لایبرری

لایبرری مجموعه ای از کدهای از پیش کامپایل شده است که می تواند توسط برنامه ها دوباره استفاده شود و به دو دسته static و dynamic تقسیم می شوند

- **لایبرری static:** یک لایبرری استاتیک حاوی کدهایی است که در زمان کامپایل به برنامه های کاربران مرتبط می شود. فایل اجرایی تولید شده کپی خود را از کد لایبرری نگه می دارد.
- **لایبرری dynamic:** یک لایبرری داینامیک حاوی کدهایی است که برای چندین برنامه اشتراک گذاری شده است. محتوای لایبرری در زمان اجرا در حافظه بارگذاری می شود.

Static vs Dynamic Linking

- **Static Linking:** پیوند استاتیک یک فایل اجرایی مستقل تولید می کند بنابراین این به معنی استقرار فایل‌های کمتر ، اما یک فایل اجرایی بزرگ‌تر است. همچنین، بیلد برنامه بیشتر طول می کشد.مزیت اصلی پیوند استاتیک سرعت بیشتر برنامه در زمان اجرا است.نقطه ضعف پیوند استاتیک این است که افزودن یا به روز رسانی یک لایبرری نیاز به ریبیلد کامل و استقرار مجدد برنامه دارد.
- **Dynamic Linking:** پیوند داینامیک منجر به یک فایل اجرایی می شود که به موجود بودن لایبرری های پیوند داده شده بستگی دارد. بنابراین استقرار فایل‌های بیشتری را شامل می‌شود، اما خود برنامه کوچک‌تر خواهد بود. این برای برنامه هایی با استقرار مکرر عالی است.مزیت اصلی پیوند داینامیک این است که کامپایل زمان کمتری برای تکمیل می‌برد و فایل اجرایی کوچک‌تر است. همچنین کاربر می تواند کتابخانه Qt مورد استفاده برنامه را به طور مستقل به روز کند.

نتیجه: جایی که باید کامپایل‌های سریع زیادی انجام دهید، پیوند داینامیک انتخاب بهتری است.

زمانی از پیوند استاتیک استفاده می کنیم که:

- برنامه باید execution performance بالایی داشته باشد
- افزایش سایز برنامه مشکل ساز نباشد
- زمان بیلد طولانی تر مشکل ساز نباشد
- امنیت ضروری باشد (همه لایبرری ها را از کد منبع رسمی کامپایل می شوند)

زمانی از پیوند داینامیک استفاده می کنیم که:

- execution performance ضروری نباشد
- نیاز به استقرار مکرر باشد
- چندین برنامه از لایبرری های یکسانی در ماشینی با فضای دیسک محدود استفاده کنند
- امنیت ضروری نباشد (به این دلیل است که کاربر ممکن است قبلاً یک نسخه از Qt یا یک لایبرری خاص را که توسط شخص ثالث ساخته شده است نصب کرده باشد که شما نمی توانید قانونی بودن و درستی آن را تضمین کنید)

پلاگین

پلاگین یک library برای گسترش برنامه است که در run_time اجرا می شود. مشخصه اصلی یک پلاگین کلاس ابسترکتی به اسم interface است که تمام متد های آن از نوع pure virtual است که کلاس پلاگین از این interface به ارث می برد. در زمان اجرا برنامه QPluginLoader پلاگین را بارگذاری می کند.

در کلاس اینترفیس برای اینکه به query این امکان رو بدیم در run_time یک interface را در پلاگین پیاده سازی کند از Q_DECLARE_INTERFACE استفاده می کنیم.

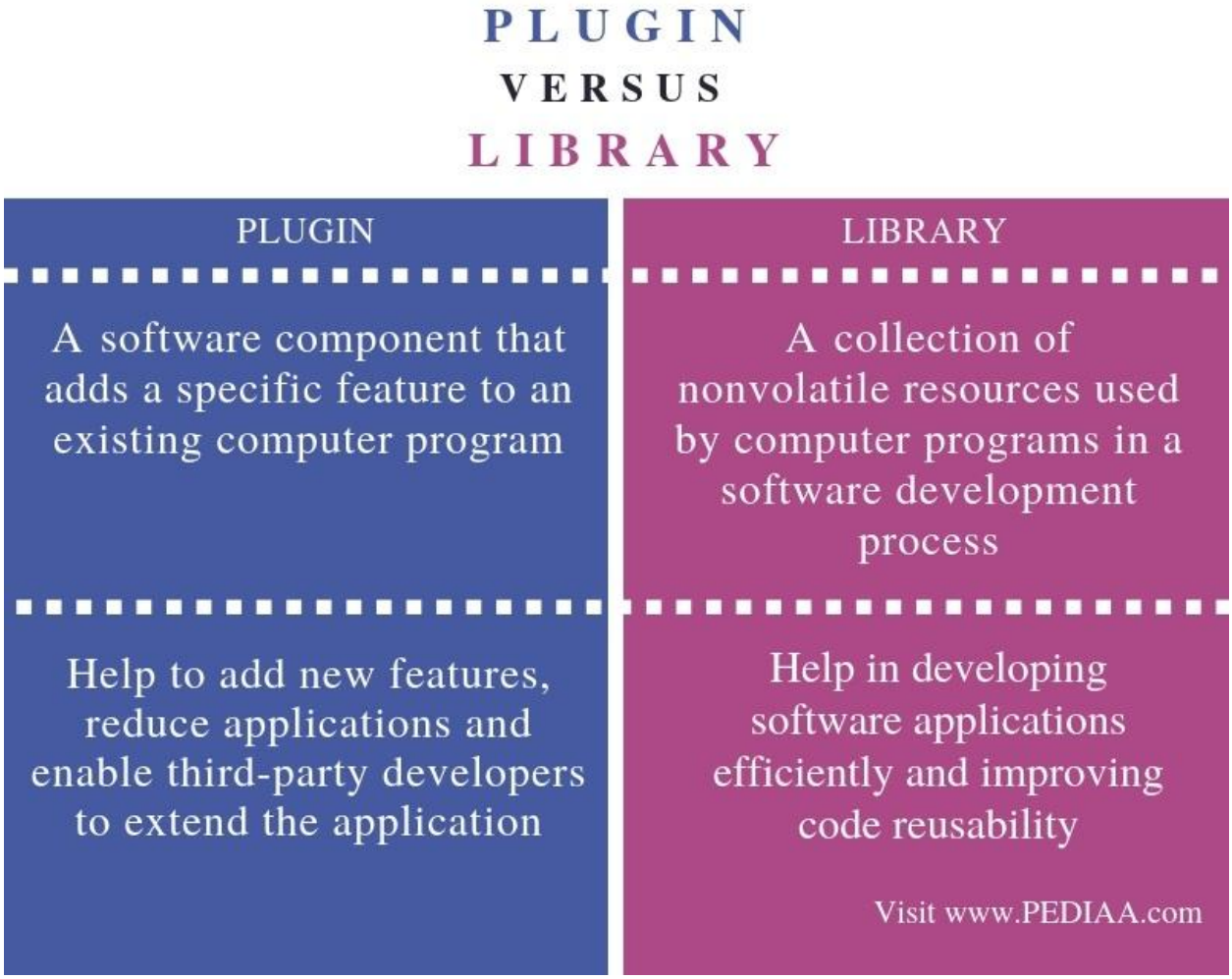
Q_DECLARE_INTERFACE

ماکرو Q_DECLARE_INTERFACE() در interface دو ورودی میگیرد اولی اسم interface و دومی یک استرینگی است که اینترفیس را به روش منحصر به فردی شناسایی میکند (Java package name" syntax)

نکته: اگر بعداً اینترفیس ها را تغییر دهیم، باید از استرینگ دیگری برای شناسایی اینترفیس جدید استفاده کنیم. در غیر این صورت، برنامه ممکن است از کار بیفتد. بنابراین ایده خوبی است که ورژن را در استرینگ اضافه کنید.

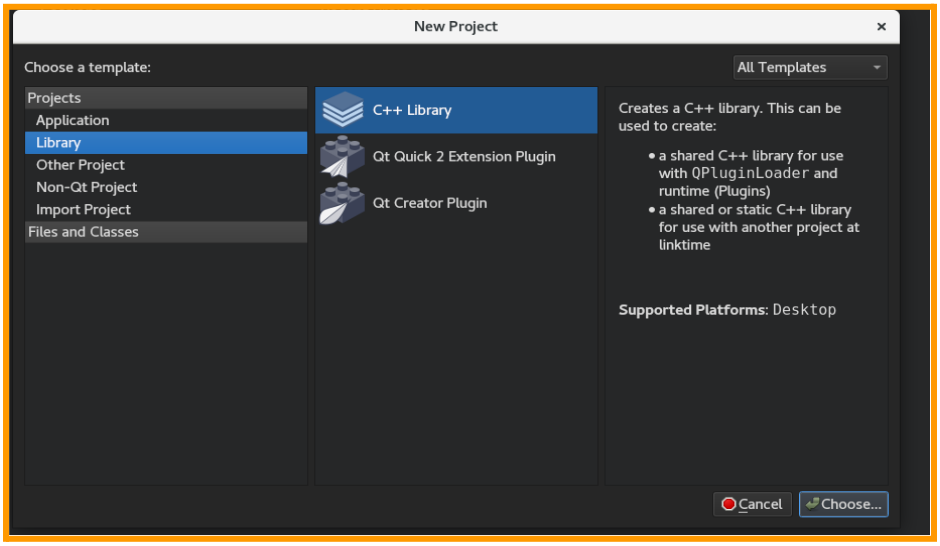
کاربرد پلاگین

ما برای گسترش یک برنامه و افزودن ابزار های مختلف به آن از پلاگین استفاده می کنیم که یک بار آن را نوشته و آن را در برنامه هایی که اینترفیس منحصر به فرد پلاگین را دارند استفاده می کنیم



نحوه ایجاد لایبرری

برای ساخت لایبرری ابتدا وارد new project شده در قسمت لایبرری مطابق شکل زیر C++ لایبرری را انتخاب می کنیم



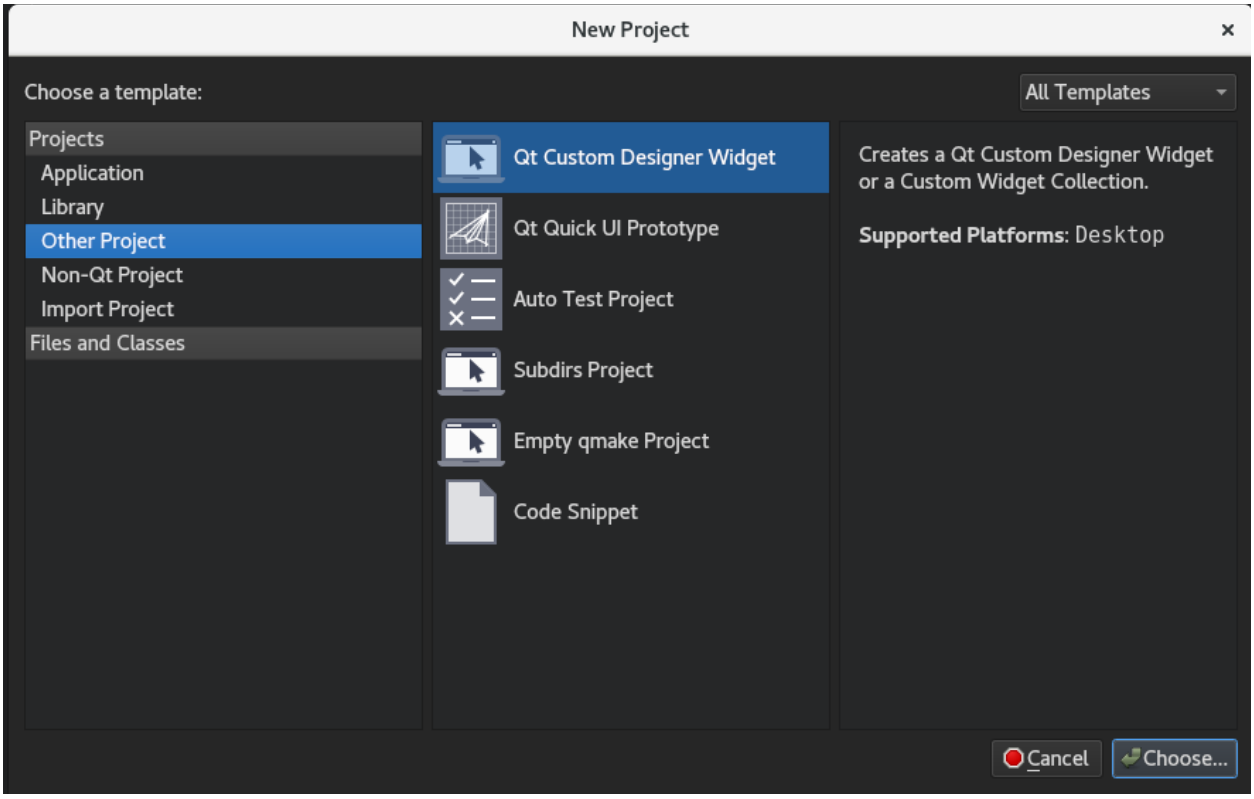
سپس در قسمت type نوع لایبرری را انتخاب می کنیم (static/dynamic) .

اضافه کردن لایبرری به پروژه

ابتدا روی پروژه مورد نظر راست کلیک کرده گزینه add library را انتخاب کرده در پنجره باز شده اگر لایبرری در بیلد پروژه تان قرار دارد از گزینه internal library را انتخاب می کنیم در غیر این صورت گزینه external library را انتخاب کرده اگر internal library را انتخاب کرده باشیم در پنجره باز شده در قسمت لایبرری لایبرری مورد نظر را انتخاب کرده و در قسمت پلتفرم سیستم عامل مورد استفاده را انتخاب می کنیم درمورد external هم ابتدا در قسمت لایبرری تایپ نوع لایبرری بر اساس سیستم عامل مورد استفاده را انتخاب می کنیم سپس در قسمت لایبرری فایل آدرس لایبرری را می دهیم و آدرس بیلد لایبرری مورد نظر را به include path میدهیم حال در پروژه در قسمتی که به لایبرری نیاز داریم کافی است هدر لایبرری مورد نظر را include کنیم.

نحوه ایجاد پلاگین

ابتدا اینترفیس پلاگین مورد نظر را در پروژه ایجاد می کنیم (در اینجا از اینترفیس QDesignerCustomWidgetInterface استفاده شده) سپس بر روی new project کلیک کرده مطابق شکل زیر:



Qt Custom Designer Widget را انتخاب می کنیم در پنجره Custom Widget List لیست پلاگین هایی از نوع ویجت که می خواهیم را درست می کنیم اگر بیشتر از یک ویجت به لیست داده باشیم در پنجره بعد اسم collection class را می دهیم سپس وارد پروژه می شویم به صورت پیش فرض پلاگین ما داینامیک هست برای استاتیک کردنش کافیه وارد فایل pro. پلاگین شویم و مانند شکل زیر CONFIG را به حالت plugin static در بیاریم:

اضافه کردن پلاگین به پروژه

بسته به داینامیک بودن یا استاتیک بودن پلاگین اضافه کردن آن ها به پروژه متفاوت می باشد

داینامیک

با استفاده از plugin loader مطابق شکل زیر عمل می کنیم

```cpp
void MainWindow::LoadPlugin()
{
    const auto staticInstances = QPluginLoader::staticInstances();
    for (QObject *plugin1 : staticInstances){
        auto iClock = qobject_cast<QDesignerCustomWidgetInterface *>(plugin1);
        QWidget* StatPlugin = iClock->createWidget(nullptr);
        ui->gridLayout_2->addWidget(StatPlugin,0,2);
    }
    QDir pluginsDir;
    pluginsDir = QDir(qApp->applicationDirPath());
    pluginsDir.cd("..");
    pluginsDir.cd("DynPlugClock");
    const auto entryList = pluginsDir.entryList(QDir::Files);
    for (const QString &fileName : entryList) {
        QPluginLoader loader(pluginsDir.absoluteFilePath(fileName));
        QObject *plugin = loader.instance();
        if(plugin){
            auto iClock2 = qobject_cast<QDesignerCustomWidgetInterface *>(plugin);
            QWidget* DynPlugin = iClock2->createWidget(nullptr);
            ui->gridLayout_2->addWidget(DynPlugin,0,3);
        }
}}
```

استاتیک

برای استاتیک ابتدا در قسمت pro. پروژه، آدرس و نام پلاگین مطابق شکل زیر اضافه میکنیم

```
25
26 FORMS += \
27     mainwindow.ui
29 LIBS +=-L../staticlibclk    آدرس پلاگین
30 macx-xcode {
31     LIBS += -lstaticlibclockplugin$($${QMAKE_XCODE_LIBRARY_SUFFIX_SETTING})
32 } else {
33     LIBS += -lstaticlibclockplugin        نام پلاگین
34 }
35 CONFIG += install_ok
36
37
38 # Default rules for deployment.
39 qnx: target.path = /tmp/$${TARGET}/bin
40 else: unix:!android: target.path = /opt/$${TARGET}/bin
41 !isEmpty(target.path): INSTALLS += target
```

بعد وارد main پروژه شده و با استفاده از Q_IMPORT_PLUGIN(your static plugin) پلاگین را اضافه می کنیم سپس مانند شکل زیر عمل می کنیم

```
void MainWindow::LoadPlugin()
{
    const auto staticInstances = QPluginLoader::staticInstances();
    for (QObject *plugin1 : staticInstances){
        auto iClock = qobject_cast<QDesignerCustomWidgetInterface *>(plugin1);
        QWidget* StatPlugin = iClock->createWidget(nullptr);
        ui->gridLayout_2->addWidget(StatPlugin,0,2);
    }
    QDir pluginsDir;
    pluginsDir = QDir(qApp->applicationDirPath());
    pluginsDir.cd("..");
    pluginsDir.cd("DynPlugClock");
    const auto entryList = pluginsDir.entryList(QDir::Files);
    for (const QString &fileName : entryList) {
        QPluginLoader loader(pluginsDir.absoluteFilePath(fileName));
        QObject *plugin = loader.instance();
        if(plugin){
            auto iClock2 = qobject_cast<QDesignerCustomWidgetInterface *>(plugin);
            QWidget* DynPlugin = iClock2->createWidget(nullptr);
            ui->gridLayout_2->addWidget(DynPlugin,0,3);
        }

}}
```