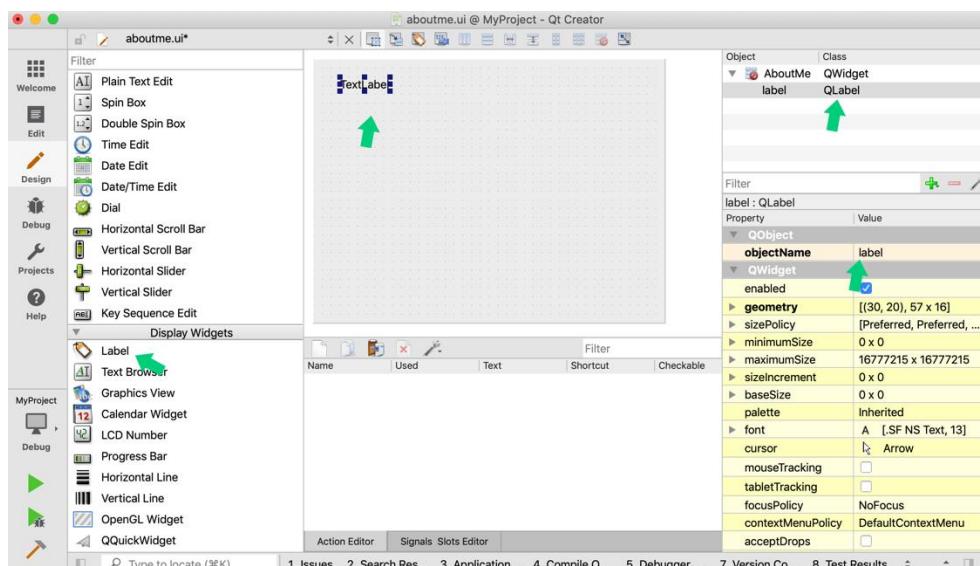
```
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    Write("/Users/compez/Desktop/test.txt");

    return a.exec();
}
```

معرفی و کار با برچسبها

برای نمایش متن یا عنوان بخشی از طرح که شامل تصویر می‌تواند باشد، نیاز است از کنترل برچسب یا همان **Label** که در کتابخانه کیوت با نام **QLabel** مشخص شده است استفاده می‌شود. اسم این برچسب به صورت پیشفرض **label** بوده که برای تغییر آن می‌توانید در قسمت **objectName** نام آن یا هر کنترل دیگری را تغییر دهید.



برای مثال برچسبی را بر روی یک فرم قرار داده و آن را مقدار دهی می‌کنیم. این کنترل دارای چندین گزینه است که یکی از مهمترین آنها **setText** خواهد بود که در این مثال ما قصد داریم توسط این گزینه محتوای برچسب را مشخص نماییم.

```
ui->label->setText("<b>Hello</b> , My name is Kambiz");
```

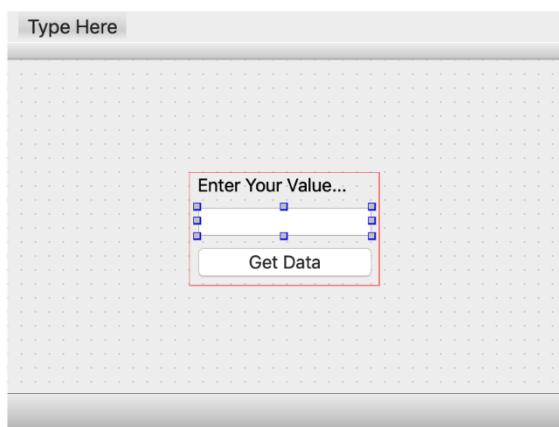
در این کد توسط `ui` که فضای نامی است از کلاس `MainWindow` که با اشاره به آن می‌توانیم به کنترل‌ها یا اشیاء موجود بر روی فرم دسترسی پیدا کنیم؛ بنابراین با اشاره به آن و انتخاب `label` که در این مثال به صورت پیشفرض نام آن مورد استفاده قرارداده شده است را انتخاب و گزینه `setText` آن را انتخاب می‌کنیم که وظیفه‌ی این ویژگی نگه داشتن متن و در کل محتوای کنترل برچسب (label) است که از نوع `QString` است.

معرفی و کار با Button

یکی از پر کاربردترین کنترل‌های طراحی دکمه یا همان `Button` است. مخصوصاً رویداد کلیک شدن آن بسیار کاربرد دارد بنابراین به صورت زیر پروژه را ایجاد و از `QPushButton` استفاده می‌کنیم که در ابتدای آموزش به آن اشاره شده است.

معرفی و کار با LineEdit

کنترل `LineEdit` همان کنترل `TextBox` معروف است که وظیفه‌ی آن گرفتن اطلاعات از کاربر است که دارای گزینه‌های پر کاربردی است.



برای کدنویسی و استفاده از خاصیت دریافت اطلاعات طبق راهنمایی‌های قبلی به رویداد کلیک شدن در `LineEdit` توسط `Go to slot` رفته و قصد داریم هنگام کلیک شدن متنی که کاربر در داخل `Button` می‌نویسید نمایش داده شود بنابراین به صورت زیر خواهد بود:

```
void MainWindow::on_pushButton_clicked()
```

```

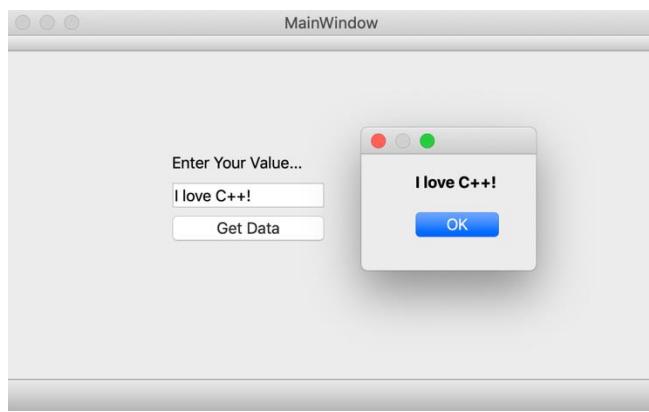
{
    QMessageBox msgBox;
    msgBox.setText(ui->lineEdit->text());
    msgBox.exec();
}

```

با این تفاوت که به جای پیغام قبلی خاصیت متن موجود در **LineEdit** را فراخوانی می‌کنیم با دستور زیر:

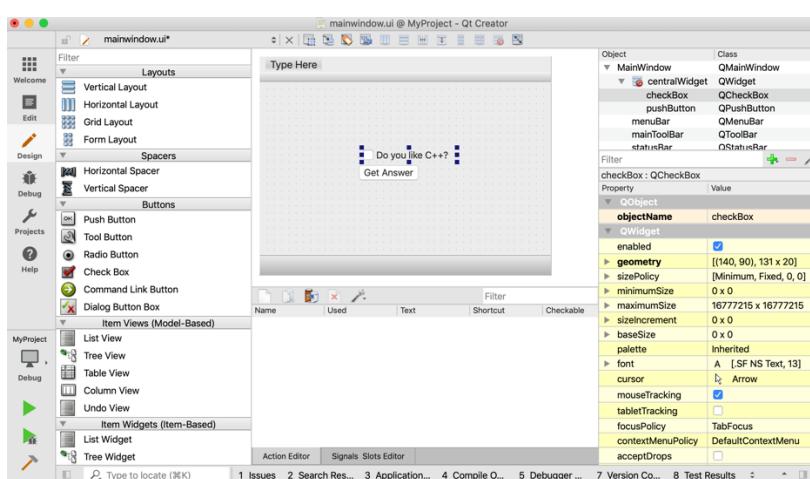
```
ui->lineEdit->text();
```

بعد از اجرای برنامه با وارد کردن متن مورد نظر در **LineEdit** و کلیک بر روی **Run** نتیجه به صورت زیر خواهد بود:



معرفی و کار با **CheckBox**

یکی از کنترل‌های مهم و کار آمد که در بسیاری از موارد که نیاز است گزینه‌ای را انتخاب نماییم از کنترل "کادر انتخاب" یا **CheckBox** استفاده می‌شود.



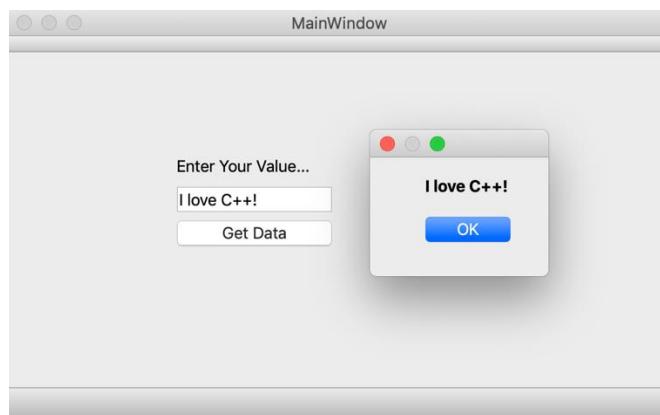
در صورتی که نیاز باشد با انتخاب کادر انتخابی نتیجه‌ی آن مشخص شود در صورت انتخاب شدن و یا نشدن آن جواب بر اساس شرط متفاوت خواهد بود به صورت زیر :

```

void MainWindow::on_pushButton_clicked()
{
    if (ui->checkBox->isChecked())
    {
        QMessageBox msgBox;
        msgBox.setText("Yes I like it :)");
        msgBox.exec();
    }
    else {
        QMessageBox msgBox;
        msgBox.setText("No I don't like it :(");
        msgBox.exec();
    }
}

```

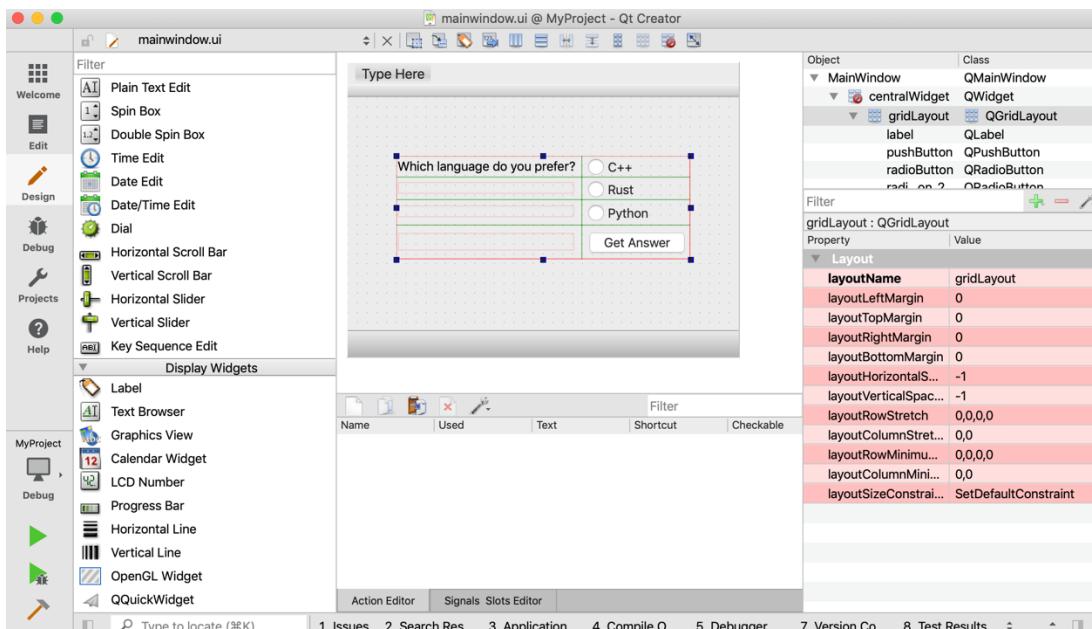
و در نهایت نتیجه به صورت زیر نمایش داده خواهد شد:



*نکته: مشخصه‌ی isChecked در صورتی که انتخاب شده باشد مقدار true را برمی‌گرداند.

معرفی و کار با RadioBox

بکی از کارآمد ترین کنترل‌های انتخابی که معمولاً برای مشخص کردن یکی از گزینه‌های پیشنهادی مانند سوال‌های چند گزینه‌ای که باید یکی از آن‌ها انتخاب شود، استفاده از **RadioBox** است. در فرم زیر سه گزینه، **radioCpp**، **radioPython** و **radioRust** تغییر داده‌ایم.



ابتدا نیاز است هنگام بارگذاری فرم یکی از این گزینه‌ها به صورت پیشفرض انتخاب شده باشند؛ بنابراین برای انتخاب گزینه به صورت پیشفرض از خاصیت **setChecked** استفاده می‌شود.

```
ui->radioCpp->setChecked(true);
```

در این مثال شیء مربوط به **radioCpp** را با مقدار **true** تعریف می‌کنیم که کد آن در بخش سازنده فرم اعمال شده است.

```
MainWindow::MainWindow(QWidget *parent) :  
    QMainWindow(parent),  
    ui(new Ui::MainWindow)  
{  
    ui->setupUi(this);  
    ui->radioCpp->setChecked(true);  
}
```

گزینه‌ی **ui** که قبلاً هم توضیح داده شده است اشاره‌گری به به فرم اصلی یا همان **MainWindow** است. با توجه به آن، کنترل‌های موجود بر روی والد تحت **ui** در دسترس خواهند بود.

در ادامه در رویداد کلیک شدن کنترل Button کد زیر را می‌نویسیم:

```
void MainWindow::on_pushButton_clicked()
{
    if (ui->radioCpp->isChecked() == true)
    {
        QMessageBox msgBox;
        msgBox.setText("Your selected option is : " + ui->radioCpp-
>text());
        msgBox.exec();
    }
    else if (ui->radioRust->isChecked() == true)
    {
        QMessageBox msgBox;
        msgBox.setText("Your selected option is : " + ui->radioRust-
>text());
        msgBox.exec();
    }
    else if (ui->radioPython->isChecked() == true)
    {
        QMessageBox msgBox;
        msgBox.setText("Your selected option is : " + ui->radioPython-
>text());
        msgBox.exec();
    }
}
```

همانطور که در کد مشخص است در ساده‌ترین روش با استفاده از دستور شرطی که اگر کاربر هر کدام از گزینه‌های مربوطه را انتخاب کرد، دستورات مربوط به آن اجرا خواهد شد. گزینه `isChecked` در صورت انتخاب بودن شیء مقدار `true` و در غیر اینصورت مقدار `false` را بر می‌گرداند. همچنین جهت چاپ محتوی گزینه از خاصیت `text` آن استفاده شده است که متناسب با نیاز شما می‌تواند تغییر کند.



معرفی و کار با ComboBox

این کنترل به عنوان نگه‌دارنده‌ی لیستی از داده‌های قابل انتخاب کاربرد بسیار مفیدی دارد، معمولاً در لیست‌داده‌ها بر اساس شاخص (index) مقادیر مربوط به داده‌های خود را برمی‌گرداند. جهت دریافت مقادیر و مشخصه‌های مربوطه می‌توان از طریق گزینه‌های currentIndex و currentText به آن‌ها دسترسی داشت.

برای آزمایش، کنترل مربوطه را بر روی فرم مورد نظر درج کرده و در سازنده‌ی کلاس فرم کدهای داده مربوط به آن را به صورت زیر تعریف می‌کنیم.

```
/*Data*/  
QStringList Language = (QStringList() << "C++" << "Java" << "C#" <<  
"PHP" << "Rust" << "Objective-C");  
ui->comboBox->addItems(Language);
```

با توجه به اینکه این کنترل داده‌ها را به صورت لیست و شاخص‌های مربوطه می‌پذیرد می‌توان از نوع QStringList برای افزودن داده‌های مورد نظر بر روی این کنترل استفاده کرد. بنابراین داده‌های مربوطه را تحت مشخصه‌ی addItems دریافت و اعمال خواهیم کرد.

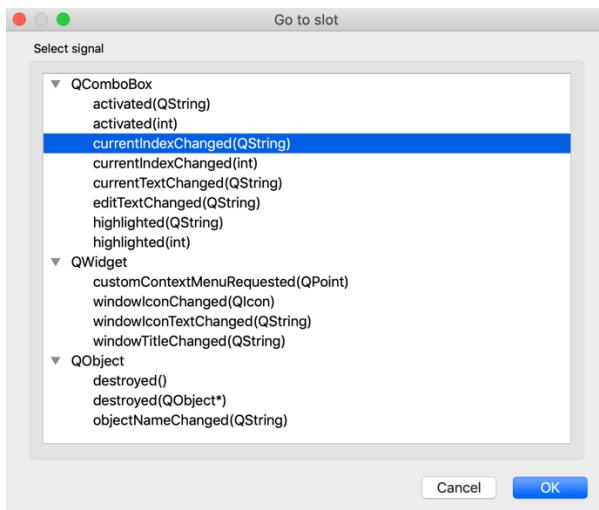
```
ui->comboBox->addItems(Language);
```

فرض بر اینکه نیاز است با انتخاب هر یک از آیتم‌های موجود در ای کنترل مقداری از آن چاپ شود.

```
void MainWindow::on_pushButton_clicked()  
{  
    QString CurrentVal;  
    CurrentVal = ui->comboBox->currentText();  
  
    QMessageBox msgBox;  
    msgBox.setText(CurrentVal);  
    msgBox.exec();  
}
```

در کد فوق متغیری از نوع QString جهت دریافت مقدار از نوع رشته که برابر با نوع currentText از کنترل مربوطه می‌باشد را ایجاد و مقدار آن را برابر با مشخصه‌ی currentText قرار داده‌ایم. با انتخاب هریک از آیتم‌های موجود در کنترل متن (عنوان آیتم) در متغیر مربوطه درج و در نهایت به عنوان پیغام نمایش داده خواهد شد.

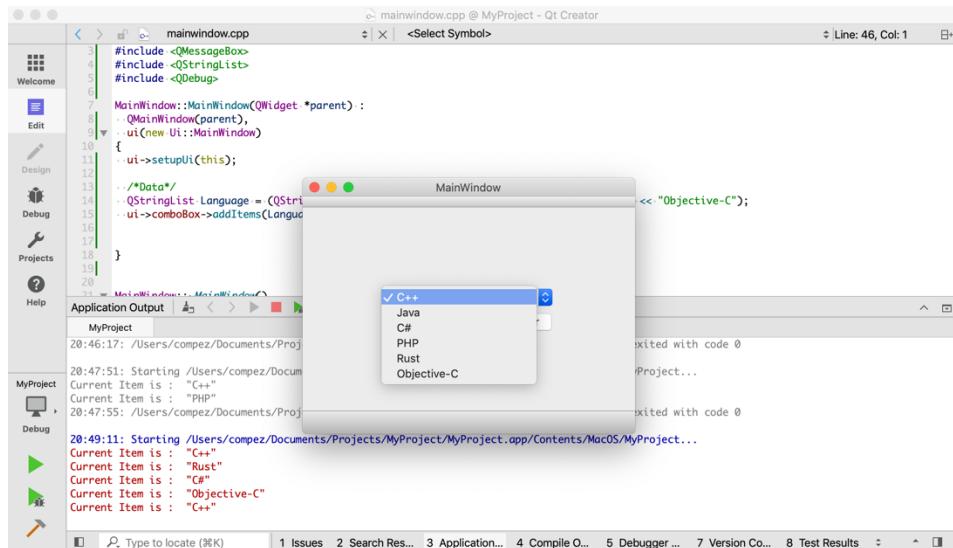
ممکن است نیاز باشد تا بر اساس رویداد تغییر در شاخص و یا مقدار آیتمها عمل خاصی را انجام دهیم، مانند فراخوانی تابع یا چاپ و موارد مورد نیاز دیگر. بنابراین سیگنال‌های پرکاربرد آن `highlighted` و حتی `editTextChanged`، `currentTextChanged`، `currentIndexChanged`



سیگنال‌های مربوطه به عنوان رویداد مورد نظر، از انواع پارامترهای صحیح و رشته‌ای پشتیبانی می‌کند. که در لیست بالا سیگنال‌های فعال این کنترل را در اختیار داریم. برای مثال سیگنال `currentIndexChanged` از نوع ورودی `QString` را برای کار با آن انتخاب می‌کنیم.

```
void MainWindow::on_comboBox_currentIndexChanged(const QString &arg1)
{
    qDebug() << "Current Item is : " << arg1;
}
```

با اجرای برنامه و همچنین تغییر مقادیر موجود در کنترل، مقادیر انتخاب شده در کنسول چاپ خواهند شد.



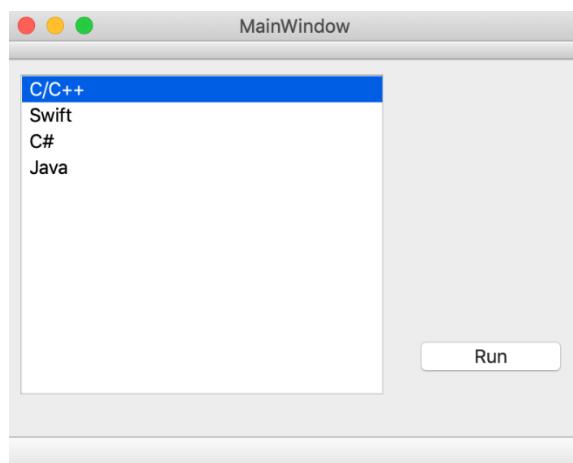
ListWidget و کار با

این کنترل مشابه **ComboBox** عمل می‌کند ولی با این تفاوت که محتویات و آیتم‌های این کنترل به صورت **DataGrid** در دسترس هستند و به صورت یک لیست به آیتم‌ها دسترسی خواهیم داشت چیزی شبیه ولی با خصوصیات منحصر بفرد.

برای درج اطلاعات در این کنترل از نوع لیست، مشابه کنترل قبلی عمل خواهیم کرد:

```
/*Data*/
QStringList C_Types = (QStringList() << "C/C++" << "Swift" << "C#"
<< "Java");
ui->listWidget->addItems(C_Types);
```

نتیجه‌ی این کد در سازنده‌ی کلاس فرم والد بعد از بارگذاری به صورت زیر خواهد بود:



در ادامه لازم است با عمل کلیک بر روی دکمه Run مقدار انتخاب شده در لیست در قالب پیغام به نمایش درآید که کد مربوط به آن به صورت زیر خواهد بود:

```
void MainWindow::on_pushButton_clicked()
{
    QString Val;
    Val = ui->listWidget->currentItem()->text();
    QMessageBox msgBox;
    msgBox.setText(Val);
    msgBox.exec();
}
```

کد کامل به صورت زیر:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QMMessageBox>
#include <QStringList>
#include <QDebug>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    /*Data*/
    QStringList C_Types = (QStringList() << "C/C++" << "Swift" << "C#"
    << "Java");
    ui->listWidget->addItems(C_Types);

}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::changeEvent(QEvent *e)
{
    QMainWindow::changeEvent(e);
    switch (e->type()) {
        case QEvent::LanguageChange:
            ui->retranslateUi(this);
            break;
        default:
            break;
    }
}

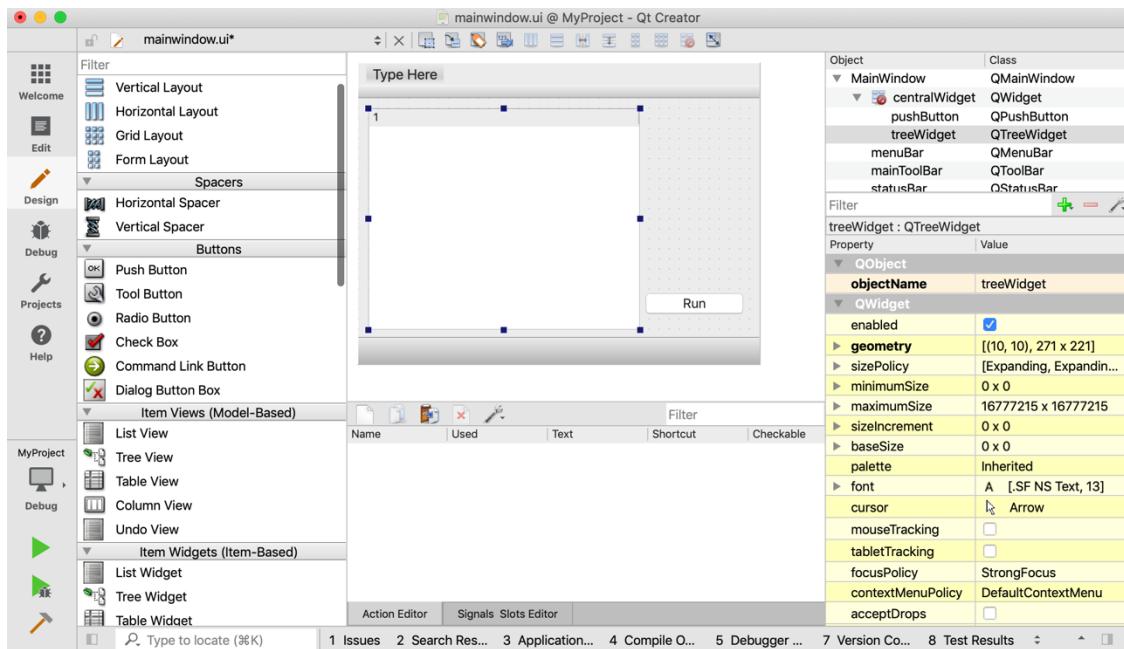
void MainWindow::on_pushButton_clicked()
{
    QString Val;
    Val = ui->listWidget->currentItem()->text();
    QMessageBox msgBox;
    msgBox.setText(Val);
    msgBox.exec();
}
```

لازم است توجه داشته باشیم که می‌توان بر اساس ایندکس به جای currentItem نیز استفاده کرد و کد را به صورت سفارشی به گونه‌ای که لازم است تغییر داد.



معرفی و کار با TreeWidget

معرفی و کار با لیست‌های درختی (Tree Widget) به این صورت است که در بسیاری از مواقع که مقادیر قابل انتخاب حاوی شاخه‌ها و زیر شاخه‌هایی هستند و برنامه‌نویس باید امکان این را برای کاربر فراهم کند تا شاخه‌ها و زیر شاخه‌های بخشی از آن را که کاربر می‌خواهد انتخاب کند.



در رابطه با این شیئی باید اینگونه توضیح داد که خاصیت والد و فرزند بودن را ارائه می‌کند. برای مثال، کنترل TreeWidget را بر روی فرم قرار داده و نام آن را به MyTreeWidget تغییر دهید. برای شروع مثال ساختار زیر را در نظر بگیرید:

• کتاب C++

◦ فصل اول

◦ فصل دوم

◦ و

دققت کنید که در این مثال یک والد (زیرشاخه) داریم و دیگر موارد زیر شاخه‌ی همان والد (فرزنده) هستند.
در اصل همان Child و Parent.

ابتدا برای دسترسی و راحتی کار دو عدد تابع تعریف می‌کنیم برای parent و child در فایل هدر مربوط به MainWindow که به صورت mainwindow.h مشخص شده است ایجاد کنید:

```
void AddRoot(QString name, QString Discreption);
```

```
void AddChild(QTreeWidgetItem * parent, QString name, QString Discreption);
```

در این بخش متدها و همچنین نام و نوع تابع را مشخص کرده‌ایم که برای AddRoot یا همان والد نام و توضیحات را در نظر می‌گیریم و برای تابع AddChild یا همان افزودن فرزند علاوه بر نام و توضیحات هر یک شناسه تعیین کننده برای معرفی آیتم مورد نظر برای سر شاخه و آن را از نوع QTreeWidgetItem قرار داده‌ایم. در بخش بعدی در فایل اصلی mainwindow.cpp توابع را صدا زده و بدنه توابع را می‌سازیم به صورت زیر:

```
void MainWindow::AddRoot(QString name, QString Discreption) {}  
void MainWindow::AddChild(QTreeWidgetItem* parent, QString name,  
QString Discreption) {}
```

در ادامه برای تکمیل کدهای درونی این دو تابع به صورت زیر عمل خواهیم کرد:

```
void MainWindow::AddRoot(QString name, QString Discreption)  
{  
    QTreeWidgetItem * MyItem = new QTreeWidgetItem(ui->MyTreeWidget);  
  
    MyItem->setText(0, name);  
    MyItem->setText(1, Discreption);  
    ui->MyTreeWidget->addTopLevelItem(MyItem);  
  
    AddChild(MyItem, "C++", "Qt Widgets");  
    AddChild(MyItem, "C++", "Qt Quick");  
  
}  
  
void MainWindow::AddChild(QTreeWidgetItem* parent, QString name,  
QString Discreption)  
{  
  
    QTreeWidgetItem * MyItem = new QTreeWidgetItem();  
    MyItem->setText(0, name);  
    MyItem->setText(1, Discreption);  
    parent->addChild(MyItem);  
  
}
```

در تابع اول AddRoot ابتدا یک کپی از QTreeWidgetItem می‌سازیم تا به عنوان آیتم‌های درختی بکار گرفته شود و در مرحله‌ی بعدی که در شاخص اول یعنی ۰ متنی را که تابع دریافت می‌کند مشخص کرده‌ایم و در خط بعد از آن مشخص شده است که شاخص بعد از آن یعنی ۱ را مشخص کند و در نهایت گفته شده

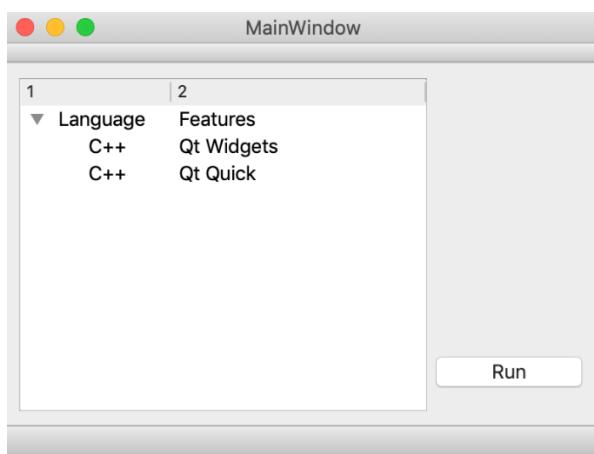
است که لیست درختی را که شامل این دو گزینه است را از نوع آیتم‌های سر شاخه قرار بدهد که با دستور `addTopLevelItem` این کار را انجام داده‌ایم.

در خط بعدی `AddChild` برابر با آیتم‌های تعریف شده قرار گرفته است و به صورت بالا که از تابع `AddChild` فراخوانی می‌شوند به صورت ... سه نقطه، خط اول در تابع `AddChild` مثل تابع قبلی برای عنوان و شناسایی اینها است و در خط آخر اشاره شده است که آیتم‌های زیر مجموعه برابر باشد با آیتم‌های فرزند که ایجاد شده است. در ادامه در بخش بارگذاری فرم لازم است که تعداد ستون‌ها را همراه با عناوین آن‌ها را مشخص کنیم

به صورت زیر:

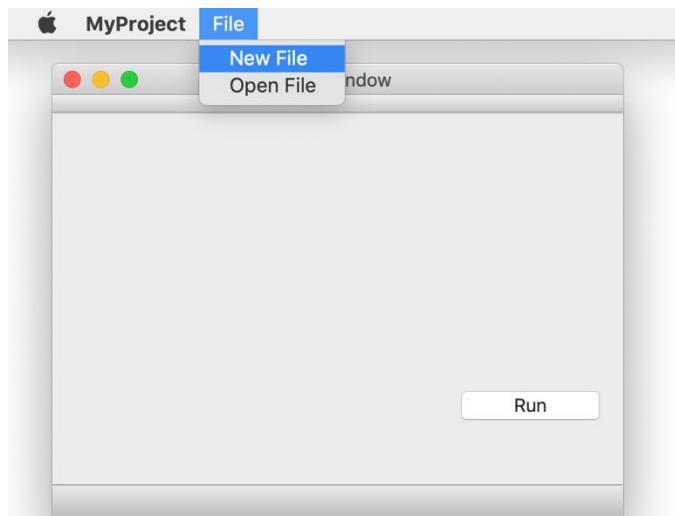
```
ui->MyTreeWidget->setColumnCount(2);  
AddRoot("Language", "Features");
```

دقیق کنید که دو ستون از نوع زبان و ویژگی‌های زبان درج شده است و بعد از این تابع `Addroot` فراخوانی شده است و تابع `AddChild` نیز همانطور تعریف شده است که در نهایت با اجرای برنامه نتیجه‌ی آن به صورت زیر نمایان خواهد شد:

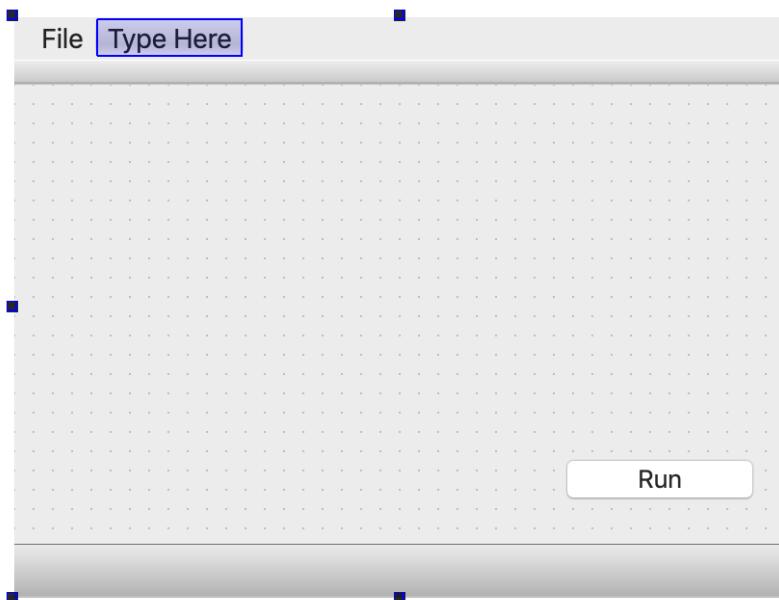


معرفی و کار با Action‌ها

در رابطه با اکشن‌ها می‌توان گفت همان منوهای برنامه هستند که توسط هر اکشن ایجاد شده یک منو با مشخصه خاص تعریف و قابلیت‌ها و همچنین سیگنال (رویدادهای) خاص خودشان را ایجاد می‌کنند برای مثال برای اینکه روی فرم منوهای زیر را به صورت زیر ایجاد کنیم:



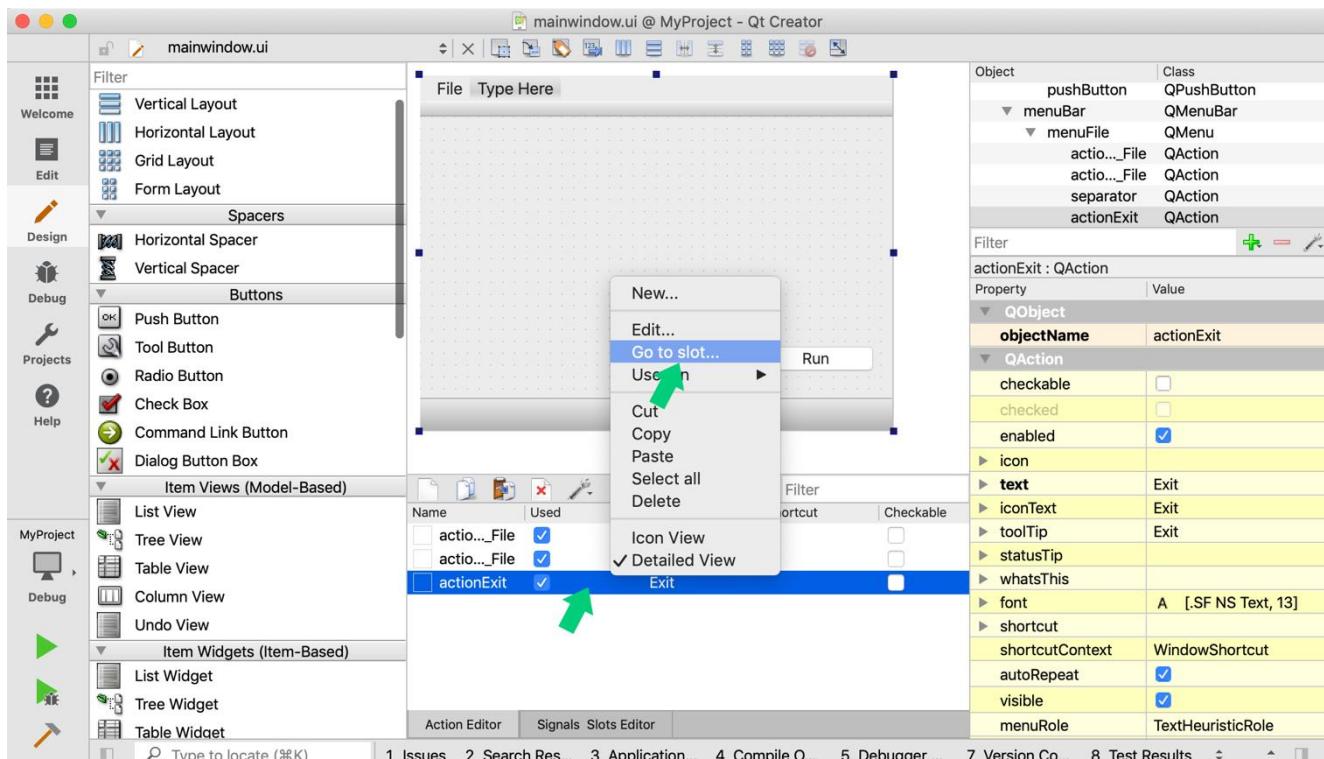
ساده‌ترین روش برای ایجاد منو و زیر منوها کلیک کردن بر روی گزینه‌ی Type Here است که با کلیک بر روی آن و وارد کردن متن مورد نظر اقدام به ساخت منو و زیرمنوهای آن نماییم.



توجه کنید که هر والد می‌تواند زیر شاخه‌های خود را داشته باشد همچنین می‌توانید با کلیک بر روی Add خط فاصله‌ی بین منوها را به عنوان جدا کننده را ایجاد کند.

هریک از منوهایی که ایجاد می‌شوند به عنوان یک Action است، مانند دیگر کنترل‌هایی که دارای رویداد و اسلات‌های خاص خودشان هستند. حال برای اینکه با کلیک کردن بر روی دکمه Exit یا همان منوی ساخته شده با نام Exit عملی را انجام دهد چگونه باید عمل کنیم؟

ابتدا باید در رویداد کلیک شدن یا انتخاب شدن این Action که در اینجا با شناسه‌ی actionExit در دسترس است وارد شده و کدهای مورد نظر را برای انجام کار مورد نظر بنویسیم به صورت زیر قسمت اکشها و استلات‌ها مشخص شده است:



در ادامه با وارد شدن به اسلات مورد نظر و ایجاد تابع رخداد triggered مراحل زیر را ادامه می‌دهیم:

```
void MainWindow::on_actionExit_triggered() {
}

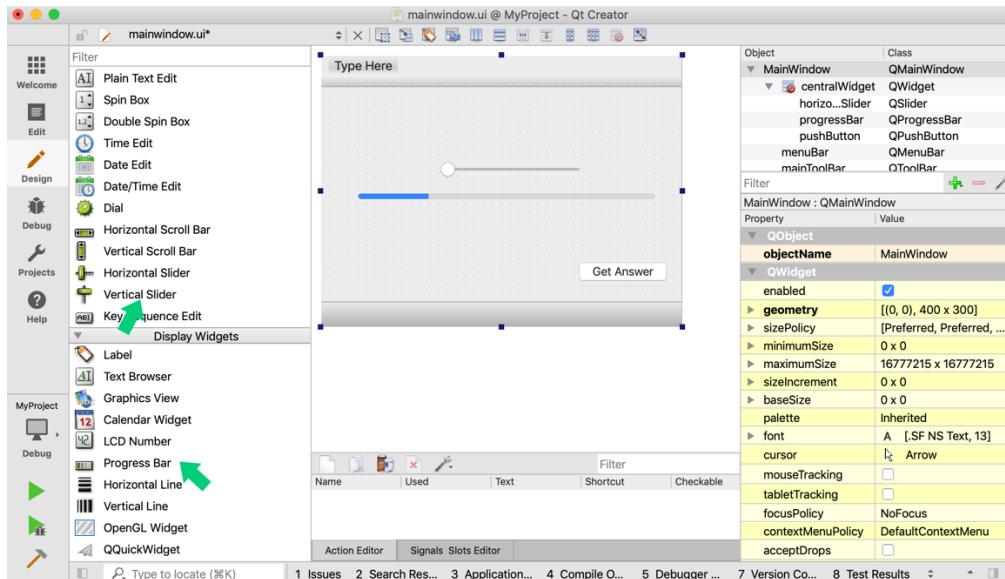
void MainWindow::on_actionExit_triggered()
{
    QApplication::exit();
}
```

کد مربوط به عمل خروج از برنامه که مختص کتابخانه کیوت است.

نتیجه‌ی این کد خروج برنامه‌های تحت کیوت می‌باشد. در واقع بعد از کلیک بر روی آن برنامه از پروسه‌ی اجرایی در لیست پروسه‌های سیستم عامل خارج خواهد شد.

معرفی و کار با Slider و Progress ها

در رابطه با کنترل های Slider و Progress چون مشخصه های تقریباً مشابهی دارند این دو کنترل را باهم در این مثال ادغام می کنیم. بنابراین فرم مربوطه شامل یک Slider و یک Progressbar خواهد بود.



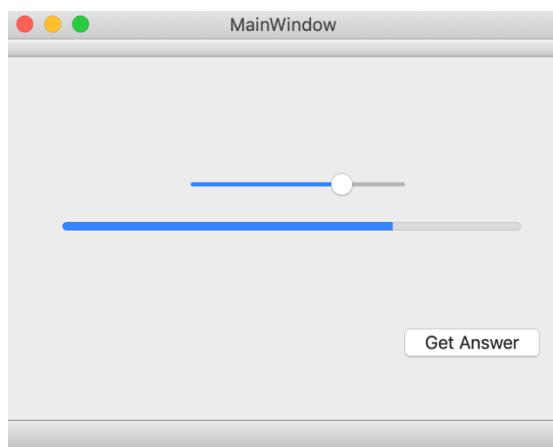
در رابطه با شیء Progressbar باید بگوییم که بر اساس مشخصه value مقدار دهی می شوند یعنی اگر ما برای این کنترل Maximum value را برابر با مقدار ۱۰۰ قرار دهیم و مشخصه value آن برابر باشد با ۲۴ آن زمان از ۱۰۰ درصد مقدار این کنترل ارزش دهی می شود که در تصویر نیز مشخص شده است، البته باید اشاره کرد که این کار را می توان به صورت کدنویسی انجام داد.

قبل از شروع کدنویسی این بخش ابتدا شناسه های این دو شیء را مشخص می کنیم شناسه MyhorizontalSlider را به Slider و شناسه MyprogressBar را به Progressbar تغییر داده و شناسه MyhorizontalSlider را به Sliderbar تغییر کند مقدار Progressbar هم بر اساس آن نیز تغییر خواهد کرد.

با توجه به اینکه در Qt بحث مکانیزم Signal و Slot مطرح می شود که برای ارتباط بین اشیاء استفاده می شود. به طور خلاصه فرض کنید لازم است به یک شخص ایمیل ارسال کنیم و در قبال آن ایمیل انتظار دریافت جواب را خواهیم داشت بنابراین ارسال کننده در این بخش به عنوان Signal و شخص دریافت کننده ایمیل به عنوان Slot است که ارتباط بین این دو مورد توسط تابعی با نام connect که وظیفه این تابع برقراری ارتباط بین Signal و Slot است را در اختیار داریم.

```
connect(ui->MyhorizontalSlider, SIGNAL(valueChanged(int)),  
ui->MyprogressBar, SLOT(setValue(int)));
```

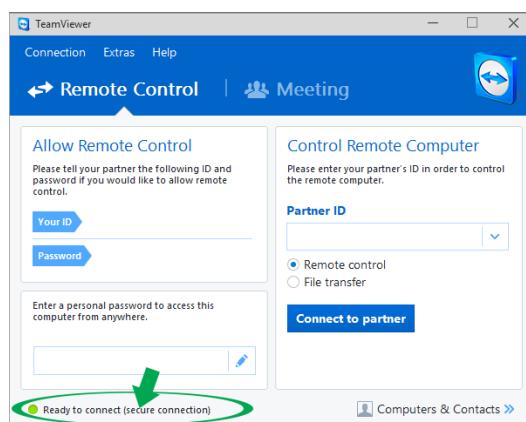
در این کد تابع `connect` فراخوانی شده است این تابع ابتدا `Signal` را گرفته و بعد `Slot` را برای سیگنالی که ارسال کرده است تنظیم می‌کند به طور خلاصه وظیفه‌ی آن این است سیگنال را ارسال کرده و اسلات را برای سیگنال ارسال شده دریافت کند. در این مثال مقدار ارزشی که ما ارسال و دریافت می‌کنیم از نوع `int` است که در هر دو شیء ما مقدار `value` ارسال و دریافت می‌شود. حال بعد از اجرا کردن برنامه هر بار من مقدار Slider را تغییر دهیم مقدار Progressbar هم تغییر خواهد یافت:



در رابطه با تنظیم کردن جدایانه مقدار در کنترل Progressbar برای این کار از مشخصه‌ی `setValue` باید استفاده شود.

معرفی و کار با Statusbarها

معمولًاً در طراحی رابط کاربری، بخشی در نرم‌افزارها به عنوان نوار وضعیت یا همان `StatusBar` وجود دارد که می‌تواند شامل اشیاء مختلفی باشد. برای مثال به صورت زیر نرم‌افزار `TeamViewer` در قسمت پایین آن بخشی مشخص شده است که توسط یک متن وضعیت برنامه را مشخص می‌کند که اصولاً در این موارد از `StatusBar` استفاده می‌شود.



این امکان در چهارچوب خود Qt کیوت موجود است که همه‌ی Widgets با صورت پیشفرض از این امکانات پشتیبانی کنند. یک نوع برچسب به عنوان خوش آمدگویی بر روی ویجت ایجاد کنید:

```
QLabel *Mylbl = new QLabel();  
Mylbl->setText("Welcome to C++ Qt Designer tutorial");
```

در این کد نمونه‌ای از کلاس QLabel را ایجاد کرده و محتوای آن که از نوع رشته است را مشخص می‌کنیم؛ سپس برای افزودن کنترل برچسب به Statusbar موجود در ویجت کد زیر را خواهیم نوشт:

```
ui->statusBar->addWidget(Mylbl);
```

دستور **addWidget** از Statusbar را به آن اضافه می‌کند و با اجرای برنامه نتیجه به صورت زیر خواهد بود:



MessageBox و کار با

یکی از مواردی که در بسیاری از برنامه‌ها مشاهده می‌شود مخصوصاً برای نمایش پیغام‌های خاص مانند (اخطر، اطلاعات، سوال و جواب و ...) استفاده از **MessageBox** است که در C++ در حالت‌های مختلفی می‌شود از آن استفاده کرد.

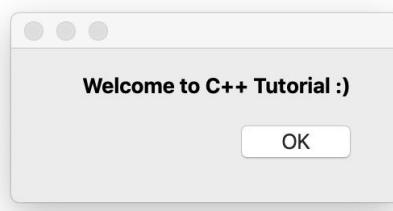
برای اینکه از این امکان استفاده کنیم ابتدا فایل سرآیند آن را وارد برنامه می‌کنیم به صورت زیر:

```
#include <QMessageBox>
```

حال ما می‌توانیم از آن یک نمونه ساخته و آن را مورد استفاده قرار دهیم:

```
QMessageBox MyMSG;  
MyMSG.setText("Welcome to C++ Tutorial : )");  
MyMSG.exec();
```

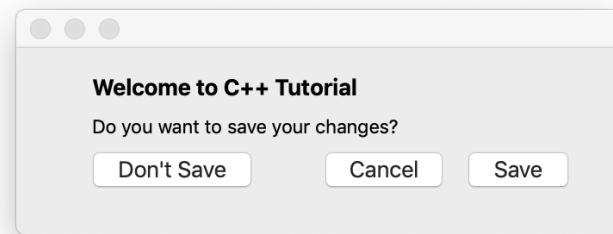
در خط اول از کلاس **QMessageBox** یک نمونه ساخته و در خط دوم متنی را درون این آبجکت وارد کرده‌ایم و در خط بعدی از متدهای **exec** برای نمایش **Dialog** استفاده کرده‌ایم که نتیجه آن باید به صورت زیر باشد:



در ادامه با خصیصیت‌های این کلاس بیشتر آشنا خواهیم شد. برای مثال می‌خواهیم از این قابلیت برای پرسش سوال استفاده کنیم و نتیجه آن را با استفاده از Yes یا No مشخص کنیم به صورت زیر کد را خواهیم داشت:

```
QMessageBox msgBox;  
msgBox.setText("Welcome to C++ Tutorial");  
msgBox.setInformativeText("Do you want to save your changes?");  
msgBox.setStandardButtons(QMessageBox::Save |  
QMessageBox::Discard | QMessageBox::Cancel);  
msgBox.setDefaultButton(QMessageBox::Save);  
int ret = msgBox.exec();
```

نمونه‌ی مربوطه ایجاد و متن مورد نظر آن برای نمایش به عنوان سوال تحت خاصیت `setText` و توضیحات جزئی آن را توسط مشخصه‌ی `setInformativeText` اعمال شده است. در خط بعدی توسط مشخصه‌ی `setStandardButtons` دکمه‌هایی را که می‌خواهیم کاربر هنگام پرسش سوال مشاهده نماید مشخص می‌کنیم که در اینجا سه گزینه `Save / Cancel / Discard` را بکار گرفته شده است. در خط بعد مشخص شده است که دکمه `Save` به عنوان دکمه پیشفرض در حالت انتخاب شده قرار خواهد گرفت. در نهایت پیغام را نمایش داده و مقدار بازگشتی حاصل از انتخاب کاربر را به متغیر `ret` که از نوع `int` است ارسال می‌کند.



هنگامی که کاربر هر کلیدی را که انتخاب می‌کند باید مقداری را بازگشت داده و کاری را انجام دهد برای مثال هرکی از دکمه‌ها قرار است هنگام فشرده شدن عملیات متفاوتی را انجام دهد؛ بنابراین برای این کار به روش زیر عمل می‌کنیم:

دستور `Switch` یکی از روش‌های معروفی است که در موقع پیچیده‌تر و چند گزینه‌ای پیشنهاد می‌شود. متغیر `ret` که با انتخاب هریک از دکمه‌ها توسط دستور `switch` ارزیابی می‌شود و نتیجه بر اساس سلیقه مشخص خواهد شد. برای مثال قرار است با انتخاب هریک از آن‌ها کد نتیجه بر اساس کدی که بازگشت داده می‌شود در یک برچسب نمایش داده شود:

```
switch (ret) {
    case QMessageBox::Save:
        // Save was clicked
        ui->MyLabel->setText("Save");
        break;
    case QMessageBox::Discard:
        // Don't Save was clicked
        ui->MyLabel->setText("Discard");

        break;
    case QMessageBox::Cancel:
        // Cancel was clicked
```

```

ui->MyLabel->setText ("Cancel");

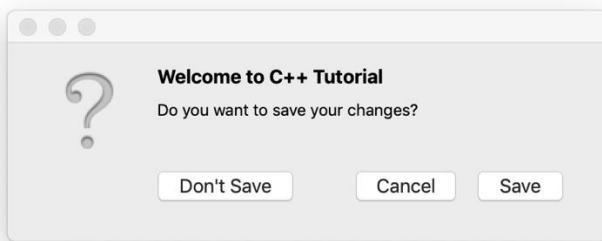
break;
default:
// should never be reached
break;
}

```

در ادامه یک مشخصه‌ی مهم دیگری که این کلاس دارد به عنوان ICON است که برای این کار هم به صورت

زیر عمل کنید:

```
msgBox.setIcon (QMessageBox::Question);
```



به طور کلی کلاس QMessageBox دارای خصوصیات زیادی است که به عنوان نمونه خاصیت‌های مربوط به

آن به صورت زیر آمده است:

توضیحات	ارزش	مقدار ثابت
جعبه پیغام دارای هیچ آیکونی نخواهد بود.	0	QMessageBox::None
یک آیکون به عنوان سوال پرسیده شده نمایش داده می‌شود.	4	QMessageBox::Question
یک پیغام از نوع اطلاعات رسان که نشانگر یک پیغام عادی است.	1	QMessageBox::Information
یک آیکون جهت اخطار نمایش داده خواهد شد که می‌تواند مهم باشد.	2	QMessageBox::Warning
یک آیکون در رابطه با موقعیت بحرانی نمایش خواهد داد.	3	QMessageBox::Critical

توضیحات	ارزش	مقدار ثابت
یک دکمه نامعتبر.	-1	QMessageBox::InvalidRole
با کلیک بر روی دکمه باعث می‌شود گفتگو پذیرفته شود (به عنوان مثال OK)	0	QMessageBox::AcceptRole
با کلیک بر روی دکمه باعث می‌شود گفتگو پذیرفته نشود (به عنوان مثال Cancel)	1	QMessageBox::RejectRole
با کلیک بر روی دکمه باعث تغییر در آن می‌شود (به عنوان مثال رد کردن یا بی عمل)	2	QMessageBox::DestructiveRole
با کلیک بر روی دکمه باعث تغییر در عناصر آن می‌شود.	3	QMessageBox::ActionRole
با کلیک بر روی دکمه باعث درخواست کمک یا همان "Help" می‌شود.	4	QMessageBox::HelpRole
دکمه ای شبیه Yes را شناسایی می‌کند.	5	QMessageBox::YesRole
دکمه ای شبیه No را شناسایی می‌کند.	6	QMessageBox::NoRole
دکمه ای اعمال تغییرات جاری را تعریف می‌کند.	8	QMessageBox::ApplyRole
مقدار دهی عناصر و فیلدها به صورت از پیش تعریف شده و پیشفرض را تعریف می‌کند.	7	QMessageBox::ResetRole

خصوصیات مربوط به دکمه‌های سفارشی در آن به صورت زیر است:

توضیحات	ارزش	مقدار ثابت
یک دکمه "OK" که توسط شرایط "AcceptRole" تعریف می‌شود.	0x00000400	QMessageBox::Ok
یک دکمه "Open" که توسط شرایط "AcceptRole" تعریف می‌شود.	0x00002000	QMessageBox::Open
یک دکمه "Save" که توسط شرایط "AcceptRole" تعریف می‌شود.	0x00000800	QMessageBox::Save
یک دکمه "Cancel" که توسط شرایط "RejectRole" تعریف می‌شود.	0x00400000	QMessageBox::Cancel
یک دکمه "Close" که توسط شرایط "RejectRole" تعریف می‌شود.	0x00200000	QMessageBox::Close
یک دکمه "Don't Save" یا همان "Discard" که توسط شرایط "DestructiveRole" تعریف می‌شود.	0x00800000	QMessageBox::Discard
یک دکمه "Apply" که توسط شرایط "ApplyRole" تعریف می‌شود.	0x02000000	QMessageBox::Apply

یک دکمه "Reset" که توسط شرایط "ResetRole" تعریف می‌شود.	0x04000000	QMessageBox::Reset
یک دکمه "RestoreDefaults" که توسط شرایط "ResetRole" تعریف می‌شود.	0x08000000	QMessageBox::RestoreDefaults
یک دکمه "Help" که توسط شرایط "HelpRole" تعریف می‌شود.	0x01000000	QMessageBox::Help
یک دکمه "Save All" که توسط شرایط "AcceptRole" تعریف می‌شود.	0x00001000	QMessageBox::SaveAll
یک دکمه "Yes" که توسط شرایط "YesRole" تعریف می‌شود.	0x00004000	QMessageBox::Yes
یک دکمه "YesToAll" که توسط شرایط "YesRole" تعریف می‌شود.	0x00008000	QMessageBox::YesToAll
یک دکمه "No" که توسط شرایط "NoRole" تعریف می‌شود.	0x00010000	QMessageBox::No
یک دکمه "NoToAll" که توسط شرایط "NoRole" تعریف می‌شود.	0x00020000	QMessageBox::NoToAll
یک دکمه "Abort" که توسط شرایط "RejectRole" تعریف می‌شود.	0x00040000	QMessageBox::Abort
یک دکمه "Retry" که توسط شرایط "AcceptRole" تعریف می‌شود.	0x00080000	QMessageBox::Retry
یک دکمه "Ignore" که توسط شرایط "AcceptRole" تعریف می‌شود.	0x00100000	QMessageBox::Ignore
یک دکمه نامعتبر.	0x00000000	QMessageBox::NoButton

معرفی و کار با Timer / QTimer

کلاس QTimer یک تایمر تکراری و تک-شات را فراهم می‌کند. برای مثال قرار است تحت آن ساعت سیستم را گرفته و تغییر ساعت، دقیقه و ثانیه را به صورت به روز و لحظه‌ای نمایش دهیم. برای این کار ابتدا کلاس QTime و QTimer را وارد برنامه می‌کنیم.

```
#include <QTimer>
#include <QTime>
```

می‌توانیم تابعی برای به روز رسانی مشخص کنیم، اما در حالت کنونی کد زیر را کافی است در بخش سازنده‌ی کلاس فرم والد خود بسازید:

```
QTimer *timer = new QTimer(this);
timer->setInterval(1000);
timer->start();
connect(timer, SIGNAL(timeout()),
```

```
SLOT(updateClock());
```

در این کد ابتدا یک نوع کپی از تایمر `QTimer` ایجاد شده است در خط بعد، مقدار ۱۰۰۰ میلی ثانیه یا همان ۱ ثانیه را به آن اختصاص داده شده است که بعد از آن در خط سوم فرایند آغاز اعمال شده است. در خط بعدی از تابع `connect` برای ارسال `Signal` و دریافت جواب از طرف `Slot` را مشخص می‌شود که برای ایجاد اسلات نیاز است تا با یک تابعی به نام `updateClock` که زمان سیستم را برای برگشت دهد.

بنابراین در فایل `mainwindow.h` اسلات (`Slot`) مربوط به این تابع را معرفی می‌کنیم به صورت زیر:

```
//System Clock update
private slots:
void updateClock();
```

نوع تابع را از نوع تابع با دسترسی `private` به عنوان `Slot` در نظر می‌گیریم و در فایل `mainwindow.cpp` بندۀ تابع را کامل خواهیم کرد به صورت زیر:

```
void MainWindow::updateClock()
{
    QString timeString = QTime::currentTime().toString("hh:mm:ss");
    ui->LCurrenTtime->setText(timeString);
}
```

بعد از اعلان و تعریف بدن، یک متغیر از نوع رشته `QString` ایجاد کرده و بعد توسط تابع موجود در کلاس `QTime` که وظیفه‌ی آن برگشت دادن زمان جاری سیستم است که توسط مقدار بازگشتی ثابت (`Static`) به صورت `currentTime` را وارد متغیر از نوع رشته کرده‌ایم. البته قبل از آن خروجی آن تبدیل به رشته شده و توسط الگوی `hh:mm:ss` ساعت : دقیقه : ثانیه ارسال می‌شود به داخل متغیر و در خط بعدی مقدار متن موجود بر روی فرم را که شناسه‌ی آن `LCurrenTtime` است برابر با متغیری می‌باشد که زمان جاری `label` موجود در آن ذخیره خواهد شد. حال با اجرای برنامه بعد از بارگذاری فرم، ساعت سیستم دقیقاً قابل مشاهده سیستم در آن ذخیره خواهد شد. و بر حسب ثانیه به روز رسانی خواهد شد.

معرفی و کار با Threadها

Thread چیست و چه کارهایی را انجام می‌دهد؟

به طور کلی در نظر بگیرید دستگاه‌های صوتی یا نوارهای ویدئویی قدیمی را که از فناوری Caset استفاده می‌کنند؛ در این حالت شما وقتی موزیک یا فیلمی را تماشا می‌کنید اگر بین فیلم کنونی نیاز داشته باشید تا چند ترک از آن را به جلو هدایت کنید و چند فیلم دیگر را ببینید مجبور هستید تا عمل Seek را انجام دهید، یعنی جلو کشیدن ترک تا زمانی که به فیلم یا بخش مورد نظرتان برسید. در این حالت عمل ترتیبی برای رسیدن به هدف صورت گرفته یک نوع جستجوی ترتیبی برای رسیدن به هدف و اجرای آن است.

حال ما زمانی که پشت کامپیوتر هستیم و انتظار آن را داریم که چندین کار را هم زمان انجام دهیم، برای مثال مایل هستیم یک طرف برنامه کامپایل کرده و در طرف دیگر موزیک گوش دهیم، یک طرف دیگر هم نیز برنامه‌های Messenger آنلاین استفاده کنیم و ... در اینجا کسی فرصت این را نخواهد داشت که همه این کارها را پشت سر هم انجام دهد. یا در واقع، اول پیغام داده، کمی صبر کند تا عملیات پیغام به اتمام رسیده و سپس اقدام به پخش موزیک مورد نظر کند! در همین لحظه است که بحث Thread را وارد برنامه‌ی ما خواهد شد و مدیریت وظایف را به صورت هم زمان انجام خواهد داد.

توجه داشته باشید که، سرعت کامپیوترها به اندازه‌ای زیاد است که این پروسه‌ها در پیش‌زمینه‌ی پردازشی صورت می‌گیرند و شما احساس نخواهید کرد که کدام رشته در چه زمانی اجرا می‌شود! اما نتیجه‌ای که خواهید دید این است که مواردی را که برای پردازش به طرف پردازندۀ مرکزی فرستاده‌ایم همه در حال اجرا هستند چون در کل CPU بر اساس Thread‌ها برنامه‌ها را اولویت‌بندی خواهد کرد.

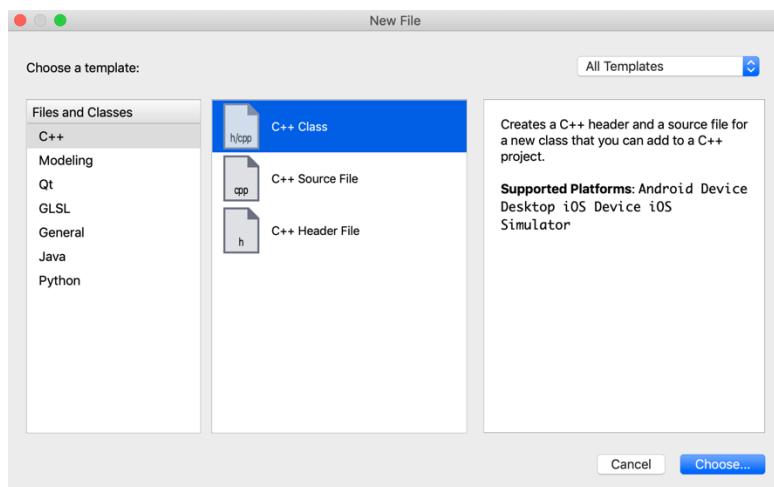
معمولًاً تعداد بسیاری از برنامه‌ها توسط سیستم‌عامل در حال اجرا هستند که هر کدام از آن‌ها حافظه‌ی مورد نیاز خود را مصرف و عملیات متفاوتی انجام می‌دهند که اگر چنین موارد را توسط Thread‌ها مدیریت نکنیم CPU قادر به پردازش هریک از درخواست‌های متفاوت و مختص همه‌ی این برنامه‌های در حال اجرا به صورت هم زمان را نخواهد بود. بنابراین، در حین اجرا برنامه‌ها کرش (هنگ) خواهند کرد! بنابراین استفاده از Thread‌ها در برنامه‌نویسی امری است ضروری!

به طور کلی هر کدام از برنامه‌های در حال اجرا بر اساس اولویت‌بندی Thread آن برنامه در اختیار CPU قرار می‌گیرد یعنی اگر اولویت برنامه Visual Studio بیشتر از برنامه مثلا Qt Creator باشد و در این صورت اگر

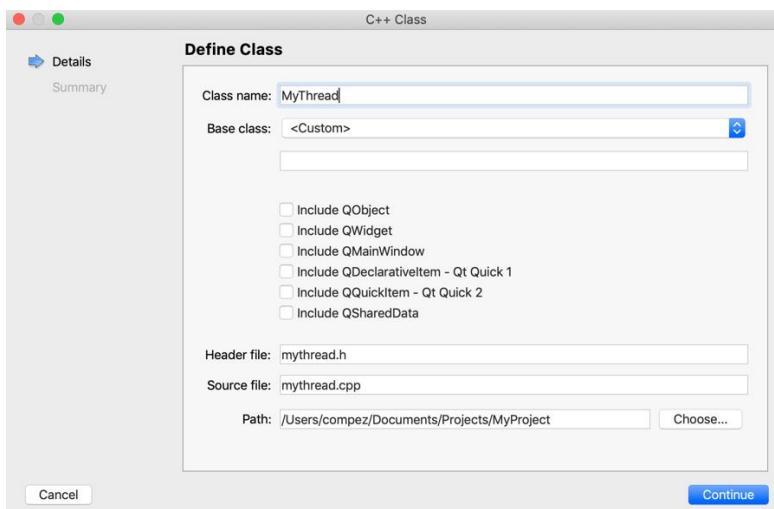
همزمان بر روی هر دو برنامه‌ای را کامپایل کنیم، آن برنامه‌ای که اولویت پردازش روی آن بالاتر است سریع‌تر و ابتدا در اختیار پردازش برای CPU ارسال می‌شود که این یکی دیگر از فواید Thread همین موضوع است.

به طور خلاصه، هر زمانیکه بخواهیم بیش از یک کار "وظیفه" را به طور هم زمان پیش ببریم بهتر است استفاده از Thread‌ها در برنامه خود فراموش نکنیم.

برای شروع این کار بهتر است کلاسی ایجاد کنیم، ابتدا یک پروژه از نوع QConsole Application و بعد به صورت زیر روی پروژه راست کلیک کنید و new Add گزینه C++ Class را انتخاب می‌کنیم:



سپس نام کلاس را MyThread انتخاب کرده و آن را ایجاد می‌کنیم.



در ادامه فایل‌های mythread.h و mythread.cpp مربوط به این کلاس ایجاد خواهند شد که هر کدام شامل کدهای اولیه خواهند بود:

کد مربوط به فایل h. به صورت زیر تولید شده است:

```
#ifndef MYTHREAD_H
```

```
#define MYTHREAD_H

class MyThread
{
public:
    MyThread();
};

#endif // MYTHREAD_H
```

و در نهایت کد مربوط به فایل .cpp. به شکل زیر خواهد بود:

```
#include "mythread.h"

MyThread::MyThread()
{ }

}
```

در این بخش ما کلاسی داریم با نام MyThread از نوع public که برای شروع کار با کلاس‌های Thread به کلاس Thread نیاز داریم و نیاز است به روش زیر فراخوانی شود:

```
#include <QThread>
```

لازم است برای دسترسی عمومی کلاس را به صورت public از نوع QThread در نظر بگیریم :

```
class MyThread : public QThread
{
public:
    MyThread();
};

}
```

برای ایجاد یک تابع در داخل این کلاس با عنوان Run می‌بایست به صورت زیر عمل کنیم :

```
//Function run
void run();
```

به این تابع برای اجرای Thread نیاز خواهیم داشت بنابراین در ادامه به صورت زیر در فایل .cpp. بدنه تابع را کامل می‌کنیم:

```
void MyThread::run()
{
    qDebug() << "My Thread is running !";
}
```

فراخوانی تابع برگرفته شده از کلاس MyThread و ایجاد بدنی تابع با محتویات چاپ این برنامه در حال اجرا است!

نحوه‌ی صدا زدن تابع در برنامه به شکل زیر خواهد بود، که به ابتدا فایل بدن را فراخوانی خواهیم کرد :

```
#include "mythread.h"
```

حال برای اجرای تابع لازم است آن را کپی گرفته و اجرا نماییم :

```
MyThread iThread;  
iThread.start();
```

کد نهایی به شکل زیر خواهد بود:

```
#include <QCoreApplication>  
#include "mythread.h"
```

```
int main(int argc, char *argv[]){  
    QCoreApplication a(argc, argv);  
  
    MyThread iThread;  
    iThread.start();  
  
    return a.exec();  
}
```

ادامه معرفی و کار با Thread ها

در این بخش قصد داریم چند پروسه ساده ایجاد و آنها را برای پردازش در اختیار پردازنده قرار دهیم، برای این کار نیاز است متغیری از نوع رشته را در فایل mythread.h در کلاس MyThread تعریف کنیم :

```
QString Value;
```

در فایل cpp قسمت چاپ داخل بدن را به شکل زیر تغییر خواهیم داد :

```
qDebug() << this->Value() << "Thread is running !\n" ;
```

حال نتیجه حاصل از آن به صورت زیر خواهد بود:

T2Thread is running!

T1Thread is running!

T3Thread is running!

لازم است به این نکته اشاره شود که اطلاعات سنگین و هم‌زمان باید توسط Thread کنترل و مدیریت شوند، در غیر این صورت برنامه شما با مشکلاتی مانند کرش، هنگ کردن و یا پاسخ‌گویی با تاخیر مواجه خواهد شد.

اگر پروژه ای داشته باشیم که قرار است در هر ثانیه عملیاتی را انجام دهد مانند: چاپ زمان کنونی به صورت زیر ادامه خواهیم داد:

در سازنده‌ی فرم کد مربوط به صورت زیر خواهد بود:

```
//Start to get System time.  
QTimer *timer = new QTimer(this);  
timer->setInterval(1000);  
QThread * iTH = new QThread;  
timer->moveToThread(iTH);  
timer->start();  
connect(timer, SIGNAL(timeout()), SLOT(updateClock()));  
iTH->start();
```

در این مثال نمونه‌ای از QThread را به روش زیر ایجاد کرده‌ایم :

```
QThread * iTH = new QThread;
```

در خط بعدی توسط کد زیر Timer را به داخل Thread را به داخل Thread ارسال کرده‌ایم :

```
timer->moveToThread(iTH);
```

و در نهایت با برقراری ارتباط توسط مکانیزم سیگنال و اسلات بین تایمر و تابع مربوطه آن را نهایی می‌کنیم.

```
QObject::connect(timer, SIGNAL(timeout()), SLOT(updateClock()));  
iTH->start();
```

حال با اجرای این کد پروسه مورد نظر وارد نخ خواهد شد.

معرفی و کار با QMap

QMap چیست و چه کارهایی را انجام می‌دهد؟ یک نوع قالب با الگوی خاصی است که قابلیت تعریف لیستی متشکل از چندین رشته و کلیدهای اختصاصی را دارد. کلاس QMap یک کلاس از نوع قلب (Template) در سی‌پلاس‌پلاس است که یک شیوه‌ی ذخیره سازی به شکل دیکشنری را فراهم می‌کند. برای وارد کردن کلید و مقدار می‌توان از عملگر [] استفاده کرد.

```
map[ "one" ] = 1;
map[ "three" ] = 3;
map[ "seven" ] = 7;
```

و برای درج کلید و مقدار آن به صورت جفتی می‌تواند از عملگر () استفاده کرد به صورت زیر :

```
map.insert(1, "C++");
```

حال با توجه به این موارد لیستی را ایجاد می‌کنیم که هر کدام از آن‌ها دارای مقدار و کلید اختصاصی باشند.

ابتدا کلاس QMap را فراخوانی می‌کنیم :

```
#include <QMap>
```

کلاس QMap را فراخوانی کرده و یک الگو با شناسه‌ای از نوع int و مقادیر رشته‌ای ایجاد می‌کنیم، سپس توسط تابع insert کلید و رشته‌ی مورد نظر را در لیست وارد کرده و برای چاپ تمامی لیست توسط یک حلقه foreach، کلید و رشته‌های مربوط به هر کدام را چاپ خواهیم کرد.

```
QMap<int, QString> Programming_Languages;

Programming_Languages.insert(1, "C++");
Programming_Languages.insert(2, "Rust");
Programming_Languages.insert(3, "Java");
Programming_Languages.insert(4, "Php");
Programming_Languages.insert(5, "Python");

foreach(int i, Programming_Languages.keys())
```

```
{
    qDebug() << Programming_Languages[ i ];
}
```

خروجی کد فوق به صورت زیر خواهد بود:

```
"C++"
"Rust"
"Java"
"PHP"
"Python"
```

و اما در رابطه با نحوه فراخوانی کلیدها همراه با مقادیر آنها باید از کلاس **QMapIterator** استفاده شود، که در کل به سبک تکرار کننده‌های Java و STL فراهم می‌شود.

کد را به صورت زیر پیاده سازی می‌کنیم:

```
QMap<int, QString> Programming_Languages;  
  
Programming_Languages.insert(1, "C++");  
Programming_Languages.insert(2, "Rust");  
Programming_Languages.insert(3, "Java");  
Programming_Languages.insert(4, "Php");  
Programming_Languages.insert(5, "Python");  
  
QMapIterator<int, QString> ILTH(Programming_Languages);  
while (ILTH.hasNext())  
{  
    ILTH.next();  
    qDebug() << ILTH.value() << " Key is = " << ILTH.key();  
}
```

کلاس **QMapIterator** تکرار کننده را به سبک Java برای کلاس‌های **QMap** و **QMuliMap** فراهم می‌کند، بنابراین با پیاده‌سازی آن می‌توان مقدار و کلیدهای مربوطه را چاپ کرد که برای لیست کردن آنها نیز از حلقه‌ای مانند **while** استفاده کنید.

معرفی و کار با **QHash**

QHash چیست و چه تفاوتی با **QMap** دارد؟ این کلاس دقیقاً مانند کلاس **QMap** است با تفاوت‌های زیر نسبت به **QMap** قابل ارائه است:

- قابلیت جستجو و دسترسی بسیار سریع‌تر را نسبت به **QMap** دارد.
- در صورتی که لیست تعریف شده حاوی اطلاعات بسیار زیادی باشد در **QMap** لیست‌های رشته‌ای بر اساس کلیدهای تعریف شده مرتب می‌شوند ولی در **QHash** به صورت خودکار تمامی لیست‌ها بدون قاعده کلیدی مرتب می‌گردند.
- و تفاوت جزئی در نوع کلیدهای این دو کلاس که با **عملگرهای <** و **=** مشخص است.

برای استفاده به کلاس **QHash** نیاز خواهیم داشت به صورت زیر:

```
#include <QHash>
```

نحوه پیاده سازی و استفاده از آن دقیقاً مانند مثال قبلی در رابطه با QMap است در کل به صورت زیر باز نویسی می‌کنیم:

```
QHash<int, QString> Programming_Languages;  
  
Programming_Languages.insert(1, "C/C++");  
Programming_Languages.insert(2, "Ruby");  
Programming_Languages.insert(3, "Objective-C");  
Programming_Languages.insert(4, "Java");  
Programming_Languages.insert(5, "PHP");  
Programming_Languages.insert(6, "C#");  
Programming_Languages.insert(7, "VB.NET");  
  
foreach(int i, Programming_Languages.keys())  
{  
    qDebug() << Programming_Languages[i];  
}  
QHashIterator<int, QString> ILTH(Programming_Languages);  
while (ILTH.hasNext())  
{  
    ILTH.next();  
    qDebug() << ILTH.value() << " Key is = " << ILTH.key();  
}
```

QStringList و کار با

QList چیست و چه کاربردی دارد؟ کلاسی است از رشته‌ها که امکان ایجاد لیستی از رشته‌ها را فراهم می‌کند. برای مثال اگر بخواهیم متغیری تعریف کنیم که لیستی از زبان‌های برنامه‌نویسی را به ترتیب چاپ کند، این کلاس می‌تواند مورد استفاده قرار بگیرد.

برای فراخوانی به صورت زیر عمل می‌کنیم:

```
#include <QStringList>  
در کد زیر ابتدا نمونه‌ای از QStringList ساخته شده است و یک متغیر از نوع رشته که حاوی لیستی از نام زبان‌های برنامه‌نویسی است را مشخص کرده است:
```

```
QStringList MyList;  
QString MyLine =  
"C/C++,ObjectiveC,Java,Ruby,C#,PHP,ASP.Net,JavaScript";  
MyList = MyLine.split(",");  
  
foreach(QString Item, MyList)  
{
```

```
    qDebug() << Item;  
}
```

در خط بعدی متغیری که از نوع QStringList مشخص شده است را مقداردهی کرده و در حین مقدار دهی foreach محتویات رشته را جدا گانه در داخل myList قرار داده است، سپس توسط دستور split متغیری را به عنوان آیتم از نوع رشته در نظر گرفته و چاپ می‌کنیم و نتیجه به صورت زیر خواهد بود:

"C/C"++

"Objective-C"

"Java"

"Ruby"

"C" #

"PHP"

"ASP"

"JavaScript"

برای مثال می‌خواهیم مقدار حرف C را برابر با C++ قرار دهیم یعنی حرف C را با C++ جایگزین کنیم به راحتی توسط دستور زیر خواهیم داشت:

```
MyList.replaceInStrings("C", "C++");
```

کد کلی آن به شکل زیر است:

```
QStringList myList;  
QString MyLine = "A,B,C";  
myList = MyLine.split(",");
```

```
myList.replaceInStrings("C", "C++");
```

```
foreach(QString Item, myList)  
{  
    qDebug() << Item;  
}
```

در مواقعي نیاز است لیستی را مشخص کرده و آنها را چاپ کنیم، برای مثال لیستی از فونت‌ها را در نظر بگیرید که می‌توان به روش زیر آنها را چاپ کرد:

```
QStringList fonts;  
fonts << "Arial" << "Tahoma" << "Times" << "Courier";
```

```
for (int i = 0; i < fonts.size(); ++i)
    qDebug() << fonts.at(i).toLocal8Bit().constData();
```

ابتدا نوعی از QStringList تعریف کرده و سپس توسط عملگر `<<` رشته‌های مورد نظر را در fonts درج می‌کنیم، سپس توسط یک حلقه for مقادیر موجود را گرفته و توسط مجرای qDebug مقار ثابت آنها را چاپ کرده‌ایم.

شاید لازم باشد در طراحی از لیست فونت‌ها استفاده شود برای این کار شما نیاز خواهید داشت با تعریف یک تابع از نوع QStringList و بازگشتنی آن را پیاده سازی کنید، به این صورت که با مشخص سازی نوع تابع از نوع لیست رشته‌ای و ارسال مقدار به خروجی تابع می‌توان به لیست فونت‌های مورد نظر دسترسی داشت، پیاده سازی این تابع به صورت زیر است:

```
QStringList fonts();
```

در تابع می‌توانیم با تعریف نوع لیستی از رشته‌ها و استفاده از خاصیت append ارزشی را در انتخاب لیست اضافه نماییم که در اینجا سعی شده است از داخل منبع qrc که شامل یک سری فونت‌ها است اطلاعات را وارد و در نهایت نام فونت را در مواردی که نیاز است استفاده کنیم که کد کلی آن به صورت زیر است:

```
QStringList fonts()

{
    QStringList dataList;
    dataList.append("Yekan ");
    dataList.append("Tahoma");
    dataList.append("Calibri");

    return dataList;
}
```

برای دریافت مقدار بر اساس ایندکس که در اینجا با وارد کردن مقدار ۰ تا ۲ می‌توان به مقدار را گرفت.

```
qDebug() << fonts().at(0);
```

معرفی و کار با الگوریتم‌ها

همانطور که می‌دانید در C++ الگوریتم‌های متفاوتی وجود دارد که در این میان کتابخانه Qt نیز در کلاس‌های خودش الگوریتم‌های مشخصی را پیاده سازی کرده است که توسط کلاس **QtAlgorithms** فراخوانی می‌شوند؛ البته باید به این نکته اشاره کنیم که این اعضا منسوخ شده هستند و پیشنهاد می‌شود از توابع جدید استفاده شود.

Qt function	STL function
<i>qBinaryFind</i>	std::binary_search or std::lower_bound
<i>qCopy</i>	std::copy
<i>qCopyBackward</i>	std::copy_backward
<i>qEqual</i>	std::equal
<i>qFill</i>	std::fill
<i>qFind</i>	std::find
<i>qCount</i>	std::count
<i>qSort</i>	std::sort
<i>qStableSort</i>	std::stable_sort
<i>qLowerBound</i>	std::lower_bound
<i>qUpperBound</i>	std::upper_bound
<i>qLess</i>	std::less
<i>qGreater</i>	std::greater

qSort چیست و چه کاربردی دارد؟ معمولاً برای مرتب سازی مقادیر نیاز است از تابع std::sort

که در اینجا از تابع qSort است استفاده شود، بنابراین وظیفه‌ی این تابع مرتب سازی مقادیر موجود در لیست است.

یک مثال بسیار ساده از مرتب سازی اعداد را در نظر بگیریم، اگر اعداد زیر را داشته باشیم:

```
4  
8  
2  
1  
6  
3
```

به روش زیر لیستی از اعداد نا مرتب را وارد کرده و توسط تابع qSort آنها را مرتب می‌کنیم:

```
QList<int> myList;  
  
myList << 4 << 8 << 2 << 1 << 6 << 3;  
qSort(myList);  
  
foreach(int i, myList) {  
    qDebug() << i;  
}
```

در خط اول متغیر را از نوع لیست در نظر گرفته‌ایم و در خط بعدی آن مقادیر را وارد لیست کرده و در خط بعد توسط تابع qSort مقادیر را مرتب کرده‌ایم و بعد توسط qDebug و foreach آنها را لیست کرده و چاپ می‌کنیم.

```
1  
2  
3  
4  
6  
8
```

qFind چیست و چه کاربردی دارد؟

برای جستجو مقدار در لیست تابع qFind مورد استفاده قرار می‌گیرد. مثالی از الگوریتم جستجو کننده در کیوت توسط تابع qFind برگرفته شده از std::find است. در این مثال لیستی از اعداد را مشخص کرده و توسط تابع qFind از ابتدا تا انتهای لیست را جستجو می‌کند که آیا مقدار ۶۴ در لیست موجود است یا خیر، و بر اساس نتیجه پیغام مورد نظر را نمایش خواهد داد.

```
QList<int> myList;
myList << 12 << 0 << 133 << 64 << 512 << 128;
QList<int>::const_iterator MyIterator = qFind(myList.begin(), myList.end(), 64);

if (MyIterator != myList.end())
{
    qDebug() << "Found = " << *MyIterator;
}
else
{
    qDebug() << "Not found!";
}
```

qFill چیست و چه کاربردی دارد؟

به کمک تابع qFill می‌توان دامنه‌ی لیست موجود را مجدداً

مقدار دهی و جایگزین کرد (در واقع دامنه لیست را پر کرد).

```
QStringList list;
list << "one" << "two" << "three";
qFill(list.begin(), list.end(), "eleven");
```

qBinaryFind چیست و چه کاربردی دارد؟

در بازه‌ی آغاز و پایان لیست انجام داد که مقدار باز گشته آن موقعیت ارزش یافت شده در لیست است.

```
QVector<int> vect;
vect << 3 << 3 << 6 << 6 << 6 << 8;

QVector<int>::iterator i =
qBinaryFind(vect.begin(), vect.end(), 6);
```

qCopy چیست و چه کاربردی دارد؟

این تابع کمک می‌کند تا به راحتی آیتم‌های محدوده‌ای از

یک لیست را به محدوده‌ی یک لیست دیگر کپی کنیم، به عنوان مثال قرار است ۴ مقدار اولیه را که در لیست

اول موجود است در لیست دوم با یک محدوده مشخص کپی کنیم، قرار است بعد از ۲ آیتم آغاز کپی مقادیر

صورت بگیرد در واقع باید خروجی لیست نهایی به این شکل باشد:

```
 "", "", "one", "two", "three"
```

```
QStringList list;
list << "one" << "two" << "three";

QVector<QString> vect1(3);
qCopy(list.begin(), list.end(), vect1.begin());

QVector<QString> vect2(5);
qCopy(list.begin(), list.end(), vect2.begin() + 2);
```

در ابتدا لیستی از آیتم‌ها را مشخص کرده و در لیست دوم دقیقاً همان مقادیر را کپی خواهیم کرد، سپس در لیست سوم با مشخص سازی بازه ۵ آیتم توسط تابغ `qCopy` ابتدا و انتهای لیست اصلی را کپی کرده و به علاوه‌ی مقدار ۲ که مشخص می‌کند بعد از ۲ آیتم این کپی صورت بگیرد. در این صورت اگر از ابتدا لیست ما دارای ۵ آیتم بوده باشد ۲ آیتم ابتدایی آن جایگزین خواهد شد در غیر اینصورت دو آیتم شیفت شده و با مقدار خالی از آیتم سوم عمل کپی صورت خواهد گرفت.

چیست و چه کاربردی دارد؟ این تابع نیز همانند تابع `qCopy` عمل می‌کند، با این تفاوت که عمل شروع کپی را از انتهای لیست آغاز و به ابتدای آن ختم می‌شود.

```
QStringList list;
list << "one" << "two" << "three";

QVector<QString> vect(3);
qCopyBackward(list.begin(), list.end(), vect.end());
```

چیست و چه کاربردی دارد؟ این تابع عمل مقایسه از ابتدا تا انتهای آیتم‌های لیست اول و لیست بعدی را انجام می‌دهد که در صورت برابر بودن مقادیر هر دو لیست مقدار `true` و در غیر اینصورت مقدار `false` را بازگشت خواهد داد.

برای مثال در کد زیر مقدار لیست اول با دوم برابر است و نتیجه آن `true` خواهد بود :

```
QStringList list;
list << "one" << "two" << "three";

QVector<QString> vect(3);
vect[0] = "one";
vect[1] = "two";
vect[2] = "three";
```

```
bool ret1 = qEqual(list.begin(), list.end(), vect.begin());
```

در صورت گرفتن عمل چاپ مقدار `ret1` برابر `true` نمایش خواهد شد.
برای دریافت مقدار `false` کافی است ارزش `vect[2]` را در لیست دوم نا برابر با لیست سوم قرار دهیم به صورت زیر :

```
vect[2] = "seven";
bool ret2 = qEqual(list.begin(), list.end(), vect.begin());
```

در این حالت نتیجه بعد از چاپ `false` خواهد بود.

qCount چیست و چه کاربردی دارد؟ توسط این تابع می‌توان تعداد مقادیر تکراری را در بازه آغاز و پایان لیست نمایش داد که کد زیر نمونه‌ای از این عمل را مشخص می‌کند.

```
QList<int> list;
list << 1 << 2 << 3 << 4 << 4 << 4;

int countOf = 0;
qCount(list.begin(), list.end(), 4, countOf);
```

در این مثال لیست شامل اعداد ۱ تا ۴ است که در بین آنها عدد ۴ سه بار تکرار شده است بنابراین یک متغیر با مقدار صفر تعریف کرده و توسط تابع `qCount` ابتدا و انتهای لیست را خوانده و مقداری را که لازم است جستجو کنیم را مشخص می‌کنیم که در انتهای آن نیز `countOf` به عنوان مقدار یا ارزش آغاز کننده مشخص می‌شود، خروجی این مثال عدد ۳ خواهد بود که نشان دهنده سه بار تکرار شدن عدد ۴ است.

qStableSort چیست و چه کاربردی دارد؟ توسط این تابع می‌توان انواع آیتم‌های موجود در بازه [شروع، پایان) به ترتیب صعودی با استفاده از یک الگوریتم مرتب‌سازی پایدار را فراهم کرد.

```
QList<int> list;
list << 33 << 12 << 68 << 6 << 12;
qStableSort(list.begin(), list.end());
```

خروجی این کد به صورت 6, 12, 12, 33, 68 خواهد بود.

qLowerBound چیست و چه کاربردی دارد؟ این تابع یک جستجوی دودویی از محدوده [شروع، پایان) که موقعیت یک ارزش قبل از موقعیت مقدار مورد نظر مشخص می‌شود، یعنی اگر عدد ۵ را برای آن در نظر بگیریم و مقدار را در تابع ۷ قرار دهیم عدد ۵ هنگام مرتب شدن قبل از عدد ۷ نمایش داده خواهد شد (خروجی این کد به صورت 3, 3, 6, 6, 5, 7, 8 خواهد بود).

```

QList<int> list;
list << 3 << 3 << 6 << 6 << 8;

QList<int>::iterator i = qLowerBound(list.begin(), list.end(), 5);
list.insert(i, 5);

```

چیست و چه کاربردی دارد؟ این تابع نیز همانند تابع **qUpperBound**

می‌کند، با این تفاوت که در این حالت موقعیت مقدار مورد نظر بعد از ارزش آن مشخص می‌شود، یعنی اگر عدد ۵ را برای آن در نظر بگیریم و مقدار را در تابع ۷ قرار دهیم عدد ۵ هنگام مرتب شدن بعد از عدد ۷ نمایش داده خواهد شد.

```

QList<int> list;
list << 3 << 3 << 6 << 6 << 6 << 7 << 8;

QList<int>::iterator i = qUpperBound(list.begin(), list.end(), 7);
list.insert(i, 5);

```

خروجی این کد به صورت 3, 3, 6, 6, 6, 7, 5, 8 خواهد بود.

چیست و چه کاربردی دارد؟ وظیفه‌ی این تابع بازگشت دهنده آبجکت‌ها و عمل کننده‌هایی

است که می‌تواند در اختیار **qSort** و **qStableSort** قرار داد و طریقه استفاده آن به شکل زیر است:

```

QList<int> list;
list << 33 << 12 << 68 << 6 << 12;
qSort(list.begin(), list.end(), qLess<int>());

```

خروجی این کد به صورت 6, 12, 12, 33, 68 خواهد بود.

چیست و چه کاربردی دارد؟ این تابع نیز همانند تابع **qLess** است.

```

QList<int> list;
list << 33 << 12 << 68 << 6 << 12;
qSort(list.begin(), list.end(), qGreater<int>());

```

خروجی این کد به صورت 6, 12, 12, 33, 68 خواهد بود.

معرفی و کار با شبکه

در اینترنت همانند سایر شبکه‌های کامپیوتری از پروتکل‌های متعدد و با اهداف مختلف استفاده می‌شود، هر پروتکل از یک ساختار خاص برای ارسال و دریافت اطلاعات (بسته‌های اطلاعاتی) استفاده نموده و ترافیک مختص به خود را در شبکه ایجاد می‌کند.

در این میان HTTP (برگرفته از Hyper Text Transfer Protocol)، یکی از متدائل‌ترین پروتکل‌های لایه application است که مسئولیت ارتباط بین سرویس گیرنده‌گان و سرویس دهنده‌گان وب را برعهده دارد. در C++ کتابخانه‌های Qt کلاس‌هایی را برای شبکه دارند که برای فراخوانی کلاس‌ها به برنامه ابتدا لازم است در فایل .pro موجود در پروژه کد زیر را اضافه کنیم:

```
QT += network
```

حال پروژه‌ای را ایجاد کرده و تحت توضیحات قبلی مژول network را به پروژم اضافه می‌کنیم.

ابتدا یک کلاس برای پروتکل HTTP ایجاد کنیم، بنابراین قبل از هر چیز کلاس‌های مورد نیاز را به فایل کلاس اضافه می‌کنیم.

```
#include <QObject>
#include <QtGlobal>
#include <QObject>
#include <QNetworkAccessManager>
#include <QNetworkRequest>
#include <QNetworkReply>
#include <QFile>
#include <QTimer>
```

در این کلاس برای HTTP کلاس و توابع مورد نیاز را تحت کلاس‌های QNetworkAccessManager و QNetworkRequest ایجاد کرده‌ایم که وظیفه هرکدام به صورت زیر است:

- **کلاس QNetworkAccessManager** : وظیفه این کلاس این است که برای ما قابلیت ارسال اطلاعات و دریافت اطلاعات را از سرور فراهم کند.
- **کلاس QNetworkRequest** : وظیفه این کلاس دریافت و نگه‌داری اطلاعاتی است که کلاس QNetworkAccessManager به آنها نیاز خواهد داشت.
- **کلاس QNetworkReply** : وظیفه این کلاس نگه‌داری داده‌ها و هدرهایی است که توسط کاربر درخواست می‌شود.

• **کلاس QtGlobal**: گاهی لازم است این کلاس را فراخوانی کنیم زیرا هدرها و توابع اساسی Qt توسط اینکلود کردن این کلاس قابل شناسایی هستند که البته روش‌های دیگری هم وجود دارد تا به طور

کلی از هر بار اینکلود کردن چنین موارد در قسمتهای مختلف پروژه جلوگیری کنیم که بعدها توضیحاتی در رابطه با این موضوع خواهیم داد.

- **کلاس QObject**: برای ایجاد و شناسایی آبجکت‌ها استفاده می‌شود.
- **کلاس QFile**: برای کار با فایلها مورد نیاز است به عنوان مثال اگر قرار باشد فایلی را دانلود کنیم و عمل کپی کردن را روی فایل دانلود شده انجام دهیم به این کلاس نیاز اساسی خواهیم داشت.
- **کلاس QTimer**: برای کار با زمان مورد استفاده قرار می‌گیرد که در این مثال از آن برای مشخص سازی زمان دریافت و ارسال استفاده خواهیم کرد.

```
class MyHttp : public QObject
{
    Q_OBJECT

public:
    explicit MyHttp(QObject *parent = 0);
    virtual ~MyHttp();

signals:
    void addLine(QString qsLine);
    void downloadComplete();
    void progress(int nPercentage);

public slots:
    void download(QUrl url);
    void pause();
    void resume();

private slots:
    void download();
    void finishedHead();
    void finished();
    void downloadProgress(qint64 bytesReceived, qint64
bytesTotal);
    void error(QNetworkReply::NetworkError code);
    void timeout();

private:
    QUrl _URL;
    QString _qsFileName;
    QNetworkAccessManager* _pManager;
    QNetworkRequest _CurrentRequest;
    QNetworkReply* _pCurrentReply;
    QFile* _pFile;
    int _nDownloadTotal;
```

```

        bool _bAcceptRanges;
        int _nDownloadSize;
        int _nDownloadSizeAtPause;
        QTimer _Timer;
    };

```

در این حالت ما کلاسی را از نوع آبجکت و دسترسی حالت عمومی تعریف کرده‌ایم.

```
class MyHttp : public QObject
```

برای اینکه تمامی هدر فایل‌های آبجکت ما توسط زبان C++ قابل شناسایی باشند لازم است از ماکروی مخصوص آبجکت در Qt استفاده کنیم که به صورت زیر در کلاس ازش استفاده خواهد شد.

```
Q_OBJECT
```

در خط بعدی تعریف کلی کلاس مشخص شده است و بعد سیگنال و اسلات‌ها و متغیرهای مورد نیاز اعمال شده است. تا اینجا کار با فایل MyHttp.h تمام شده است و در ادامه برای تکمیل بدنه‌ی کلاس و توابع با فایل MyHttp.cpp کار خواهیم کرد.

فایل cpp ما حاوی کدهای زیر خواهد بود که در این فایل بدنه‌ی توابع را کامل کرده‌ایم و برای دانلود فایل همراه با اطلاعات دقیق ارسال و دریافت لازم داریم. نکته: شاید ساده‌تر از این می‌توان کد دانلود فایل را توسط http نوشت ولی نیاز می‌دانیم کمی پروژه پیچیده‌تر را انتخاب کنیم تا نتیجه حاصل از فایل دانلود شده همراه با پکیج‌های ارسالی و دریافتی همانند سیستم ارسال و دریافت کننده TeamViewer مشخص شده است.

```

#include "myhttp.h"
#include <QFileInfo>
#include <QDateTime>
#include <QDebug>

MyHttp:: MyHttp (QObject *parent) :
QObject(parent)
, _pManager(NULL)
, _pCurrentReply(NULL)
, _pFile(NULL)
, _nDownloadTotal(0)
, _bAcceptRanges(false)
, _nDownloadSize(0)
, _nDownloadSizeAtPause(0)
{
}

```

ابتدا فایل h. را باید وارد نماییم، زیرا که دسترسی به کلاس و توابع از پیش تعریف شده در فایل h. امکانپذیر نخواهد بود. QFileInfo برای کار با اطلاعات فایل است و گزینه‌ی QDateTime هم برای کار با تاریخ و زمان است در نهایت qDebug برای کار با qDebug مورد استفاده قرار می‌گیرد.

ابتدا کلاس را همراه با آبجکتها و هر یک از کلاس‌های مشتق شده آن که به نوعی کپی از کلاس‌ها برای استفاده از آن گرفته شده‌اند در اینجا آمده است فراخوانی کامل می‌کنیم که در ابتدا برای هر یک مقادیر پیشفرض بر اساس نوع آنها مشخص شده است که در طول اجرای برنامه متغیر خواهند بود.

```
MyHttp::~ MyHttp ()
{
    if (_pCurrentReply != NULL)
    {
        pause();
    }
}
```

در بدنه‌ی کلاس ویرانگر دستور شرطی آمده است که اگر پاسخی ارسال شده از طرف سرور به صورت NULL نباشد تابع pause اجرا خواهد شد.

```
void MyHttp::download(QUrl url)
{
    qDebug() << "download: URL=" << url.toString();

    URL = url;
    QFileinfo fileInfo(url.toString());
    qsFileName = fileInfo.fileName();
}

nDownloadSize = 0;
nDownloadSizeAtPause = 0;

_pManager = new QNetworkAccessManager(this);
_CurrentRequest = QNetworkRequest(url);

_pCurrentReply = _pManager->head(_CurrentRequest);

_Timer.setInterval(5000);
_Timer.setSingleShot(true);
connect(&_Timer, SIGNAL(timeout()), this, SLOT(timeout()));
_Timer.start();
```

```

    connect(_pCurrentReply, SIGNAL(finished()), this,
SLOT(finishedHead()));
    connect(_pCurrentReply,
SIGNAL(error(QNetworkReply::NetworkError)), this,
SLOT(error(QNetworkReply::NetworkError)));
}

```

تابع download را به صورت بالا می‌نویسیم که حاوی پارامتر url از نوع QUrl است که در حین اجرا ابتدا آدرسی که مختص فایل قابل دانلود است و حاوی یک کد چاپ برای آدرسی که ارسال کردیم است.

```
qDebug() << "download: URL=" << url.toString();
```

در این بخش آدرس ارسال شده دریافت و مشخصات آدرس توسط کلاس QFileinfo بررسی و نام فایل از آدرس گرفته شده و به متغیر از قبل تعریف شده یعنی _qsFileName ارسال می‌شود.

```

_URL = url;
{
    QFileinfo fileInfo(url.toString());
    _qsFileName = fileInfo.fileName();
}

```

در خط بعدی مقدار _nDownloadSizeAtPause را به متغیرهای _nDownloadSize و همچنین اختصاص می‌دهیم:

```
_nDownloadSize = 0;
_nDownloadSizeAtPause = 0;
```

حال کلاس اصلی که از آن استفاده خواهیم کرد به صورت زیر فراخوانی می‌شود:

```
_pManager = new QNetworkAccessManager(this);
_CurrentRequest = QNetworkRequest(url);
```

در این بخش مقدار کلاس مشتق شده _pManager را برابر کلاس اصلی QNetworkAccessManager می‌کنیم و همچنین کلاس‌های _CurrentRequest را با درخواست کننده آدرس یعنی QNetworkRequest با پارامتر url اختصاص می‌دهیم.

```
_pCurrentReply = _pManager->head(_CurrentRequest);
```

مقدار _pCurrentReply را برابر می‌کنیم با هدر (Header) ای که توسط کلاس _CurrentRequest گرفته شده است.

```
_Timer.setInterval(5000);
_Timer.setSingleShot(true);
```

نیاز است به مدت زمانی که باید توسط تایمر ایجاد کنیم برای مثال در اینجا میانگین زمانی را برابر ۵۰۰۰ میلی ثانیه قرار داده ایم و در خط بعدی آن مقدار QTimer یا همان تایمر شات که یکی از خاصیت‌های است را برابر با true قرار ساخته‌ایم.

```
connect(&_Timer, SIGNAL(timeout()), this, SLOT(timeout()));
_Timer.start();
```

با استفاده از تابع connect سیگنال ارسالی توسط تایمر را توسط تابع timeout ارسال و مجددا سیگنال بازگشته‌ی یا همان اسلات را توسط تابع timeout ارسال می‌کنیم.

```
connect(_pCurrentReply, SIGNAL(finished()), this,
SLOT(finishedHead()));
connect(_pCurrentReply,
SIGNAL(error(QNetworkReply::NetworkError)), this,
SLOT(error(QNetworkReply::NetworkError)));
```

دو کد نهایی عمل اتصال بین سیگنال و اسلات برای ارسال و دریافت جواب به عنوان اتمام کار انجام خواهند داد و در خط بعدی همین کار را برای زمانی که ارتباط با مشکل برخورده باشه ایجاد می‌کنیم.

نکته: ممکن است در موقعی نیاز باشد تا تابع connect را همراه با کلاس آن یعنی QObject فراخوانی کنید، از این روش زمانی استفاده می‌شود که connect به تنها ی قابل شناسایی نباشد.

تعریف تابع pause به صورت زیر خواهد بود :

```
void MyHttp::pause()
{
    qDebug() << "pause() = " << _nDownloadSize;
    if (_pCurrentReply == NULL)
    {
        return;
    }
    _Timer.stop();
    disconnect(&_Timer, SIGNAL(timeout()), this, SLOT(timeout()));
```

```

        disconnect(_pCurrentReply, SIGNAL(finished()), this,
SLOT(finished()));
        disconnect(_pCurrentReply, SIGNAL(downloadProgress(qint64,
qint64)), this, SLOT(downloadProgress(qint64, qint64)));
        disconnect(_pCurrentReply,
SIGNAL(error(QNetworkReply::NetworkError)), this,
SLOT(error(QNetworkReply::NetworkError)));

        _pCurrentReply->abort();
//        _pFile->write( _pCurrentReply->readAll());
        _pFile->flush();
        _pCurrentReply = 0;
        _nDownloadSizeAtPause = _nDownloadSize;
        _nDownloadSize = 0;
}

```

موارد ابتدایی مشخص است در خطی که () موجود است عمل متوقف سازی عملیات است، که در اینجا توسط تابع disconnect تمامی ارتباطات موجود بین سیگنال‌ها و اسلات‌ها از بین خواهد رفت یعنی در این صورت اگر فایلی در حال دانلود باشد یا مثلاً وظیفه دانلود به اتمام رسیده باشه توسط این تابع عمل توقف یا اتمام صورت می‌گیرد.

در این تابع در کل با استفاده از تابع disconnect تمامی ارتباطها از بین رفته و بعد از آن دستور از بین بردن تمامی درخواست‌ها توسط تابع abort ارسال می‌شود و در خط بعد یک گزینه‌ای به نام Flush به صورت زیر ازش استفاده شده:

```
_pFile->flush();
```

اگر در طول دریافت فایل اطلاعات به درستی و کامل بدون هیچگونه خرابی روی فایل به Buffer انتقال پیدا کنند در این حالت Flush مقدار true را ارسال می‌کند و در غیر اینصورت مقدار false را ارسال خواهد کرد در واقع توسط این می‌توانیم به راحتی راحت مشخص کنیم که فایل با موفقیت و بدون هیچگونه خرابی دانلود شده است یا خیر!

و اما تابع resume در باره این تابع هم اینطور توضیح می‌توان داد که وظیفه‌اش ادامه عملیات دانلود است در واقع با اجرای این تابع مسلماً باید تابع دیگری به نام download عملیات خودش را انجام بدهد.

```
void MyHttp::resume()
{
    qDebug() << "resume() = " << _nDownloadSizeAtPause;
    download();
}
```

در نهایت تابع فراخوانی شده و در بدنه تابع توسط qDebug متنی را هنگام درخواست ادامه با عنوان مقدار سایز از متغیری که در تابع pause مقدار دهنده شده بود دریافت می‌کند و در نهایت ادامه کار توسط اجرا شدن تابع download صورت خواهد گرفت.

نیاز است مجدداً تابعی برای download بنویسیم ولی با این تفاوت که دیگر اینجا آدرس فایلی برای دانلود download خواست این تابع زمانی اجرا می‌شود که در پست قبلی به آن اشاره شد پس نباید از تابع download ای که شامل پارامتر URL بود استفاده می‌کردیم. و اما تابع finishedHead که وظیفه این تابع دریافت اطلاعات از هدر و ارسال آن برای کاربر است.

```
void MyHttp::finishedHead()
{
    _Timer.stop();
    _bAcceptRanges = false;

    QList<QByteArray> list = _pCurrentReply->rawHeaderList();
    foreach(QByteArray header, list)
    {
        QString qsLine = QString(header) + " = " + _pCurrentReply-
>rawHeader(header);
        addLine(qsLine);
    }

    if (_pCurrentReply->hasRawHeader("Accept-Ranges"))
    {
        QString qstrAcceptRanges = _pCurrentReply-
>rawHeader("Accept-Ranges");
        _bAcceptRanges = (qstrAcceptRanges.compare("bytes",
Qt::CaseInsensitive) == 0);
        qDebug() << "Accept-Ranges = " << qstrAcceptRanges <<
_bAcceptRanges;
    }

    _nDownloadTotal = _pCurrentReply-
>header(QNetworkRequest::ContentLengthHeader).toInt();

    //      _CurrentRequest = QNetworkRequest(url);
    _CurrentRequest.setRawHeader("Connection", "Keep-Alive");

    _CurrentRequest.setAttribute(QNetworkRequest::HttpPipeliningAllowedA
ttribute, true);
    _pFile = new QFile(_qsFileName + ".part");
}
```

```

if (!_bAcceptRanges)
{
    _pFile->remove();
}
_pFile->open(QIODevice::ReadWrite | QIODevice::Append);

_nDownloadSizeAtPause = _pFile->size();
download();
}

```

و در ادامه تابع اتمام دانلود به صورت زیر خواهد بود :

```

void MyHttp::finished()
{
    _Timer.stop();
    _pFile->close();
    QFile::remove(_qsFileName);
    _pFile->rename(_qsFileName + ".part", _qsFileName);
    _pFile = NULL;
    _pCurrentReply = 0;
    emit downloadComplete();
}

```

تابع downloadProgress وظیفه این تابع مشخص کردن وضعیت و اطلاعات کلی در رابطه با مقدار اندازه دریافت شده و یا مقدار اندازه باقی مانده همراه با زمان آن را مشخص و نمایش خواهد داد.

```

void MyHttp::downloadProgress(qint64 bytesReceived, qint64
bytesTotal)
{
    _Timer.stop();
    _nDownloadSize = _nDownloadSizeAtPause + bytesReceived;
    qDebug() << "Download Progress: Received=" << _nDownloadSize <<
": Total=" << _nDownloadSizeAtPause + bytesTotal;

    _pFile->write(_pCurrentReply->readAll());
    int nPercentage =
static_cast<int>((static_cast<float>(_nDownloadSizeAtPause +
bytesReceived) * 100.0) / static_cast<float>(_nDownloadSizeAtPause +
bytesTotal));
    qDebug() << nPercentage;
    emit progress(nPercentage);

    _Timer.start(5000);
}

```

و باز هم یک تابعی برای مشخص سازی مشکلات ممکن به صورت زیر پیاده سازی می شود:

```

void MyHttp::error(QNetworkReply::NetworkError code)
{
    qDebug() << "(" << code << ")";
}

```

و تابع آخر نیز برای مشکلات ناشی در timeout

```

void MyHttp::timeout()
{
    qDebug() << "Time out error message!";
}

```

معرفی و کار با باینری و سریالیز کردن آبجکت‌ها

در رابطه با بحث **Serialization** می‌توان گفت که به طور کلی در کامپیوتر و سیستم‌های ذخیره‌سازی محتویات فایل‌ها به صورت‌های متنی و باینتری ذخیره می‌شوند، حال در حالت عادی یا همان پیشفرض اگر شما طبق مثال زیر رشته‌ای را در داخل یک فایل بنویسید کاملاً به صورت عادی و متنی در فایل مورد نظر ذخیره خواهد شد؛ اما در حالت باینری تمامی اشیاء به صورت یک حالت منحصر بفرد و خاصی ذخیره می‌شوند یعنی قبل از ذخیره مستقیم محتویات در فایل اشیاء مورد نظر به صورت Byte stream تبدیل شده و سپس در فایل ذخیره می‌شود. لازم است به کد زیر توجه کنید:

```

// writing on a text file
#include <iostream>
#include <fstream>
using namespace std;

int main () {
    ofstream myfile ("example.txt");
    if (myfile.is_open())
    {
        myfile << "This is a line.\n";
        myfile << "This is another line.\n";
        myfile.close();
    }
    else cout << "Unable to open file";
    return 0;
}

```

در کل شاید شنیده باشد دو حالت **DeSerialization** و **Serialization** را در بحث برنامه‌نویسی و مبحث ذخیره‌سازی داده‌ها بیان می‌شوند که به عمل تبدیل آبجکت به صورت باینری همان **Serialization** و به عکس آن عمل **DeSerialization** می‌گویند. حالا در این مثال روشی را برای خواندن و نوشتן اطلاعات در

داخل یک فایل را به صورت QDataStream به کمک کلاس DeSerialization و Serialization توضیح می‌دهیم.

برای مثال در نظر داریم متنی را به صورت زیر وارد نماییم :

C++11

C++14

C++17

و در نهایت نتیجه ذخیره شده در فایل به صورت باینری باشد که برای این کار کدی به صورت زیر خواهیم نوشته:

```
#include <QCoreApplication>
#include <QFile>
#include <QString>
#include <QDebug>
#include <QMap>

void SaveObject()
{
    int iNum = 64;
    QMap<int,QString> iMap;
    iMap.insert(1,"C++11");
    iMap.insert(2,"C++14");
    iMap.insert(3,"C++17");

    //Save to file.
    QFile iFile("c:/Test/itest.txt");

    if(!iFile.open(QIODevice::WriteOnly))
    {
        qDebug() << "Could not open file !";
        return;
    }

    QDataStream iOut (&iFile);
    iOut.setVersion(QDataStream::Qt_5_2);

    iOut << iNum << iMap;

    iFile.flush();
    iFile.close();
}
```

```

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

    SaveObject();

    return a.exec();
}

```

ابتدا هدرهای QMap و QFile را فراخوانی می‌کنیم چون هدف نوشتن در داخل یک فایل است از Model استفاده کرده و بعد یک تابع تعریف کرده‌ایم که ابتدا به صورت سلیقه‌ای یک نوع صحیح در نظر گرفته‌ایم برای اینکه همراه با متن مورد نظر ترکیب شود دنبال آن یک نوع رشته‌ای از QMap لیست ساخته و یک سری اطلاعات را به صورت لیست ایجاد کرده‌ایم.

توسط دستور زیر یک کپی از کلاس QFile گرفته‌ایم و مسیر مورد نظر را برای ذخیره سازی محتوا در فایل مشخص کرده‌ایم. در دستور بعدی یک شرط تعریف شده است که اگر فایل قابل باز شدن نباشد پیغامی را بدهد در غیر اینصورت ادامه مراحل صورت بگیرد. البته دقیق کنید که چون قصد ما نوشتن در فایل است از اصلی ترین قسمت QDataStream است که وظیفه‌ی این کلاس فراهم کردن امکان انتقال داده‌های باینری را به مجرای QIODevice است که اینجا ما یک کپی از آن ساخته و مشخص کرده‌ایم که فایل را در نظر بگیرد و بر اساس نسخه تعیین شده که اینجا از ۵.۱۲.۱ Qt خود استفاده شده اختصاص داده‌ایم که اصولاً از iMap نسخه‌های اولیه Qt_1_0 تا Qt_5_12 موجود است. در خط بعدی iNum را از نوع عدد صحیح همراه با iOut مخلوط و در Out وارد می‌کند که به ترتیب اولویت از راست به چپ وارد خواهد شد. خط بعدی Flush در نظر گرفته شده است که در توضیحات قبلی در رابطه با دلیل استفاده آن توضیح داده‌ایم. در نهایت فایل را بعد از نوشتن به حالت بسته شدن تغییر می‌دهیم. نتیجه باید در مسیر ذکر شده به صورت باینری ذخیره شده باشد.

DeSerialize و خواندن اطلاعات از داخل محتوای باینری شده برای این کار تابعی به صورت زیر ایجاد می‌کنیم:

```

void ReadObject()
{
    int iNum;
    QMap<int,QString> iMap;

    //Read from file.
    QFile iFile("c:/Test/itest.txt");

```

```

if(!iFile.open(QIODevice::ReadOnly))
{
    qDebug() << "Could not open file !";
    return;
}

QDataStream iIn (&iFile);
iIn.setVersion(QDataStream::Qt_5_2);

iIn >> iNum >> iMap;
iFile.close();

qDebug() << "iNum = " << iNum;

foreach(QString Item , iMap.values())
{
    qDebug() << Item;
}

}

```

ابتدا مقدار صحیح را خالی گذاشته‌ایم چون اینجا هدف خواندن اطلاعات است پس مقدار از طرف فایل گرفته و برگشت داده می‌شود به همین متغیر از نوع صحیح (البته منظور از مقدار همان مقدار ۶۴ است که به صورت باینری ذخیره شده است) مراحل بعد توسط QIODevice::ReadOnly اطلاعات را به صورت رشته‌ای در کنسول نمایش داده خواهد داد، ابتدا مقدار بازگشته عدد را در متغیر iNum ریخته و چاپش می‌کند و برای محتویات رشته‌ای هم به صورت زیرخواهد بود.

```

foreach(QString Item , iMap.values())
{
    qDebug() << Item;
}

```

در این بخش یک Item از نوع رشته مشخص کرده و بعد مقادیر iMap را که بازگشت داده شده است دریافت و در نهایت تمامی آنها را چاپ می‌کند.

معرفی و کار با TextStream‌ها

در رابطه با خواندن و نوشتمن در رشته‌ها کلاسی به نام **QTextStream** که امکان نوشتمن و خواندن را برای ما فراهم می‌کند وجود دارد؛ روش کار آن مشابه **QDataStream** که قبلًا در مورد آن توضیح داده شده است می‌باشد. با این تفاوت که در اینجا با رشته‌ها کار خواهیم کرد.

پروژه‌ای ایجاد می‌کنیم از نوع کنسول و ابتدا تابعی برای خواندن با نام **Read** و تابعی برای نوشتمن با نام **Write** تعریف کرده و به صورت زیر کلاس‌های مورد نیاز را فراخوانی خواهیم کرد.

```
#include <QTextStream>
#include <QFile>

void Read()

{
    QFile MyFile("d://MyFile.txt");
    if (MyFile.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        QTextStream MyStream(&MyFile);
        QString MyLine;

        do
        {

            MyLine = MyStream.readLine();
            qDebug() << MyLine;

        }
        while (!MyLine.isNull());
    }
    MyFile.close();
    qDebug() << "MyFile Read.";
}

}


```

و تابع نوشتمن به صورت زیر خواهد بود :

```
void Write()
{
    QFile MyFile("d://MyFile.txt");
    if (MyFile.open(QIODevice::WriteOnly | QIODevice::Text))
    {
        QTextStream MyStream(&MyFile);
        MyStream << "Hello \r\n";
        MyStream << "World \r\n";
        MyStream.flush();
        MyFile.close();
        qDebug() << "MyFile Written.";
    }
}
```

}

و در نهایت به صورت پشت سرهم هر دو تابع را به ترتیب اول تابع Write و بعد Read را فراخوانی می‌کنیم.

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

    Write(); //First step
    Read(); //Second step

    return a.exec();
}
```

توضیحات این بخش دقیقاً همانند `QDataStream` است.

معرفی انواع حالت‌های کامپایل در Qt

در رابطه با حالت داینامیک (Dynamic) توضیح مختصر:

در این حالت شما در بسیاری از موارد برای توسعه نرم‌افزار در دو حالت OpenSource و تجاری (انحصاری) قادر خواهید بود.

مزایا:

معمولًاً برنامه برای کاربر نهایی یا همان End user یک بسته جمع و جور و کامل را فراهم می‌کنید و همچنین فایل اجرائی در کم حجمترین و فشرده‌ترین حالت خارج خواهد شد.

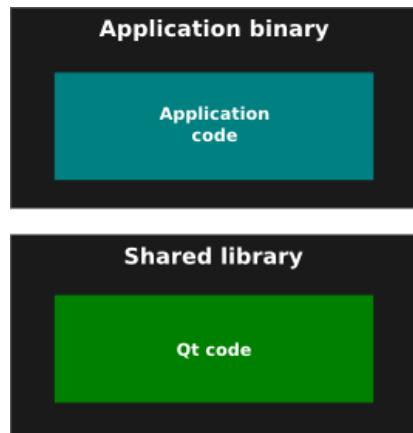
کتابخانه‌های Qt را شما می‌توانید بدون تغییر و کامپایل مجدد پروژه آنها را برای توسعه دهنگان بازخورد داده و یا به روز رسانی نمایید و حتی آنها را تغییر دهید.

ニاز به منابع سخت افزاری بسیار کمی است برای مثال اشغال حافظه Ram بسیار کمتر از حالت Static است.

معایب:

برای مطمئن شدن از کارائی درست برنامه در هر سیستم و هر سکویی نیاز است بارها و بارها از هر جوانبی برنامه را نسبت به کتابخانه بررسی نمایید تا وقتی مورد استفاده توسط کاربر یا همان User - End User قرار می‌گیرد بدون بروز مشکل یا خطای اجرا شود، معمولًاً بر روی سیستم‌های Linux نیز باید کتابخانه‌های به درستی نصب گردد.

شما باید مطمئن شوید که تمام کتابخانه‌های مورد نیاز در سیستم هدف (End User) در دسترس هستند در صورتی که بر روی سیستم مورد نظر در دسترس نیستند باید راه حل مناسبی برای ارائه کتابخانه‌های مورد نظر فراهم نمایید و خدمات آن را در اختیار کاربر قرار دهید.



در رابطه با حالت استاتیک (Static) توضیح مختصر :

در این حالت شما می‌توانید اطمینان باشید که برنامه شما در هر سیستمی بدون نیاز به پیش نیازی قابل اجرا خواهد بود.

جوانب مثبت و منفی زیر است

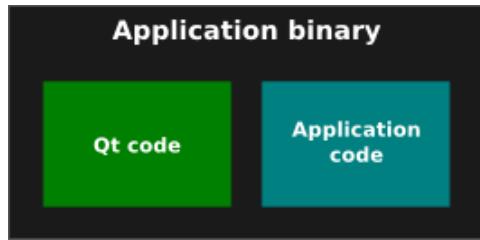
مزایا :

معمولًاً برنامه برای کاربر نهایی یا همان End user یک بسته جمع و جور و کامل را فراهم می‌کنید. برنامه شما می‌توانید مستقل از هر نسخه از کتابخانه‌های موجود بر روی سیستم کاربر برنامه را اجرا کنید. حالا چه Qt4 باشد و چه Qt5.5.1 باشه هیچ تداخلی نخواهد داشت.

معایب :

درخواست‌های برنامه شما به کتابخانه بسیار زیاد و سنگین خواهد بود زیرا کتابخانه‌ها نیز به برنامه شما متصل و لینک شده هستند.

ممکن است برای رفع مشکلات کتابخانه و تغییر / بهروز رسانی و ... مجبور به کامپایل مجدد برنامه شوید. مصرف منبع Ram در صورت درخواست‌های پی در پی و چند گانه بسیار زیاد خواهد بود. در حالت Runtime شما قادر به اجرای plugins QPluginLoader توسط توسعه نخواهید بود.



نحوه افزودن دیگر کتابخانه‌های C++ در محیط Qt Creator و استفاده

همراه با کتابخانه Qt

معمولًاً برنامه‌نویسان C++ مشتاق هستند از کتابخانه‌های سفارشی خودشان و یا کتابخانه‌هایی که مورد علاقه آن‌ها است استفاده کنند؛ این امر در محیط Qt Creator نیز همانند دیگر محیط‌های توسعه در C++ امکان‌پذیر است، برای مثال می‌توانیم کتابخانه‌ای مانند Boost, Poco و غیره را همراه با Qt مورد استفاده قرار دهیم.

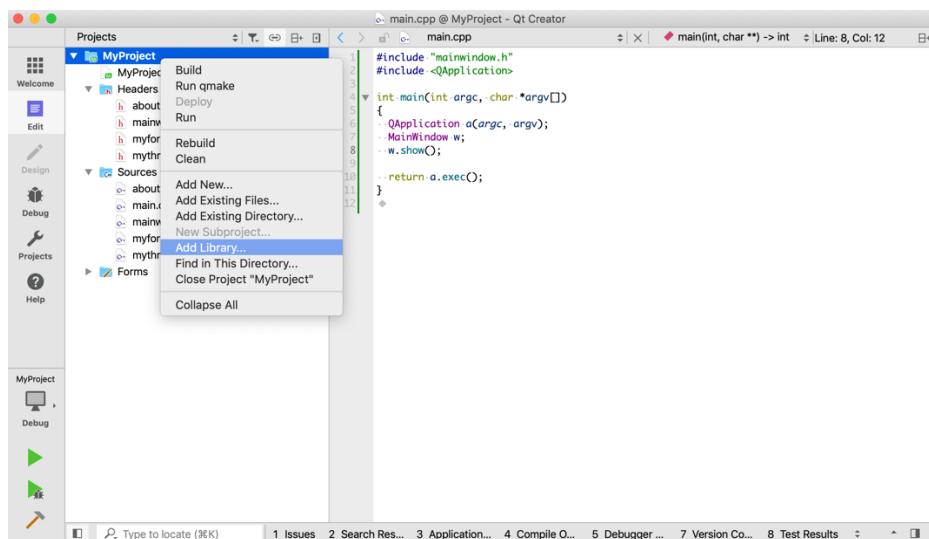
زبان C/C++ یکی از قابلیت‌هایی که نسبت به زبان‌های دیگری دارد نامحدود بودن استفاده از کتابخانه‌های این زبان است که به صورت پیشفرض کتابخانه‌های استاندارد و از قبل تعریف شده در زبان C++ قابل استفاده هستند. مانند کلاس‌های `iostream` و ... که کاملاً پیشفرض روی این زبان ارائه شده است.

هر کتابخانه‌ای باید قبل از کامپایل پیکربنی شده و سپس بیلد (ساخته) شود. برای مثال در کتابخانه‌ی poco دستورات زیر را بعد از دریافت اجرا خواهیم کرد.

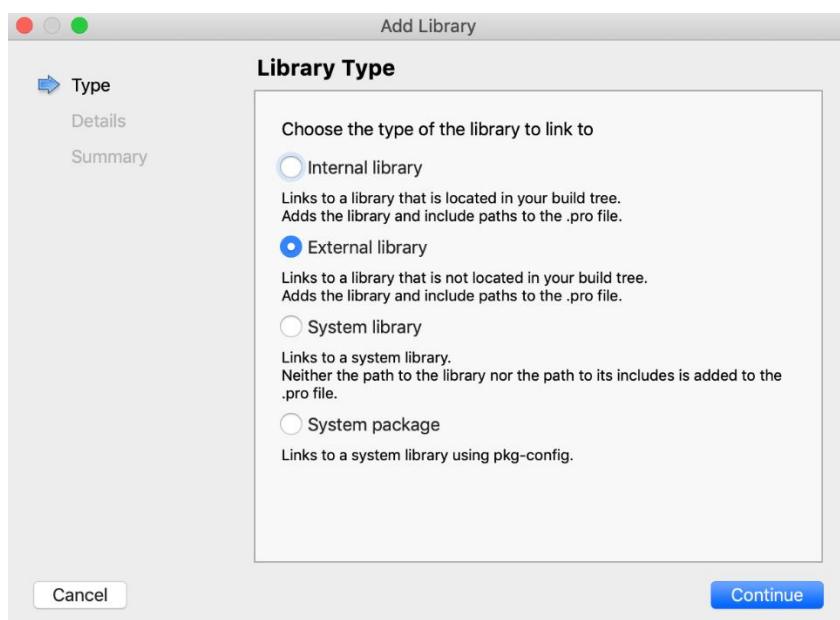
```
cd /Users/username/Documents/Libs/poco-1.9.0-all
```

سپس دستور `./configure` و بعد از آن دستور `make` را اجرا کنید تا کتابخانه شروع به کامپایل شدن کند. بعد از کامپایل در پوشه‌ی `lib` تمامی کتابخانه‌ها ایجاد خواهند شد.

وارد محیط Qt Creator شده و سپس بعد از ایجاد یک پروژه بر روی آن راست کلیک و گزینه‌ی `add library` را انتخاب کنید.



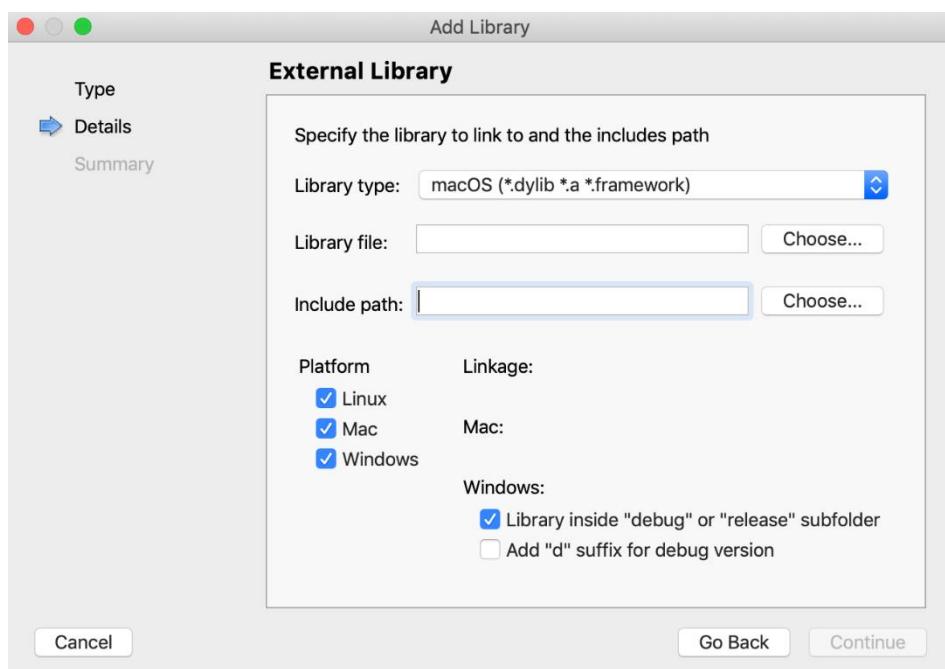
سپس مرحله بعد از آن به صورت تصویر بعدی نمایان خواهد شد:



۱. گزینه‌ی Internal Library مربوط است برای زمانی که شما کتابخانه را در داخل پروژه خودتان ایجاد کرده اید که معمولاً مسیر واقع توسط فایل pro مشخص خواهد شد به طور کلی کتابخانه‌های داخلی و غیر آن را می‌توانید از این قسمت شناسایی کنید.
۲. گزینه‌ی External library از این گزینه زمانی استفاده می‌شود که کتابخانه‌ی ما در مسیر پروژه نباشد یا به طور کلی در داخل پروژه قرار نگرفته باشد دقیقاً برعکس گزینه اول که همان کتابخانه خارجی یا خارج از پروژه است.
۳. گزینه System library هم مربوط می‌شود به کتابخانه‌های سیستمی.

۴. و یک گزینه ای هم خواهیم داشت در محیط‌های Unix که به نام Package Library قابل مشاهده خواهد بود این گزینه زمانی مورد استفاده قرار می‌گیرد که شما نیاز دارید کتابخانه را از طریق سرویس pkg-config در ایستگاه‌های Unix که شامل Linux و macOS است تنظیم کنید.

در حالت عادی از External Library استفاده خواهیم کرد بنابراین گزینه External Library را انتخاب و libPocoFoundation.60.dylib و Next را انتخاب خواهیم کرد که در مرحله‌ی بعد از آن فایل‌های آن را در بخش library file و سپس مسیر آن را در include path وارد کنید که مشابه مسیر زیر خواهد بود.



```
Library file : /Users/kambiz/Documents/Libs/poco-1.9.0-all/lib/Darwin/x86_64/libPocoFoundation.60.dylib
Include path : /Users/kambiz/Documents/Libs/poco-1.9.0-all/Foundation/include
Library file : /Users/kambiz/Documents/Libs/poco-1.9.0-all/lib/Darwin/x86_64/libPocoNet.60.dylib
Include path : /Users/kambiz/Documents/Libs/poco-1.9.0-all/Net/include
```

بعد از افزودن کسیر و کتابخانه گُد تولید شده در فایل pro. به صورت زیر خواهد بود:

```
win32:CONFIG(release, debug|release): LIBS += -L$$PWD/../../Libs/poco-1.9.0-all/lib/Darwin/x86_64/release/ -lPocoFoundation.60
else:win32:CONFIG(debug, debug|release): LIBS += -L$$PWD/../../Libs/poco-1.9.0-all/lib/Darwin/x86_64/debug/ -lPocoFoundation.60
else:unix: LIBS += -L$$PWD/../../Libs/poco-1.9.0-all/lib/Darwin/x86_64/ -lPocoFoundation.60

INCLUDEPATH += $$PWD/../../Libs/poco-1.9.0-all/Foundation/include
DEPENDPATH += $$PWD/../../Libs/poco-1.9.0-all/Foundation/include
```

```

win32:CONFIG(release, debug|release): LIBS += -L$$PWD/../../Libs/poco-1.9.0-
all/lib/Darwin/x86_64/release/ -lPocoNet.60
else:win32:CONFIG(debug, debug|release): LIBS += -L$$PWD/../../Libs/poco-1.9.0-
all/lib/Darwin/x86_64/debug/ -lPocoNet.60
else:unix: LIBS += -L$$PWD/../../Libs/poco-1.9.0-all/lib/Darwin/x86_64/ -
lPocoNet.60

INCLUDEPATH += $$PWD/../../Libs/poco-1.9.0-all/Net/include
DEPENDPATH += $$PWD/../../Libs/poco-1.9.0-all/Net/include

```

مراحل افزودن به پایان رسید، وارد محیط کیوت شده و داخل فایل main.cpp هدرهای مرتبط با کتابخانه و همچنین گُد آزمایشی آن را اضافه کنید.

```

//Qt Lib
#include <QCoreApplication>

//STL & External Lib

#include "Poco/Net/SocketAddress.h"
#include "Poco/Net/StreamSocket.h"
#include "Poco/Net/SocketStream.h"
#include "Poco/StreamCopier.h"
#include <iostream>

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

    Poco::Net::SocketAddress sa("www.iofstream.ir", 80);
    Poco::Net::StreamSocket socket(sa);
    Poco::Net::SocketStream str(socket);
    str << "GET / HTTP/1.1\r\n"
        "Host: www.iofstream.ir\r\n\r\n"
        "\r\n";
    str.flush();
    Poco::StreamCopier::copyStream(str, std::cout);

    return a.exec();
}

```

نحوهٔ خروجی گرفتن، گسترش (Deployment) در Qt

قبل از هر چیز لازم است بدانید که برای نصب و راه اندازی برنامه‌های نوشته شده تحت سی‌پلاس‌پلاس و کتابخانه‌های آن باید پیش‌نیازات آن‌ها در قالب فایل‌هایی از کتابخانه در کنار برنامه قرار بگیرد.

توسعه و اجرای برنامه بر روی بستر ویندوز:

در این محیط نسبت به نوع و نسخه‌ی Qt و مترجمی که مورد استفاده قرار گرفته است باید توجه داشته باشیم که هنگام مترجم و خروجی گرفتن متناسب با سیستم مقصد آن را تهیه کنیم، برای مثال نوع معماری یعنی $x64$ یا $x86$ بودن یک سیستم بسیار مهم است.

در ابتدا بعد از ایجاد پروژه این مهم است که مشخص کنیم نوع و نسخه مترجم و حتی حالت یا همان (Mode) مترجم بر چه اساسی تنظیم شده است، با توجه به این موارد فایل‌های مورد نیاز و نحوه اجرای برنامه متفاوت است.

مواردی که باید به آن‌ها هنگام مترجم توجه کنیم:

- مشخص سازی نوع کامپایل برنامه حالت یا همان Mode ای که برنامه روی آن ساخته می‌شود، اگر برنامه بر روی Debug ساخته می‌شود تمامی موارد بعدی بر اساس دیباگ تایین و در غیر اینصورت بر اساس نوع Release مشخص خواهند شد.
- نوع معماری خروجی در برنامه، باید توجه داشته باشید برنامه‌های 32 بیتی توسط مترجم‌های $x64$ یا 32 بیتی تهیه می‌شوند و برنامه‌های 64 بیتی توسط مترجم‌های $x86$ که خود نیازمند سیستم و بستر برنامه‌نویسی است که 64 بیتی هستند، یعنی اگر نیاز باشد برنامه شما 64 بیتی کامپایل شود ابتدا باید سیستم‌عامل و نسخه مترجم محیط توسعه از آن پشتیبانی کند.
- انواع مژول‌های استفاده شده در کتابخانه Qt مهم است، به عنوان مثال در حالت عادی مژول Qt5Core نیاز است ولی اگر در پروژه شما از مژول‌های دیگری مانند Network استفاده شده باشد در این حالت نیاز خواهید داشت فایل یا مژول مربوط به آن را وارد برنامه کنید که شامل Qt5Network است که لیست کاملی از مژول‌ها را بر اساس نیاز در ادامه مشخص خواهیم کرد که بر چه اساسی چه نوع مژول و چه فایلی باید همراه برنامه موجود باشد.

شروع کامپایل و گسترش برنامه:

معمولًاً نسخه‌های آزمایشی یک محصول در حالت Debug جهت بررسی و آنالیز خطاهای موجود در آن است که توسط تیم توسعه دهنده یا افرادی که می‌توانند در باگ گیری آن همیاری نمایند استفاده خواهند کرد،

بنابراین بر فرض اینکه ما قرار است یک نسخه استاندارد و نهایی از محصول را در اختیار کاربر قرار دهیم از حالت Release استفاده خواهیم کرد.

- معمولاً نسخه‌های آزمایشی یک محصول در حالت Debug جهت بررسی و آنالیز خطاهای موجود در آن می‌باشد که توسط تیم توسعه‌دهنده یا افرادی که می‌توانند در باگ گیری آن همیاری نمایند استفاده خواهد کرد، بنابراین بر فرض اینکه ما قرار است یک نسخه استاندارد و نهایی از محصول را در اختیار کاربر قرار دهیم از حالت Release استفاده خواهیم کرد.
- در بخش Projects می‌توان نوع مترجم و مسیر خروجی از آن را مشخص کرد، دقت کنید که در این بخش قسمت Build بر روی حالت Release باشد، در این مثال ما از مترجم MSVC2017 و نسخه ۶۴ بیتی آن استفاده کرده‌ایم که مسیر خروجی آن مشخص است.
- همانند مک و لینوکس در ویندوز نیز ابزاری با نام `windeployqt` وجود دارد که در مسیر `QTDIR/bin/windeployqt` می‌باشد. توسط این ابزار می‌توان برنامه را در قالب یک پکیج جمع‌آوری و مستقر ساخت.
- همانند مک و لینوکس در ویندوز نیز ابزاری با نام `windeployqt` وجود دارد که در مسیر `QTDIR/bin/windeployqt` می‌باشد. توسط این ابزار می‌توان برنامه را در قالب یک پکیج جمع‌آوری و مستقر ساخت.

*برای مثال ما برنامه‌ای ساخته‌ایم که در مسیر مورد نظر `MyAppRoot//C:/Users/Compez/Desktop` می‌باشد، با دستور `cd` به مسیر فوق خواهیم رفت:

```
cd C:/Qt/Qt5.11.0/5.11/msvc2017_64/MyAppRoot
```

البته قرار است در این مسیر خروجی فایل بعد از کامپایل ایجاد شود که با غیر فعال سازی امکان `Shadow` این ممکن خواهد شد که فایل مربوطه در مسیر ریشه برنامه ایجاد شود. با فرض اینکه بعد از کامپایل فایل `MyApplication.app` در مسیر ذکر شده موجود باشد دستور زیر را در ترمینال وارد خواهیم کرد:

```
C:/Qt/Qt5.11.0/5.11/msvc2017_64/bin/windeployqt MyApplication.exe
```

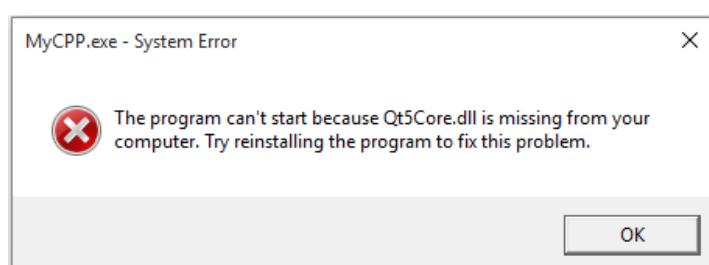
دقت کنید که اگر نیاز باشد با استفاده از گزینه‌های موجود در ابزار برنامه خود را مستقر سازید کافی است دستور ایجاد را به صورت زیر وارد کنید:

```
C:/Qt/Qt5.11.0/5.11/msvc2017_64/bin/windeployqt MyApplication.app -verbose=3 -no-plugins
```

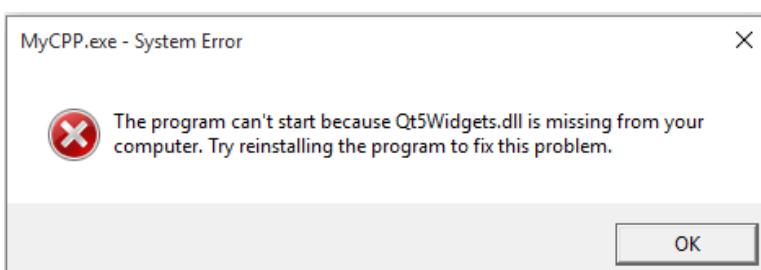
در ویندوز برخلاف ایستگاههای یونیکس فراهم آوردن تمامی فایل‌ها در کنار برنامه صورت خواهد گرفت. اما بعد از اجرای دستور فوق برنامه به تنهایی قابل اجرا نخواهد، لذا فایل‌های msvcp140.dll و vcruntime140.dll نیاز هستند تا در کنار برنامه قرار گیرند. این فایل‌ها در تمامی نرمافزارهای بزرگ در کنار برنامه موجود هستند مگر اینکه به صورت جدا پکیج مربوط به آن را نصب کنید که توصیه نمی‌شود.

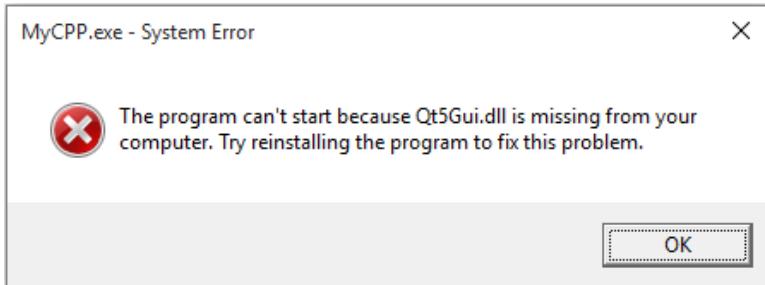
توجه داشته باشید که فایل‌هایی که قبل از پسوند .dll آخر حرف آن‌ها به d ختم می‌شود نشانگر آن است که مربوط به نسخه دیباگ هستند. در صورتی که در حالت Release برنامه خود را کامپایل می‌کنید فایل‌های QtCore.dll را در کنار برنامه خود قرار دهید که حرف آخر آن‌ها به d ختم نشده باشد. برای مثال فایل QtCore.dll مخصوص نسخه دیباگ بوده و فایل QtCore.dll مخصوص نسخه ریلیز.

*نکته: در نظر داشته باشید که همین روند برای کیوت نسخه MinGW نیز صدق می‌کند.
بعد از کامپایل برنامه و اجرای خروجی آن در ویندوزی که بر روی آن Qt و سی‌پلاس‌پلاس نصب نیست مسلماً با خطاهای زیر مواجه خواهیم شد:



و در نهایت خطای بعدی شامل خطای QtWidgets.dll خواهد بود که در ادامه خواهیم داشت:





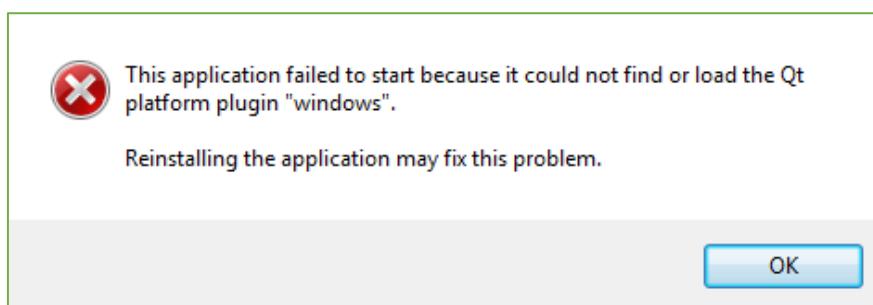
خطاهای فوق بیانگر این است که فایل‌های فوق در کنار پروژه یا در هسته سیستم عامل پوشیده نصب نشده است که در ادامه برای حل این خطا راهکار ارائه داده شده است. بنابراین به مسیر زیر بروید:

C:/Program Files (x86)/Microsoft Visual Studio
2017/Enterprise/VC/Redist/14.x/x/onecore/x64/Microsoft.VC150.CRT

سپس فایل‌های موجود در پوشیده را کپی و در کنار برنامه قرار دهید در این صورت برنامه بدون هیچ خطایی اجرا خواهد شد. مگر اینکه به جز کتابخانه‌های Qt و STL از کتابخانه‌های دیگری استفاده کرده باشد که در این صورت هم باید فایل‌های مربوط به آن‌ها را در کنار برنامه قرار دهید.

نکته : به طور پیشفرض مازول‌های Core, Gui و Widgets در پروژه موجود است، بنابراین وجود آن‌ها ضروری است.

توجه داشته باشد که بر اساس نوع برنامه‌ی شما، گزینه‌ای به نام افزونه‌ها (Plugins) مورد نیاز است چرا که برای شناسایی افزونه‌های استفاده شده به آن‌ها نیاز خواهد بود؛ لذا برای استفاده از امکانات بیشتر مانند کلاس تصاویر، دیتابیس و غیره می‌بایست این پوشیده با تمامی زیر مجموعه هایش در کنار برنامه‌ی شما قرار بگیرد که معمولاً برای نیاز به این پوشیده و محتویات آن پیغام زیر در صورت عدم وجود آن نمایش داده خواهد شد:

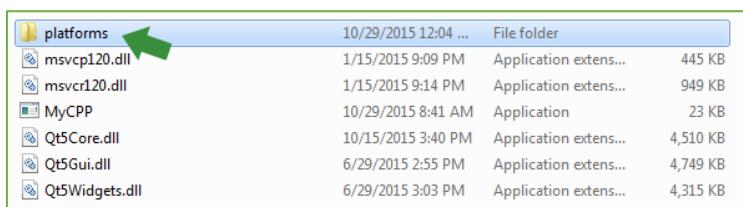


پوشه‌ی Plugins شامل زیر شاخه‌های زیر است:

audio, bearer, designer, generic, geoservices, iconengines, imageformats, mediaservice, platforms, playlistformats, position, printsupport, qml1tooling, qmltooling, qtwebengine, است که هرکدام از آنها دارای فایل‌های dll مربوط به خودشان sensorgestures, sensors, sqldrivers هستند.

که معمولاً پوشه‌ی platforms یکی از مهمترین افزونه‌های موجود در اجرای برنامه نوشته شده تحت این کتابخانه خواهد بود که مهمترین فایل موجود در آن نیز qwindows.dll است که در سیستم‌عامل ویندوز کاربرد دارد.

برای رفع این پیغام و خطأ باید پوشه‌ی platform موجود در مسیر فوق C:\Qt\Qt5.5.0-MSVC2013\plugins x86\5.5\msvc2013\plugins را در کنار فایل اجرایی کپی کنید. به صورت زیر:



حال می‌توانید بدون دریافت هیچ پیغامی ساده‌ترین برنامه‌ی نوشته شده در C++ را در کتابخانه Qt اجرا نمایید، این روش یکی از ساده‌ترین روش‌های است که در این بخش توضیح داده شد.

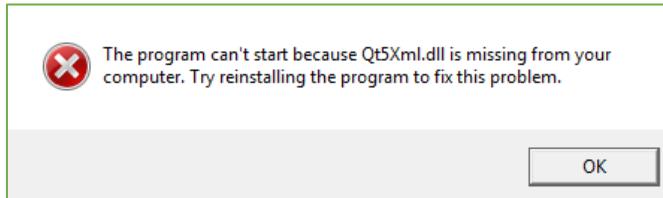
بنابراین سفارشی سازی پوشه‌یها و فایل‌های dll لازم در سطوح پیشرفته از آموزش توضیح داده خواهد شد، برای مثال می‌توان مشخص کرد که تمامی فایل‌های dll در یک پوشه‌ی ای سفارشی سازی شوند.

توجه داشته باشید در صورتی که به جز ماثول‌ها پیشفرض از ماثول‌ها و کلاس‌های انحصاری دیگری در Qt استفاده می‌کنید باید بر اساس نوع ماثول و پیش نیازات آن اقدام به فراهم کردن فایل‌های مربوطه کنید، برای این کار کافی است نام ماثول را در نظر بگیرید و فایل مرتبط با آن را در کنار برنامه خود قرار دهید.

یک مثال در رابطه با این گزینه، فرض کنیم می‌خواهیم از ماثول xml جهت استفاده از کلاس‌های استفاده کنیم که در این صورت باید در فایل pro. نام کلیدی ماثول را وارد کنیم که با روش معمول صورت می‌گیرد:

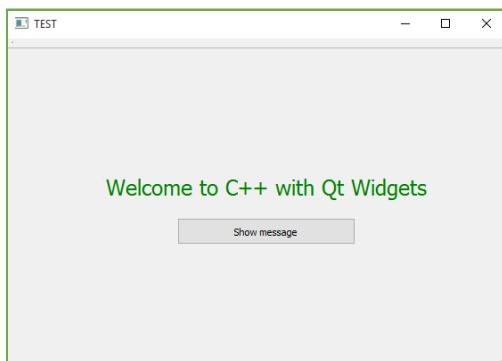
```
QT += xml
```

حال اگر برنامه مورد نظر را در سیستم کاربر اجرا کنید با پیغام زیر مواجه خواهیم شد:



کافی است وارد مسیر مربوطه C:\Qt\Qt5.12.1-MSVC2017-x86\5.12.1\msvc2017\bin شده و فایل مرتبط با این ماژول که در این مثال Qt5Xml.dll است را کپی و در کنار برنامه قرار دهید تا برنامه قادر به فراخوانی کتابخانه مربوط به XML باشد.

سپس با کلیک بر روی فایل اجرایی برنامه بدون هیچ پیغام یا خطایی به صورت زیر اجرا خواهد شد:



نکته : تمامی ماژول‌ها طبق همین قانون لازمه‌ی قرار گرفتن در کنار برنامه در صورت استفاده از آن‌ها را می‌باشند که در این میان برخی از ماژول‌ها کمی متفاوت خواهند بود که در بحث فناوری Qt Quick و QML این قضیه صدق می‌کند و توضیحات مربوط به آن‌ها در کتاب پیشرفت‌هه آمده‌است.

نکات مهم در رابطه با پیش نیازات در موارد خاص

اگر چه طبق شرایط توضیح داده شده برنامه به درستی و بدون هیچ خطای اجرا می‌شود، ولی در موارد خاص اگر شما از یک سری مازول‌ها و یا کتابخانه‌های دیگر C++ استفاده کنید بر اساس نیاز باید آن‌ها را شناسایی و در کنار پروژه فراهم نمایید.

مواردی مانند OpenGL، Direct X، Web engine‌ها و یا سیستم‌های یونیکد وجود دارد که برای پشتیبانی از این موارد باید با روش‌های انحصاری خود آن‌ها پروژه را پیکربندی نماییم.

برای مثال اگر در کدهای خود از سیستم یونیکد استفاده می‌کنید که شامل کتابخانه‌های ICU است، شما باید کتابخانه‌های مربوطه را در کنار برنامه فراهم سازید که هرگدام نسخه‌های متفاوتی را دارا است که در جدول زیر نسخه‌های موجود و همانهنج با Qt5 مشخص شده است که معمولاً آخرین نسخه‌های این کتابخانه در سایت رسمی آن موجود است.

نام فایل

icudtXX.dll

icuinXX.dll

icuucXX.dll

فایل‌های مورد نیاز در سرور دات‌ویوز برای عموم موجود است که طبق لینک‌های ذکر شده می‌توانید از آن‌ها استفاده کنید، معمولاً پروژه‌ها یی که نیازمند این فایل‌ها هستند هنگام اجرا پیغامی خواهند داد که حاوی نام و نسخه فایل مرتبط با کتابخانه ICU است.

نکته: در صورتی که از مترجم MinGW استفاده می‌کنید فایل‌های فوق مورد نیاز خواهند بود: libgcc_s_dw2-1.dll، libwinpthread-1.dll و libstdc++-6.dll کپی کنید.

مقایسه‌ی و پیکربندی موتورهای ANGLE و OpenGL در پروژه

همانطور که مشخص است این تو عنوان دو موتور رندرینگ گرافیکی هستند که در C++ بسیار پرکاربرد است، حال این دو موتور چه ویژگی‌هایی دارند و پیکربندی پروژه به چه روش خواهد بود.

موتور OpenGL : اوپن‌جی‌ال

سه‌بعدی است، این رابط برنامه‌نویسی معمولاً برای تعامل با پردازشگر گرافیکی و به‌دست‌آوردن رندرینگ شتاب‌یافته توسط سخت‌افزار استفاده می‌شود، اوپن‌جی‌ال مخفف Open Graphic Library (کتابخانه گرافیکی باز) است ولی اوپن‌جی‌ال خود به هیچ‌وجه یک کتابخانه نرم‌افزاری نیست و نرم‌افزار متن‌باز نیز به حساب نمی‌آید (چون حاوی هیچ‌کدی نیست)؛ اوپن‌جی‌ال، تنها استانداردی باز برای توصیف یک رابط گرافیکی است که توسط شرکت‌های متعددی توسعه داده شده و می‌تواند توسط درایورهای گرافیک، سیستم‌عامل‌ها و نرم‌افزارهای مختلف پیاده‌سازی شود.

Open Graphics Library for Embedded OpenGL ES 2.0 که برگرفته شده است همچنین نسخه Systems است، نسخه‌ای است برای استفاده در سیستم‌های جدا سازی شده (Embedded) که برخی از توابع جامع در آن موجود نیست.

در این میان فناوری **Qt Quick 2.0** در پشت کیوت ۵ بر پایه OpenGL است، که برای استفاده از آن framebuffer_object مورد نیاز است و یا حداقل نسخه OpenGL 2.X همراه با افزونه‌ی OpenGL 3.0 است و یا کمتر از آن با حداقل کارآیی به نسخه 1.3 OpenGL و بالاتر نیاز است.

موتور ANGLE : Almost Native Graphics Layer Engine

گرافیکی بر اساس لایه‌های بومی در سیستم‌عامل تعییه می‌شود که تقریباً بر پایه Direct X در ویندوز قابل استفاده است، معمولاً با استفاده از این گزینه نیازمند فایل‌های libEGL.dll ، d3dcompiler_XX.dll و d3dcompiler_47.dll در کنار برنامه خود خواهیم بود که در نسخه 5.5 کیوت شامل فایل libGLESv2.dll است و باید از مسیر فوق در کیوت کپی و در کنار برنامه اجرایی قرار بگیرد.

نام فایل و نسخه

libEGL.dll	libGLESv2.dll	d3dcompiler_XX.dll
------------	---------------	--------------------

نکته : در نسخه‌ی Qt 5.5 به بعد مشکل اینکه بر اساس نیاز باید از یکی از این دو نوع موتورها استفاده شود توسط سوئیچینگ حل شده است و به طور خودکار در این نسخه به بعد امکان شناسایی بهترین حالت در سیستم شناسایی شده و برنامه به صورت خودکار تشخیص خواهد داد که از چه نوع موتوری باید استفاده کند تا بهترین کارآیی با بهترین سرعت در اختیار کاربر گذاشته شود، بنابراین کافی است فایل‌های مربوطه را در کنار پروژه داشته باشید تا در صورت عدم سازگاری و منابع کافی در سیستم از این فایل‌ها جهت پشتیبانی از نوع ANGLE استفاده شود و در صورتی که سخت افزار و منابع کافی در اختیار برنامه قرار گیرد به صورت پیشفرض از موتور OpenGL استفاده خواهد شد.

نکاتی در رابطه با افزایش حجم بی رویه برنامه:

- معمولاً استفاده از تصاویر و سورس‌هایی با حجم بالا موجب افزایش حجم فایل اجرایی می‌شود که برای رفع این مشکل توصیه می‌شود از سورس‌های استاندارد استفاده شود.
- حجم برنامه‌های تولید شده در حالت Debug بیشتر از برنامه‌های Release شده است، و توصیه می‌شود به هیچ عنوان از این حالت جهت ارائه نسخه اصلی برنامه‌های نوشته شده برای کاربر تحت هیچ زبانی استفاده نشود، لذا نسخه‌ی Debug تنها مناسب توسعه‌دهنده جهت روشی رویی با خطاهای و رفع آن‌ها است.
- در صورت نیاز به فایل‌های صوتی یا تصویری آن‌ها را در پوششی ای خارج از برنامه قرار دهید.

درایور دیتابیس هایی که تحت این کتابخانه پشتیبانی می‌شوند

در این کتابخانه توسط افزونه SQLDrivers از API‌های مربوط به دیتابیس زیر پشتیبانی می‌شود.

نام درایور	توضیحات مربوطه
QDB2	پشتیبانی از نسخه‌های ۷.۱ به بالای IBM DB2
QIBASE	پشتیبانی از بورلند
QMYSQ	پشتیبانی از نسخه‌های MySQL و MariaDB
QOCI	پشتیبانی از Oracle
QODBC	پشتیبانی از درایور مایکروسافت (SQL Server)
QPSQL	پشتیبانی از نسخه‌های ۷.۳ به بالای PostgreSQL
QSQLITE2	پشتیبانی از SQLite نسخه ۲
QSQLITE	پشتیبانی از SQLite نسخه ۳
QTDS	پشتیبانی از Sybase Adaptive Server که از نسخه ۴.۷ کیوت به بعد منسوخ شده است.

ساخت راهاندازهای دیتابیس

استفاده از دیتابیس یکی از ویژگی‌های مهم کتابخانه کیوت به شمار می‌آید. با توجه به اینکه ماژول SQL از افزونه‌ها (افزونه‌ها) برای برقراری ارتباط بین API‌های مربوطه استفاده می‌کند، بعضی از آن‌ها مانند SQLite همراه با کیوت ارائه می‌شود. اما با توجه به عدم وجود درایورهای از پیش ساخته شده همراه کیوت نیاز است برخی از مهمترین آن‌ها مانند MySQL و PostgreSQL را به صورت سفارشی برای کیوت پیکربندی و ساخته شود که در این بخش به نحوه راهاندازی درایور دیتابیس MySQL می‌پردازیم.

نکته: نسخه SQLite افزونه مربوط به کیوت به صورت پیشفرض بر روی بستر شما قابل استفاده است که نیازی برای باز ساخت آن نیست.

ساخت راهانداز دیتابیس در پلتفرم‌های Windows، macOS و Linux

توجه داشته باشید که جهت ساخت درایور مربوطه شما به فایل‌های هدر مربوط به MySQL نیاز خواهد داشت. بنابراین فایل‌های به اشتراک گذاری `libmysqlclient.so` در لینوکس و `libmysqlclient.dylib` در مکینتاش جهت کارکرد صحیح برنامه مورد نیاز است که باید ساخته شوند. جهت نصب پکیج‌های مربوط به MySQL قدم اول به صورت زیر است:

در بستر مک‌اوی‌اس دستور زیر را توسط مدیریت بسته‌های ناموجود (brew) اجرا و پکیج مربوطه را نصب کنید:

```
brew install mysql
```

در صورتی که مدیریت بسته‌ی [brew](#) بر روی سیستم شما نصب نیست، دستور زیر را برای نصب آن اجرا کنید:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

در لینوکس دستور زیر جهت نصب پکیج مربوطه نیاز خواهد بود:

```
sudo apt-get install libmysqlclient-dev
```

به مسیر موجود در کیوت رفته و آن را در ترمینال تایید کنید:

```
cd $QTDIR/qtbase/src/plugins/sqldrivers
```

مثال

```
cd /Users/kambiz/Qt5.12.1/5.12.1/Src/qtbase/src/plugins/sqldrivers
```

در ادامه دستور زیر را جهت پیکربندی و شناسایی پکیج و درایورهای مورد نیاز را اجرا کنید:

```
qmake -- MYSQL_PREFIX=/usr/local
```

در صورتی که همه چیز به خوبی پیش رفته باشد، نتیجه‌ای به صورت زیر خواهیم داشت:

```
Running configuration tests...
Done running configuration tests.
```

```
Configure summary:
```

```
Qt Sql Drivers:
DB2 (IBM) ..... no
InterBase ..... no
MySQL ..... yes
OCI (Oracle) ..... no
ODBC ..... no
PostgreSQL ..... yes
SQLite2 ..... no
SQLite ..... yes
Using system provided SQLite ..... no
TDS (Sybase) ..... no
```

```
Qt is now configured for building. Just run 'make'.
Once everything is built, you must run 'make install'.
Qt will be installed into '/usr/local/Cellar/qt/5.12.1'.
```

```
Prior to reconfiguration, make sure you remove any leftovers from
the previous build.
```

لازم است به خروجی فوق دقت کنید، درایور QSqlite به دلیل وجود افزونه‌ی آن در کنار سورس کیوت به درستی شناسایی و قابل استفاده است. درایور PostgreSQL و همچنین MySQL به دلیل نصب پکیج‌های dylib مربوطه در بستر مربوطه قابل شناسایی و ساخت هستند. بنابراین برای ساخت فایل‌های مورد نیاز مک‌اواس و SO در لینوکس دستور زیر را اجرا کنید:

```
make sub-mysql
```

در صورتی که همه چیز به خوبی پیش رفته باشد، فایل libqsqlmysql.dylib در مک در مسیر زیر ساخته خواهد شد که نمونه‌ی SO. نیز در محیط لینوکس مشابه همان است:

Qt5.12.1/5.12.1/Src/qtbase/src/plugins/sqldrivers/plugins/sqldrivers/libqsqlmysql.dylib
فایل مربوطه را از مسیر فوق کپی کرده و در مسیر /usr/local/lib/ قرار دهید.

نحوه‌ی ساخت راهانداز MySQL بر روی پفترم‌های Windows

در محیط ویندوز با کمی تفاوت، نیاز است تا فایل نصبی MySQL را دریافت و آن را نصب کنید. در زمان نصب مسیر سفارشی آن را به صورت C:/MySQL انتخاب کرده و مطمئن شوید که پوشش‌های include و libs شامل محتوای مورد نیاز ھدرها و فایل‌های مربوطه می‌باشند.

توسط محیط کنسول Command Prompt به مسیر موجود در کیوت رفته و آن را در ترمینال تایید کنید:

```
cd $QTDIR/qtbase/src/plugins/sqldrivers
```

مثال

```
cd c:/Qt5.12.1/5.12.1/Src/qtbase/src/plugins/sqldrivers
```

دستور زیر عمل ساخت افزونه مربوطه را اعمال می‌کند:

```
qmake -- MYSQL_INCDIR=C:/MySQL/include "MYSQL_LIBDIR=C:/MySQL/MySQL Server 5.8/lib/opt"
```

در ادامه دستور زیر فایل مربوطه را خواهد ساخت:

```
nmake sub-mysql
```

توجه: در صورتی که دستور nmake برای شما کار نکند، بهتر است از دستور mingw32-make و یا make به عنوان جایگزین آن استفاده کنید، در این صورت توجه داشته باشید که باید توسط کنسول مخصوص MinGW اقدام کنید، در غیر این صورت امکان شناسایی و ساخت وجود نخواهد داشت. در ادامه فایل‌های ساخته شده را در مسیر مشخص شده کپی کرده و در مسیر شاخه‌ی اصلی c:/Windows/system32 قرار دهید.

قانون حمایت حقوق مولفان و مصنفان و هنرمندان

ماده ۱- از نظر این قانون به مولف و مصنف و هنرمند «پدید آورنده» و به آنچه از راه دانش یا هنر و یا ابتکار آنان پدید می‌اید بدون در نظر گرفتن طریقه یا روشی که در بیان و یا ظهور و یا ایجاد آن به کار رفته «اثر» اطلاق می‌شود.

ماده ۲ - حقوق پدید آورنده شامل حق انحصاری نشر و پخش و عرصه و اجرای اثر و حقوق بهره برداری مادی و معنوی از نام و اثر است.

بنابراین هرگونه کپی برداری از این کتاب و تکثیر آن بدون اطلاع نویسنده و شرکت توسعه دهنده پیگرد قانونی خواهد داشت، لذا خواهشمند است در صورت مشاهده و یا دریافت نسخه ای از این کتاب در منابع غیر مجاز از طرف شرکت ما آن را به ما اطلاع دهید.

<https://kambizasadzadeh.com>

<https://iostream.ir>

<https://genyleap.com>

Social Media | @KambizAsadzadeh