README.md

ODK JavaRosa

JavaRosa is a Java library for rendering forms that are compliant with ODK XForms spec. It is at the heart of many of the ODK tools. ODK JavaRosa is a fork of JavaRosa 1.0 that has been modified to NOT run on J2ME devices. The key differences are:

- Regularly updated to ensure spec compliance
- KoBo additional instance defn. and filter paths
- remember all bind attributes and any additional attributes on `<input>`, `<select>`, `<group>`, etc. statements
- numerous enhancements and contributions from SurveyCTO and others.
- J2ME sub-projects removed

ODK JavaRosa is part of ODK, a free and open-source set of tools which help organizations author, field, and manage mobile data collection solutions. Learn more about the ODK project and its history here and read about example ODK deployments here.

- ODK website: https://getodk.org
- ODK forum: https://forum.getodk.org
- ODK developer Slack chat: https://slack.getodk.org
- ODK developer wiki: https://github.com/getodk/getodk/wiki

Requirements

JavaRosa works on Android API level 21+ (with desugaring enabled) and Java 8+.

Setting up your development environment

1. Fork the javarosa project (why and how to fork)
2. Clone your fork of the project locally. At the command line:

```
git clone https://github.com/YOUR-GITHUB-USERNAME/javarosa
```

We recommend using IntelliJ IDEA for development. On the welcome screen, click `Import Project`, navigate to your javarosa folder, and select the `build.gradle` file. Use the defaults through the wizard. Once the project is imported, IntelliJ may ask you to update your remote Maven repositories. Follow the instructions to do so.

Building the project

To build the project, go to the `View` menu, then `Tool Windows > Gradle`. `build` will be in `javarosa > Tasks > build > build`. Double-click `build` to package the application. This Gradle task will now be the default action in your `Run` menu.

To package a jar, use the `jar` Gradle task.

Running benchmarks

JavaRosa can be used to parse and fill very large forms on inexpensive devices and so it's important to keep an eye on performance. Benchmarks using JMH and the JMH Gradle plugin have been introduced to compare relative performance as code changes are made. We have found that running these benchmarks with the `jmh` Gradle task or through IntelliJ can produce inconsistent results. The most reliable way we have found to run them is to first build a `jar` with the `jmhJar` Gradle task and then to run the jar:

```
    java -jar build/libs/javarosa-jmh.jar ChildVaccination
```

This also makes it easy to selectively run a subset of benchmarks by including a regular expression as an argument. In the example above, only benchmarks that include the text "ChildVaccination" will be executed. Run `java -jar build/libs/javarosa-jmh.jar -h` to see other JMH configuration options.

While benchmarks can help identify relative performance improvements or regressions, they are not always a reliable proxy for how code will perform in a realistic context. Profiling is helpful for identifying parts of the code that are worth analyzing and optimizing as well as to validate that any performance changes have the intended effects.

Contributing code

Any and all contributions to the project are welcome. ODK JavaRosa is used across the world primarily by organizations with a social purpose so you can have real impact!

If you're ready to contribute code, see the contribution guide.

Downloading builds

Per-commit debug builds can be found on CircleCI. Login with your GitHub account, click the build you'd like, then find the JAR in the Artifacts tab under $CIRCLE_ARTIFACTS.

Publishing the jar to OSSRH and Maven Central

Project maintainers have the private keys to upload signed jars to Sonatype's OSS Repository Hosting (OSSRH) service which is then synced to Maven's Central Repository. This process is outlined here.

While Gradle is the default build tool for all ODK tools (including this one), Maven is used for for publishing the jar because OSSRH's Gradle support is unreliable (e.g., snapshots don't always update). This means version and dependency changes must be made in both `build.gradle` and `pom.xml`.

Deviations from OSSRH's documentation are that maintainers use `gpg2` v2.1 and greater (not `gpg`), the latest versions of the Maven plugins in `pom.xml`, and a `secrets.xml` file that include the GPG home directory, key name, and pass phrase. All that is needed in the GPG home directory is `private-keys-v1.d` and `pubring.gpg`.

```
<!-- secrets.xml -->
<settings>
    <servers>
```

```
        <server>
            <id>ossrh</id>
            <username>getodk</username>
            <password>very-secure-password</password>
        </server>
    </servers>
    <profiles>
        <profile>
            <id>ossrh</id>
            <activation>
                <activeByDefault>true</activeByDefault>
            </activation>
            <properties>
                <gpg.executable>/path/to/gpg2</gpg.executable>
                <gpg.homedir>/path/to/javarosa/gpg/folder</gpg.homedir>
                <gpg.keyname>1234ABCD</gpg.keyname>
                <gpg.passphrase>very-secure-passphrase</gpg.passphrase>
            </properties>
        </profile>
    </profiles>
</settings>
```

Official releases are built by a Github action when a commit is tagged. Before tagging a release:

1. Update the version in `build.gradle` and `pom.xml` and merge the changes to master.
    - Use `x.x.x-SNAPSHOT` for snapshots releases and `x.x.x` for production releases.

To manually generate official signed releases, you'll need the GPG folder, GPG passwords, a configured `secrets.xml` file.

1. Run `mvn -v` to confirm the Java version and vendor used to build the release.
2. In the repo folder, run `mvn -s secrets.xml clean deploy` to publish.
    - If successful, both snapshots and production releases will appear in OSSRH here.
    - Production releases are automatically synced to Central here a few hours later.

Don't forget to update the `build.gradle` files in any downstream tools (e.g., ODK Collect, ODK Briefcase) to the newest version!

Releases 39

v4.1.0 Latest
Mar 24, 2023

+ 38 releases

Packages

No packages published

Contributors 38

+ 27 contributors

Languages

Java 99.7% **Other** 0.3%