RTSDK Java 2.2.3.L1

INSTALLATION GUIDE

1 Overview

Real-Time SDK (RTSDK) packages are specific to the product language (C/C++, C#, or Java) and include both the Enterprise Transport and Message API products. This guide describes procedures to install and build RTSDK Java, applying to versions 2.2.2.L1 and higher. Because steps to install are general to the RTSDK, they apply to both ETA and EMA.

RTSDK supports open sourcing and uses standards-based, freely-available open source tools to provide additional flexibility and benefit.

You can obtain RTSDK Java Archive (JAR) files and their dependencies from the RTSDK Java package or build open source libraries using code from GitHub. Additionally, JAR files for RTSDK Version 1.2 and higher are hosted in Maven Central.

2 Obtaining RTSDK

You can obtain the RTSDK in one of the following ways:

- Download RTSDK from LSEG. For details, see Section 2.1.
- Obtain RTSDK from GitHub. For details, see Section 2.2.
- Download RTSDK from Maven Central. For details, see Section 2.3.

If you download RTSDK from LSEG, please download two RRG packages: RTSDK RRG (platform and language specific) and RTSDK BinaryPack RRG (contains libraries for all support platforms and languages). The BinaryPack RRG contains closed source libraries which permit users to build and link all dependent libraries to have a fully function product. To combine both RRG packages, see Section 2.1.1.

If you clone RTSDK from GitHub, the binary pack is downloaded automatically. For more information, see Section 2.2.2.

2.1 Download RTSDK Packages from LSEG

Download the RRG and BinaryPack RRG packages from the following locations:

- LSEG Software Downloads page: https://myaccount.lseg.com/en/downloadcenter.

  Search downloads for product family, “MDS - API”, and product, “Real-Time SDK”.
- Developer Community Portal: https://developers.lseg.com/en/api-catalog/real-time-opnsrc/rt-sdk-java/download

For the RRG package, the Java archive file names are in this format: **Real-Time-SDK-<*version*>.java.rrg**.

Starting with version 2.2.2.L1, the BinaryPack content is available as a separate RRG package, requiring you to download and extract two archives:

- RTSDK RRG package: **Real-Time-SDK-<*version*>.java.zip**.

  The extracted archive contains a **setup** directory with the package in this format: **RTSDK-<*version*>.java.rrg.zip**. Once extracted, this package contains prebuilt RTSDK jars.

  Example: Download **Real-Time-SDK-2.2.2.L1.java.zip** which contains a **setup** directory containing the **RTSDK-2.2.2.L1.java.rrg.zip**. This zip contains prebuilt jars.

- RTSDK BinaryPack RRG package: **Real-Time-SDK-BinaryPack-<*version*>.zip**.

  The extracted archive contains a **setup** directory with the package in this format: **RTSDK-BinaryPack-<*version*>.rrg**. This package contains closed-source binaries. It is not platform specific and has content for all flavors/platforms of API. You can use this package to build RTSDK examples.

Note: To use the BinaryPack RRG package for building RTSDK examples, combine both packages as described in Section 2.1.1.

2.1.1 Using Binary Pack to Build RTSDK

To build RTSDK examples, combine the content of both packages before doing a build.

▶ To combine content of the RRG and BinaryPack RRG:

1. Download and extract both packages as described in Section 2.1.
2. Place the **RTSDK-BinaryPack-<*version*>.rrg** directory into the **RTSDK-<*version*>.java.rrg** directory.
3. Rename **RTSDK-BinaryPack-<*version*>.rrg** to **RTSDK-BinaryPack**.

Example:

1. Download **Real-Time-SDK-2.2.2.L1.java.zip** and **Real-Time-SDK-BinaryPack-2.2.2.L1.zip**.
2. Extract content from the **setup** directory of each archive: **RTSDK-2.2.2.L1.java.rrg** and **RTSDK-BinaryPack-2.2.2.L1.rrg**.
3. Move the **RTSDK-BinaryPack-2.2.2.L1.rrg** directory into the **RTSDK-2.2.2.L1.java.rrg** directory.
4. Rename **RTSDK-BinaryPack-2.2.2.L1.rrg** to **RTSDK-BinaryPack**.

2.2 Obtain RTSDK from GitHub

To obtain RTSDK from GitHub, do one of the following:

- Download packages from GitHub
- Clone the GitHub repository

2.2.1 Download RTSDK Packages from GitHub

Download both source code and binary pack packages from the GitHub RTSDK releases page:

1. Browse to https://github.com/Refinitiv/Real-Time-SDK/releases.
2. From the **Assets** drop-down section, download the packages:
   - RTSDK RRG package: **Source code zip** or **tar.gz** archive.
   - RTSDK BinaryPack RRG package: **RTSDK-BinaryPack-<*version*>.zip** or **tar.xz** archive.

2.2.2 Clone GitHub Repository

Clone the RTSDK GitHub repository from https://github.com/Refinitiv/Real-Time-SDK.

To clone the repository, use the following command:

```
git clone https://github.com/Refinitiv/Real-Time-SDK.git
```

Note: An RTSDK clone built using gradle automatically downloads the RTSDK binary pack on behalf of the user, assuming user has access to download from GitHub.

2.3 Downloading RTSDK and Dependencies from Maven Central

For ease of product use, LSEG maintains its RTSDK Jar files on Maven Central. Dependencies of RTSDK Jar files (i.e., Apache, Mockito, etc) are maintained on Maven Central by their third party producers.

RTSDK does consist of closed source libraries which are available in a BinaryPack. This must be downloaded either from GitHub (see Section 2.2) or obtained from LSEG as a BinaryPack RRG archive (see Section 2.1).

Because you can download RTSDK libraries and dependencies from Maven Central using several different tools, specific procedural instructions are not included here. Maven uses the following syntax to specify RTSDK dependencies:

```
<dependency>
    <groupId>com.refinitiv.ema</groupId>
    <artifactId>ema</artifactId>
    <version>3.8.2.0</version>
</dependency>

<dependency>
    <groupId>com.refinitiv.eta</groupId>
    <artifactId>eta</artifactId>
    <version>3.8.2.0</version>
</dependency>

<dependency>
    <groupId>com.refinitiv.eta.valueadd</groupId>
    <artifactId>etaValueAdd</artifactId>
    <version>3.8.2.0</version>
</dependency>

<dependency>
    <groupId>com.refinitiv.eta.valueadd.cache</groupId>
    <artifactId>etaValueAddCache</artifactId>
    <version>3.8.2.0</version>
</dependency>
```

```
<dependency>
    <groupId>com.refinitiv.eta.ansi</groupId>
    <artifactId>ansipage</artifactId>
    <version>3.8.2.0</version>
</dependency>
```

Gradle uses the following syntax to specify RTSDK dependencies:

```
compile group: 'com.refinitiv.ema', name: 'ema', version: '3.8.2.0'
compile group: 'com.refinitiv.eta', name: 'eta', version: '3.8.2.0'
compile group: 'com.refinitiv.eta.valueadd', name: 'etaValueAdd', version: '3.8.2.0'
compile group: 'com.refinitiv.eta.valueadd.cache', name: 'etaValueAddCache',
        version: '3.8.2.0'
compile group: 'com.refinitiv.eta.ansi', name: 'ansipage', version: '3.8.2.0'
```

3 Building RTSDK

3.1 Building RTSDK with Gradle

The RTSDK RRG packages, obtained from LSEG's GSG or via the Developer Community Portal (see Section 2.1), include all needed Java dependencies: you do not need to use Gradle to download or build your RTSDK dependencies. However, you will still need to run Gradle to build examples.

The package includes **build.gradle** files throughout its directories to assist in rebuilding libraries and examples via a single command.

The RTSDK source in GitHub does not include Java dependencies but includes all Gradle files necessary to download and build these dependencies from Maven Central. The build.gradle files specify the location of the product's Java dependencies (which can be local or remote). If needed, you can configure Gradle files to pull dependencies (such as Apache or Mockito) from other URLs or locations (see Section 2.3).

Warning! To run Gradle for building libraries for code obtained from GitHub, you must:

- Have access to the Internet
- Specify any proxy (i.e., a firewall) that you use on your network. For instructions on specifying a proxy, refer to Gradle's instructions at the following link: https://docs.gradle.org/current/userguide/build_environment.html#sec:accessing_the_web_via_a_proxy.

▶ To download and build dependencies using Gradle:

1. Clone or otherwise obtain code from GitHub (https://github.com/Refinitiv/Real-Time-SDK).
2. Open a command (on Windows) or terminal (on Linux) window.
3. Change your directory to the RTSDK Java root directory (i.e., **Real-Time-SDK/Java**).
4. Issue the appropriate Gradle command as follows:
   - On Windows, issue the command: `gradlew.bat jar`

On Linux, issue the command: `./gradlew jar`

3.2 Building Examples

The RTSDK requires that you use Gradle to build the Java examples for both EMA and ETA. Because the RTSDK comes with a large number of examples, this section discusses how to use Gradle to access the entire list of examples, and basic syntax for building and running an example.

3.2.1 Listing RTSDK Examples

The following procedure assumes that you've already downloaded, and if necessary, built the package's JAR files and dependencies (refer to Section 5).

Before running an ETA or EMA example, you need to know the example's name (for details on using Gradle to run examples, refer to Section 3.2.3). You can use Gradle to list all ETA and EMA example names.

The following diagram is an example of what Gradle prints to the screen (EMA and ETA examples display under the section **Other tasks** toward the end of the command's output):

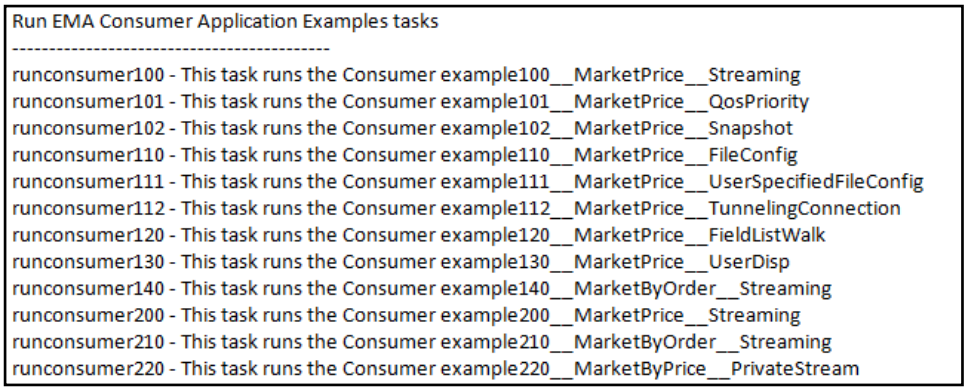


Figure 1. Gradle Output with EMA Consumer Example Names

▶ Using Gradle to list EMA and ETA examples:

1. Open a command (on Windows) or terminal (on Linux) window.
2. Change your directory to the RTSDK root directory.
   - For packages downloaded from GSG or the Developer Portal, the package directory is named **RTSDK**, where ***Version*** is represented by 3 digits and a letter (e.g., **RTSDK-2.2.2.L1.java.rrg**), while the RTSDK root directory is **RTSDK/Java**.
   - For packages pulled from GitHub, the RTSDK root directory is **Real-Time-SDK/Java**.
3. To view the list of ETA examples, issue the appropriate command as follows:
   - On Windows, issue the command: **`gradlew.bat Eta:Applications:tasks --all`**
   - On Linux, issue the command: **`./gradlew Eta:Applications:tasks --all`**
4. To view the list of EMA examples, issue the appropriate command as follows:
   - On Windows, issue the command: **`gradlew.bat Ema:Examples:tasks --all`**
   - On Linux, issue the command: **`./gradlew Ema:Examples:tasks --all`**

3.2.2 Enabling Logging in EMA

In EMA, to enable logging when running examples, you must activate the logging section in the **build.gradle** file in the ***PackageDirectory*/Ema/Examples** directory. Remove the forward slashes (**`//`**) from the **`jvmArgs`** line as follows:

```
tasks.withType ( JavaExec ) {
    // to enable the logger comment out the line below
    jvmArgs += "-Djava.util.logging.config.file=../Core/src/main/resources/
            logging.properties"
}
```

3.2.3 Building and Running an Example Using Gradle

The following procedure assumes that you've already identified the name of the example you want to run (for details, refer to Section 3.2.1 to list examples).

▶ To use Gradle in building and running an example:

1. Open a command (on Windows) or terminal (on Linux) window.
2. Change your directory to the RTSDK root directory (i.e., **RTSDK**).

   Where ***Version*** is represented by 3 digits and a letter (e.g., **RTSDK**).
3. To build and run an example, issue the appropriate command as follows:
   - On Windows, issue the command: **`gradlew.bat runExampleName [--args="arguments"]`**
   - On Linux, issue the command: **`./gradlew runExampleName [--args="arguments"]`**

   Where:
   - ***`runExampleName`*** is the name of the example you want to build and run. For example, issuing the command: **`gradlew.bat runconsumer270`** runs the EMA consumer **example270__SymbolList**.
   - ***`arguments`*** are options or arguments that you want to add to the Gradle command. The following is a Linux command illustrating the use of arguments when running Gradle:

```
./gradlew runVaConsumer --args="-c localhost:14002 DIRECT_FEED mp:TRI"
```

Tip:

- You can see a list of all possible arguments by passing the command: **`--args="-?"`**
- Instead of **`--args`**, you can use **`-PcommandLineArgs`**, which functions in an identical manner with the same arguments.

4 Package File and Directory Changes

Notable changes in the RTSDK package include:

- To support simultaneously hosting of files on Maven Central, JAR filenames have changed: all JAR files now include the package version (***Version***) as a suffix in their name (e.g., **eta-3.8.2.0.jar**). As a result of this change, you must update your class path(s) to use the appropriate filename(s).
- For multicast connections, use the appropriate JNI libraries, **librsslRelMcast** and **rsslEtaJNI/librsslEtaJNI**, located in the closed source **RTSDK-BinaryPack/Java/Eta/Libs/rssl** directory.
- For value add cache, use the appropriate JNI library, **rsslVACacheJNI/librsslVACacheJNI**, located in the closed source **RTSDK-BinaryPack/Java/Eta/Libs/rssl** directory.
- Directory structure changes.

Starting with RTSDK 1.2, Java code is built using Gradle (https://gradle.org/). You can use Gradle to do the following:

- Download JAR files and associated dependencies from Maven Central. Refer to Section 3.1.
- Build the RTSDK package's Java examples (refer to Section 3.2) and product source code (refer to Section 3).

The RTSDK package's structure (as illustrated in Table 1) is determined by how Gradle generates artifacts. Each subdirectory has its own **src** folder. All the source code in that folder can generate a jar file. Thus, the contents of the **ValueAdd** folder generate the **etaValueAdd-*Version*.jar**, the contents of the **Eta/Core** directory generate the **eta-*Version*.jar**, the contents of the **Ema/Core** directory generate the **ema-*Version*.jar**, and etc.

Starting in RTSDK 1.2, a new library **librsslRelMcast** in directory **RTSDK-BinaryPack/Java/Eta/Libs/rssl** accounts for the shared reliable multicast library. This library is dynamically loaded by **librssl** whenever the Reliable Multicast transport is selected.

Additionally, the DACS library was moved to directory **RTSDK-BinaryPack/Java/Eta/Libs**.

Starting with RTSDK 2.0.1, RTSDK supports a WebSocket Transport and introduces support for either an RWF or a JSON payload. Conversion from RWF to JSON (and from JSON to RWF) is built into **etajConverter-Version.jar**. Starting with version RTSDK-2.2.2.L1, RTSDK BinaryPack is available as a separate RRG and must be combined with RTSDK Java RRG. Refer to Section 2.1.1.

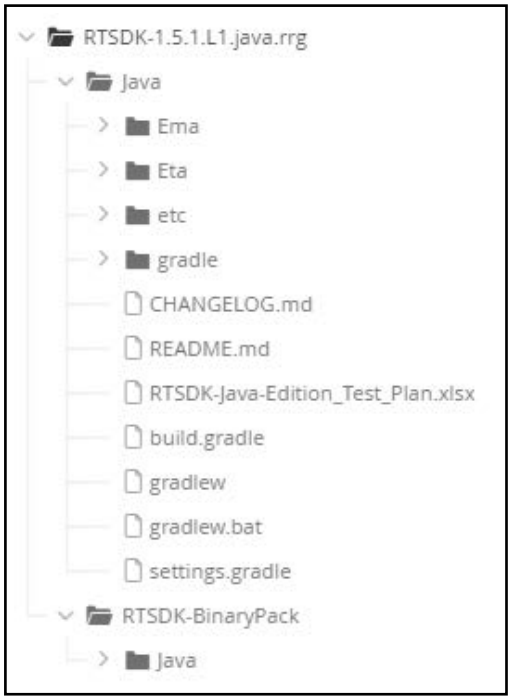
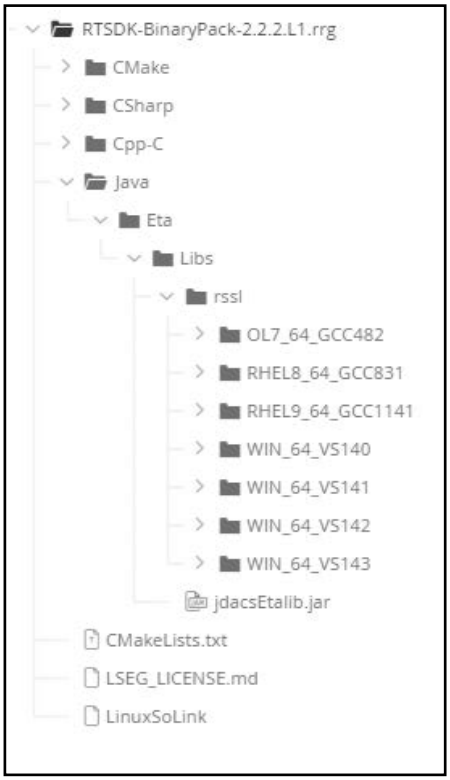
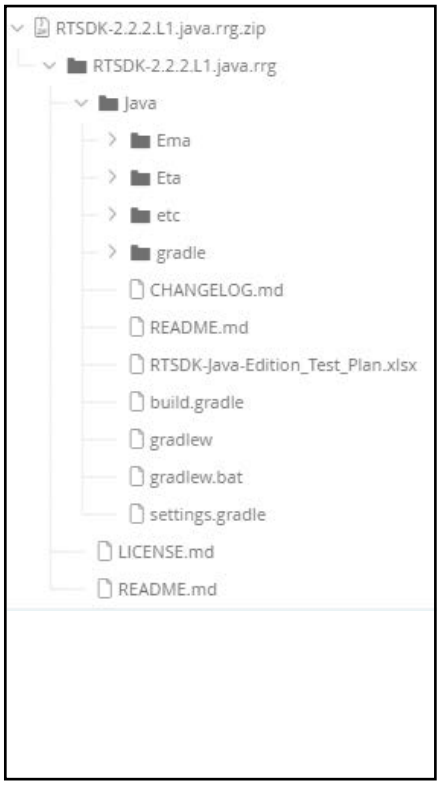
STRUCTURE STARTING WITH RTSDK 1.1.5	STRUCTURE STARTING WITH RTSDK 2.2.2	
RRG package:	BinaryPack RRG package:	RRG package:
		

Table 1: RTSDK Java Package Structures

5 Additional Resources

Encountering unique situations and scenarios is commonplace when using APIs in new and different ways, and in the face of different IDEs and build environments. For this reason, not every scenario can be addressed in this migration guide. For further information, tips, advice, etc., feel free to reach out to the wider open source community via the forums at the Developer Community Portal. The community also includes tutorials and other getting started details.

Document ID: RSKJ223IP.240
Date of issue: December 2024