RTSDK C/C++ 2.3.2.L1

INSTALLATION GUIDE

1 Overview

RTSDK packages are specific to the product language (C/C++, C#, or Java) and include both the Enterprise Transport API and Enterprise Message API products. This guide describes the procedures to install and build RTSDK C / C++, applying to RTSDK versions 2.2.1.L1 and higher. Because installation steps are specific to the RTSDK as a whole, the instructions apply to both Enterprise Transport and Enterprise Message APIs.

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

Developers must use CMake to dynamically generate the build files.

Note: Version 1.2 and later RTSDK applications are more memory-use intensive when initializing the Enterprise Transport API C library and when loading the dictionary.

2 Requirements and Limitations

Consider the following requirements and limitations before building RTSDK C/C++:

- The RTSDK C/C++ package uses Google Test in its unit tests. While the RTSDK automatically downloads Google Test whenever you run its unit tests, Google Test requires Python. So if you want to run the RTSDK unit tests, you must ensure you also have Python on your machine.
- The RTSDK C/C++ package requires CMake.
- LSEG does not support 32-bit builds in the Enterprise Message API.
- If you intend to use encrypted connections, you must also install OpenSSL.
- If you downloaded the RTSDK package from GitHub and run CMake, CMake automatically attempts to download needed libraries from GitHub including the RTSDK binary pack. Thus, you must have an Internet connection for CMake to successfully download the binaries in this manner.
- The RTSDK C/C++ package requires Python 3 to be installed.

Note: The package directory structure changed over time. For more information, see Section 5.

Document Version: 2.3.2.L1
Date of issue: December 2025
Document ID: RTSC232L1IP.250

LSEG DATA & ANALYTICS

3 Obtaining RTSDK

You can obtain RTSDK C/C++ in the following ways:

- Download RTSDK from LSEG. For details, see Section 3.1.
- Obtain RTSDK from GitHub. For details, see Section 3.2.

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. The CMake build from RRG package does not automatically download the BinaryPack. To combine both RRG packages, see Section 3.1.1.

If you clone RTSDK from GitHub, the binary pack is downloaded automatically. For more information, see Section 3.2.2.

Once you obtained RTSDK, you can build RTSDK with CMake as described in Section 4.

3.1 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-cc/downloads

For the RRG package, there are separate Linux and Windows library packages. The archive file names are in this format: **Real-Time-SDK-<version>.<*platform*>.rrg** where **<*platform*>** may be “linux” or “win” for Windows.

Starting with version 2.2.1.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*>.<*platform*>.zip**.

  The extracted archive contains a **setup** directory with the package in this format: **RTSDK-<*version*>.linux.rrg.tar.gz** or **RTSDK-<*version*>.win.rrg.zip**. This package contains prebuilt libraries for supported compilers for the platform chosen.

  Example: Download **Real-Time-SDK-2.2.1.L1.win.zip** which contains a **setup** directory containing the **RTSDK-2.2.1.L1.win.rrg.zip** directory. Once extracted, **RTSDK-2.2.1.L1.win.rrg** directory contains prebuilt libraries.
- 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.zip**. This package, once extracted, 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 4.1.1.

3.1.1 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 3.1.
2. Place the **RTSDK-BinaryPack-<*version*>.rrg** directory into the **RTSDK-<*version*>.win.rrg** directory.

3. Rename **RTSDK-BinaryPack-<*version*>.rrg** to **RTSDK-BinaryPack**.

Example:

1. Download **Real-Time-SDK-2.2.1.L1.win.zip** and **Real-Time-SDK-BinaryPack-2.2.1.L1.zip**.
2. Extract content from the **setup** directory of each archive: **RTSDK-2.2.1.L1.win.rrg** and **RTSDK-BinaryPack-2.2.1.L1.rrg**.
3. Move the **RTSDK-BinaryPack-2.2.1.L1.rrg** directory into the **RTSDK-2.2.1.L1.win.rrg** directory.
4. Rename **RTSDK-BinaryPack-2.2.1.L1.rrg** to **RTSDK-BinaryPack**.

3.2 Obtain RTSDK from GitHub

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

- Download packages from GitHub
- Clone the GitHub repository

3.2.1 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.

For information on Linux and Windows specific package formats and the packages content, see Section 3.1.

3.2.2 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 CMake automatically downloads the RTSDK binary pack on behalf of the user, assuming user has access to download from GitHub.

4 Building RTSDK with CMake

The RTSDK includes CMake configuration files (**CMakeLists.txt**) in strategic directories. You must use CMake to configure a build tree. CMake generates cleaner, more concise build environment files that correspond to users' platform and OS. In addition, it enables the creation of build environments on platforms that users wish to leverage, even if unsupported by the RTSDK product.

The RTSDK package includes a top-level, entry point for CMake (**CMakeLists.txt**), which CMake uses when you run the program. From this master file, CMake processes all downstream **CMakeLists.txt** files in the source tree to generate associated **Solution** and **vcxproj** files[1] (on Windows), or **Makefile** files (on Linux) in a build directory that you specify. After this process, you can compile your RTSDK in the same way as previous RTSDK versions (i.e., by running Make on Linux or by using Visual Studio on Windows). For details on configuring the RTSDK with CMake, refer to Section 5.5.

For both Windows and Linux, starting in version 1.5.1 with the introduction of support for Visual Studio 2019, LSEG supports only the use of CMake version 3.14 or later. Starting with version 2.0.8.L1, with the introduction of support for Visual Studio 2022, CMake version of 3.21 or later must be used. You can download CMake from https://cmake.org/download/.

4.1 Building on Windows

▶ To run CMake in a Windows environment:

1. Obtain RTSDK. For details, refer to Section 3.
2. If obtaining RTSDK by downloading RRG packages, extract the contents of the RTSDK packages as needed. Refer to Section 3.
3. Note the name of the top-level extracted directory (i.e., on Windows, the name might be something like **RTSDK-2.2.1.L1.win.rrg** or, if this is a GitHub clone, the name might be **Real-Time-SDK**).

   The name of this extracted directory is referred to as ***sourceDir*** for the remainder of this procedure.
4. In Windows Explorer, navigate to the directory that contains ***sourceDir***.
5. Press and hold down SHIFT, right-click the directory, and in the context menu, click **Open command window here**.
6. Issue the command:

```
cmake --help | -HsourceDir -BbuildDir -G "VisualStudioVersion" [-Doption ... ]
```

   Where:

   - **--help** outputs a list of available command options and generator types.
   - ***sourceDir*** is the directory in which the top-level CMake entry point (**CMakeLists.txt**) resides. By default, when you build using the **Solution** and **vcxproj** files, output is sent to directory specified in ***SourceDir***.
   - ***buildDir*** is the CMake directory where built binaries are stored. This directory is created if it does not exist.
   - ***VisualStudioVersion*** is the Visual Studio version. For example, **Visual Studio 11 2012 Win64**. Valid values for ***VisualStudioVersion*** are:
     - "Visual Studio 17 2022" -A x64
     - "Visual Studio 16 2019" -A x64

1. CMake refers to such files as 'targets'

- "Visual Studio 15 2017 Win64"
- "Visual Studio 14 2015 Win64"

Note:
- If you do not explicitly specify **`Win64`**, by default CMake builds the 32-bit version.
- A list of visual studio versions can be obtained by typing **`cmake --help`**

- ***`option`*** is a command line option and its associated value (e.g., **`-DBUILD_EMA_UNIT_TESTS=OFF`**). You can control aspects of how CMake builds the RTSDK by using command line options (for further details on the use of options, refer to Section 4.4).

The **`cmake`** command builds all needed **Solution** and **vcxproj** files (and other related files) in the CMake build directory and may be built using Visual Studio. Compiled output (after running make or from visual studio make) is located in its associated source directories (i.e., example executables are in the **Executables** directory and libraries (e.g., **libema.lib**, **librssl.lib**) in the **Libs** directory).

Note: Do not load individual project files from Visual Studio. You must first load the top-level solution file (**rtsdk.sln** in the specified ***buildDir***). After loading the full solution from **rtsdk.sln**, you can begin building individual projects.

Once CMake generation is complete, to build using CMake command line instead of Visual Studio, use this command line:

```
cmake --build . --config Debug_MDd
```

In the command above, options for **`config`** are **`Debug_MDd`** or **`Release_MD`**.

4.2 Building on Linux

LSEG uses the default GNU compiler provided by CMake and included in the Linux distribution (which builds in 64-bit; to build in 32-bit, refer to the CMake command options in Section 4.4). For supported OS and compilers, refer to the Compatibility Matrix.

▶ To run CMake in a Linux environment:

1. Obtain RTSDK. For details, refer to Section 3.
2. If obtaining RTSDK by downloading RRG packages, extract the contents of the RTSDK packages as needed. Refer to Section 3.
3. Note the name of the top-level extracted directory (i.e., on Linux, the name might be something like **RTSDK-2.2.1.L1.linux.rrg** or, if this is a GitHub clone, the name might be **Real-Time-SDK**).

   The name of this extracted directory is referred to as ***`sourceDir`*** for the remainder of this procedure.

4. At a command prompt (e.g., in a terminal window), issue the command from the directory immediately above ***sourceDir***:

```
cmake -HsourceDir -BbuildDir [-Doption ...]
```

Note: By default, CMake builds the RTSDK using the optimized build option. For the debug version, instead issue the command: **`cmake -H`*`sourceDir`*` -B`*`buildDir`*` -DCMAKE_BUILD_TYPE=Debug`**. In addition, on Linux platforms only, to build the optimized debug version, use this command: **`cmake -H`***`sourceDir`*` -B`***`buildDir`*` -DCMAKE_BUILD_TYPE=OptimizedDebug`**.

Where:

- ***`sourceDir`*** is the directory in which the top-level CMake entry point (**CMakeLists.txt**) resides. By default, when you build using **Makefile** files, output is sent to directory specified in ***`sourceDir`***.
- ***`buildDir`*** is the CMake binary directory (for the CMake build tree). This directory is created if it does not exist.
- ***`option`*** is a command line option and its associated value (e.g., **`-DBUILD_EMA_UNIT_TESTS=OFF`**). You can control aspects of how CMake builds the RTSDK by using command line options (for further details on the use of options, refer to Section 4.4).

The **`cmake`** command builds all needed **Makefile** files (and related dependencies) in the CMake build directory and may be built using **`gmake`**/**`make`**. Compiled output is located in its associated source directories (i.e., example executables are in the **Executables** directory and libraries (e.g., **libema.lib**, **librssl.lib**) in the **Libs** directory).

4.3 Rebuilding Library Packages (for Use with Developer Portal Downloads)

The RTSDK package that you obtain outside of GitHub (i.e., the Developer Portal) contains prebuilt libraries. However, you might run into use cases that require you to rebuild libraries and/or your RTSDK API package. In normal use cases, where you simply need to build the package, refer to Section 4.1 (for building on Windows) and Section (for building on Linux).

You can rebuild the RTSDK API libraries in the following ways:

- If you need to rebuild the Enterprise Transport or Message API libraries, add the following option to the command line when building. This option also rebuilds the external packages from the tarballs included in the download cache (**external/dlcache**):

```
-DRTSDK_OPT_BUILD_ETA_EMA_LIBRARIES:BOOL=ON
```

- If you need to rebuild everything (including external packages), ensure you have access to the Internet (in case a package needs to be downloaded during the build), and add the following option to the command line when building. This option does not use tarballs included in the download cache (**external/dlcache**) for building the external packages.

```
-DRTSDK_OPT_REBUILD_ALL:BOOL=ON
```

For detailed information on the options included in this section, refer to Section 4.4.

4.4 CMake Build Configuration Options

When running the CMake command, you can use any of the following options:

Tip: If you want to only build the Enterprise Transport API library, turn off the following options: **BUILD_ETA_APPLICATIONS**, **BUILD_EMA_LIBRARY**, and **BUILD_EMA_EXAMPLES**

OPTION	DESCRIPTION	DEFAULT SETTING
BUILD_RTSDK-BINARYPACK	Downloads needed libraries (as a tarball) from GitHub and builds the **RTSDK-BinaryPack**. To use this option, you must have Internet access (with any proxies specified). If you downloaded your package from the Developer Community Portal, this option skips the tarball download and simply builds the **RTSDK-BinaryPack**.	On
BUILD_EMA_DOXYGEN	Builds the Enterprise Message API reference documentation using Doxygen.	Off
BUILD_EMA_EXAMPLES	Builds all programs in **Cpp-C/Ema/Examples**. Turning this option off also turns off **BUILD_EMA_PERFTOOLS**, **BUILD_EMA_TRAINING**, and **BUILD_UNIT_TESTS**.	On
BUILD_EMA_LIBRARY	Builds with the Enterprise Message API library (**libema**)	On
BUILD_EMA_PERFTOOLS	Builds all programs in **Cpp-C/Ema/Examples/Perftools**	On
BUILD_EMA_TRAINING	Builds all programs in **Cpp-C/Ema/Examples/Training**	On
BUILD_EMA_UNIT_TESTS	Builds all unit tests for the Enterprise Message API (located in **Cpp-C/Ema/Examples/Test/UnitTest**).	On
BUILD_ETA_APPLICATIONS	The top-level control option for all Enterprise Transport API Applications. Turning this option off also turns off **BUILD_ETA_EXAMPLES**, **BUILD_ETA_PERFTOOLS**, and **BUILD_ETA_TRAINING**.	On
BUILD_ETA_DOXYGEN	Builds Enterprise Transport API reference documentation using Doxygen.	Off
BUILD_ETA_EXAMPLES	Builds all programs in **Cpp-C/Eta/Applications/Examples**	On
BUILD_ETA_PERFTOOLS	Builds all programs in **Cpp-C/Eta/Applications/Perftools**	On
BUILD_ETA_TRAINING	Builds all programs in **Cpp-C/Eta/Applications/Training**	On
BUILD_ETA_UNIT_TESTS	Builds all unit tests for Enterprise Transport API (located in **Cpp-C/Eta/TestTools/UnitTests**).	On

Table 1: CMake Command Options

OPTION	DESCRIPTION	DEFAULT SETTING
BUILD_UNIT_TESTS	Builds all unit test programs for both the Enterprise Message API (located in **Cpp-C/Ema/Examples/Test/UnitTest**) and Enterprise Transport API (located in **Cpp-C/Eta/TestTools/UnitTests**). Turning this option off also turns off **`BUILD_EMA_UNIT_TESTS`** and **`BUILD_ETA_UNIT_TESTS`**.	On
BUILD_32_BIT_ETA	Forces a 32-bit build. This option builds only the Enterprise Transport API and its examples that do not require the Binary Pack (thus VA examples such as VACons, VAProv, VANIProv, and WatchlistCons are not built). Also turns off the Enterprise Message API and associated examples. **Note:** This is used only for forcing 32-bit Linux builds. **Tip:** To force a 32-bit build in Windows, leave out the Win64 specification in the generator statement.	Off
RTSDK_OPT_BUILD_WITH_PREBUILT_ETA_EMA_LIBRARIES	Available only if you downloaded the RTSDK from the Developer Community Portal. This option sets CMake to build the RTSDK package using prebuilt Enterprise Transport API and Enterprise Message API libraries. This option does not rebuild the libraries themselves.	ON
RTSDK_OPT_BUILD_ETA_EMA_LIBRARIES	Available only if you downloaded the RTSDK from the Developer Community Portal. This option sets CMake to rebuild the Enterprise Transport and Message API libraries, the examples, and the applications, and then rebuild the RTSDK package. To build external project libraries, CMake uses the tarballs from the local download cache (**dlcache**) in the RTSDK distribution.	OFF
RTSDK_OPT_REBUILD_ALL	Available only if you downloaded the RTSDK from the Developer Community Portal. This option sets CMake to rebuild the entire RTSDK distribution. To build external project libraries, CMake downloads the tarballs from the Internet. To use this option, you must have Internet access (with any proxies specified).	OFF

Table 1: CMake Command Options

4.5 Customizing the CMake Configuration

To customize your CMake build, you must configure the **CMakeCache.txt** file in the build directory (***buildDir***). You can edit this file using either a text editor (i.e., **`vi`**) or the appropriate CMake UI[2]. After configuring the **CMakeCache.txt** file, for ease of use, LSEG recommends you use the UI to reconfigure the CMake build. For details on using the CMake UI, refer to CMake's documentation (https://cmake.org/cmake/help/v3.10/).

If you use a text editor to alter the cache. you can update your CMake build tree simply by running the command:

```
cmake -HsourceDir -BbuildDir
```

4.6 CMake Targets

Running CMake generates targets (conceptually this includes Visual Studio projects when running on Windows) that you can compile individually. CMake lists RTSDK-specific targets in **`stdout`**.[3] You can use CMake build configuration options to control the specific set of RTSDK targets generated by CMake (for details, refer to Section 4.4).

For example, when setting **BUILD_ETA_PERFTOOLS=ON** (this is the default), CMake configures the following targets:

- ConsPerf_shared
- ConsPerf
- NIProvPerf_shared
- NIProvPerf
- ProvPerf_shared
- ProvPerf
- TransportPerf_shared
- TransportPerf

When the RTSDK successfully completes the CMake configuration, any target can be built directly if it is included with the configuration (e.g., **`make ConsPerf`**).

2. On Windows, the UI is accessed through the **cmake-gui.exe** binary. On Linux, you access the UI via the **cmake-gui** and a curses interface via a Linux shell with the **`ccmake`** command.
3. For non-RTSDK targets, refer to CMake's documentation and broader CMake developer community (both accessed from https://cmake.org/documentation).

5 Package Directory Changes

The following table compares the RTSDK package directory structure of versions prior to and starting with 2.2.1.L1. The following changes were introduced into the structure over time:

- RTSDK version 1.2: Starting with this version, CMake support was introduced. This changed the directory structure to include top-level **CMake**, **Cpp-C**, and other directories to support a build using CMake.
- RTSDK version 2.2.1: Starting with this version, the RTSDK RRG package no longer contains the binary pack. Instead, the binary pack is provided as a separate BinaryPack RRG package, which is illustrated in the following table. For more information, see Section 3.4. For how to prepare the binary pack to build RTSDK examples, see Section 3.1.1.

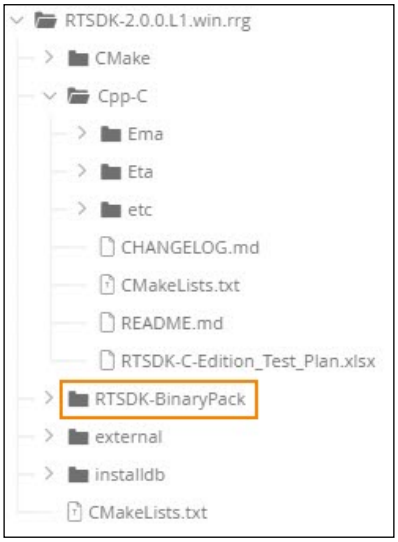
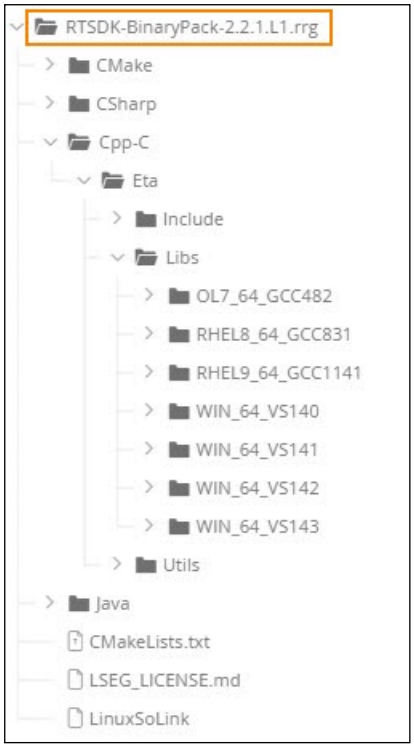
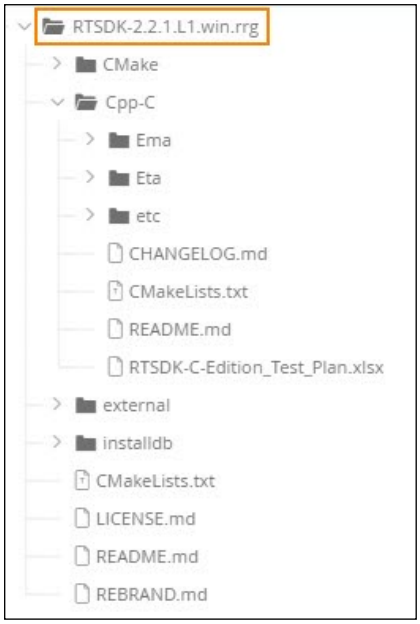
STRUCTURE STARTING WITH RTSDK 1.2.0	STRUCTURE STARTING WITH RTSDK 2.2.1	
RRG package:	BinaryPack RRG package:	RRG package:
		

Table 2: RTSDK C/C++ Package Structures

The **installdb** directory includes external libraries (including **libcurl**). Examples might need to set **LD_LIBRARY_PATH** for Linux or make sure that the libraries are otherwise accessible for Windows (e.g., include the directory in **%PATH%**).

The following sections provide details of the package directory structure changes for specific versions.

5.1 Starting in Version 1.2

Starting in version 1.2, the following changes were introduced:

- The **CMake** directory contains modules to support the CMake build harness.
- The **RTSDK-BinaryPack** presents libraries (prebuilt from non-open source code) as targets for the rest of the RTSDK to use as linkable target objects. For details on accessing the binary pack, refer to the topic called Obtaining RTSDK.
- Previous libraries **librsslRDM**, **librsslReactor**, and **librsslVAUtil** are combined to a single library **librsslVA**.
- A new library **librsslRelMcast** is added (in **RTSDK-BinaryPack/Cpp-C/Eta/Libs**) to account for the shared reliable multicast library. **librsslRelMcast** is dynamically loaded by **librssl** whenever Reliable Multicast transport is selected.
- DACS and ANSI libraries have been moved to directory **RTSDK-BinaryPack/Cpp-C/Eta/Utils**.

5.2 Starting in Version 1.3.1

Starting in version 1.3.1, ANSI is open-sourced:

- The ANSI library is located in **Cpp-C/Eta/Libs**.
- ANSI headers are located in **Cpp-C/Eta/Include/ansi**.

5.3 Starting in Version 1.5.0

RTSDK supports a WebSocket Transport and introduces support for either a RWF or JSON payload. Conversion from RWF to JSON (and from JSON to RWF) is built into **librssl** and also available as a separate shared library.

5.4 Starting in Version 2.2.1

Starting from version 2.2.1.L1, the RTSDK source code with sample applications (examples) and the closed-source binary pack are provided in separate packages:

- RTSDK RRG package
- RTSDK BinaryPack RRG package

The BinaryPack RRG package is not platform specific and has content for all flavors and platforms of RTSDK APIs.

Also, with version 2.2.1, you can build an OptimizedDebug version on Linux platforms. For more information, see Section .

Document ID: RTSC232L1IP.250
Date of issue: December 2025