KDE Frameworks on Android

Akademy 2019

Volker Krause

vkrause@kde.org
@VolkerKrause

Why Android?

- <insert same arguments as for Windows>
- LineageOS
- Proving ground for apps on the way to Plasma Mobile

Why KF5 on Android?

- Kirigami
- Portability beyond Qt’s feature scope (e.g. notifications)
- Lots of existing application code

Isn’t Android “just” Linux?

- Single process sandboxed bundle (implies e.g. no global file access)
- No D-Bus
- No platform-wide components/services can be deployed
- Platform API is mostly Java
- Typically no QWidgets, very different WM

Frameworks Status

- Working:
  - KArchive, KCoreAddons, KI18n, KCodecs, KConfig, Prison, KHolidays, etc.
  - Kirigami, KNotification
- Make sense but need work:
  - Solid, KCrash, KPeople, KDNSSD
  - KDeclarative, KIO, Sonnet

Frameworks Status (cont.)

- Can be useful as porting aids:
  - KXmlGui, KService, KParts, KConfigWidgets
- Make no sense on Android:
  - KDBusAddons, KWayland, KGlobalAccel, KAuth, KDED, KInit, KNotifyConfig, KWindowSystem
- Need investigation
  - Purpose, KIconThemes, KWallet

Challenges

- Mix of abstraction and platform
- Mix of core functionality and UI
  - Error handling via message boxes
  - Heavy widget dependency
- All this also matters for Plasma Mobile!

So much for the theory...

Setup

- See Aleix's work

  https://community.kde.org/Android
  `docker run -ti --rm kdeorg/android-arm-sdk`

- SDK/NDK version – use latest, does not impact compatibility
- API level: KF5 uses 21
- Qt: use latest, still receiving very fundamental improvements

Calling Java from C++

- Java Native Interface (JNI), via QAndroidExtras
- Error prone JNI signature notation
- `QtAndroid` provides common entry points

```
QAndroidJniObject::callStaticObjectMethod(
  "android/net/Uri", "parse",
  "(Ljava/lang/String;)Landroid/net/Uri;",
  QAndroidJniObject::fromString(url)
    .object<jstring>());
```

Calling C++ From Java

- Declare native methods in Java:

```
public native void myMethod(String data);
```

- Register methods in C++:

```
static const JNINativeMethod methods[] = {
  {"myMethod", "(Ljava/lang/String;)V", (void*)myMethod}
};

Q_DECL_EXPORT jint JNICALL JNI_OnLoad(JavaVM *vm, void*)
{
  JNIEnv *env = nullptr;
  vm->GetEnv((void**)&env, JNI_VERSION_1_4);
  auto cls = env->FindClass("org/kde/myApp/MyClass");
  env->RegisterNatives(cls, methods, 1);
}
```

File Access

- Regular file I/O only possible inside the application, or with extra permission
- Interaction with other apps and platform via content:// URLs
- Basic support in Qt >= 5.13
- But: QUrl::isLocalFile(), QSaveFile, etc

Intents

- The way to interact with other apps and sometimes the platform (e.g. file dialogs)
- Generalized form of “open this file in an appropriate application”

Shipping Java Code

- JAR – used by Qt
  - Build directly with CMake via `add_jar()`
- AAR – used by KF5::Notifications
  - Can have dependencies
  - Can contain resources and manifest fragments
  - Has to be built via Gradle – FindGradle.cmake

androiddeployqt

- Library and QML module dependencies are automatically added
- Use QRC for QML files, images, etc
- Add KF5Foo-android-dependencies.xml for plugins and AAR/JAR files
  - Plugins with missing dependencies are skipped
  - Workaround: Link app against plugin dependencies

KAndroidExtras?

- Addon framework for QAndroidExtras
- Avoid proliferation of string-based JNI code
  - As implementation detail for platform abstractions
  - For things that have no cross-platform equivalent

Conclusion

- Still work needed to avoid platform-specific code in applications
- Review for KF6
  - Clearly separate platform abstraction and platform implementation
  - Move QML bindings to their corresponding modules
  - Review widget dependencies

Questions?

KF6 BoF
Monday 09:30 U1-04
Android BoF
Thursday 16:30 U2-02