

RTSDK C/C++ 2.0.6.L1

INSTALLATION GUIDE

1 Overview

RTSDK packages are specific to the product language (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 2.0.1.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

- 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.
- Refinitiv 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 and build the RTSDK binary pack. Thus, you must have an Internet connection for CMake to successfully download the binaries in this manner. Alternatively, you can manually clone the binaries (if running CMake on a machine without Internet access). For details on manually cloning the binaries, refer to Section 3.

3 Obtaining the Package

You have the following options in obtaining the RTSDK:

- You can download the package from the Developer Community Portal at the following URL:
<https://developers.refinitiv.com/en/api-catalog/refinitiv-real-time-opnsrc/rt-sdk-cc/downloads>

Note: RTSDK packages downloaded from the Developer Community Portal already contain the RTSDK binary pack and prebuilt libraries for supported compilers and platforms.

- You can clone the RTSDK from the GitHub repository (at <https://github.com/Refinitiv/Real-Time-SDK>) by using the following command:

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

Note: An RTSDK clone built using CMake will download the RTSDK binary pack on behalf of the user.



Tip: You can also download the source and binary pack from GitHub via the browser:

- Browse to the URL <https://github.com/Refinitiv/Real-Time-SDK/releases>
- Each release will have the following options listed beneath it's release name:
- To download a compressed package, click **zip** or **tar.gz**.



With RTSDK version 1.4 and higher, the binary pack repository on GitHub is no longer updated. Refer to the preceding tip to obtain the binary pack. To build versions prior to 1.4, you can manually clone the binary pack using the following command (clone the binary pack into the **Elektron-SDK-BinaryPack** directory).

```
git clone https://github.com/Refinitiv/Elektron-SDK-BinaryPack.git
```

Note: You need to obtain the binary pack if you run CMake on a machine without access to GitHub via the Internet.

4 Package Directory Changes

The following tables illustrates the RTSDK package directory structure of Version 1.1.3 as compared against the latest directory structure (CMake directory changes were introduced in Version 1.2).

The **installldb** 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%**).

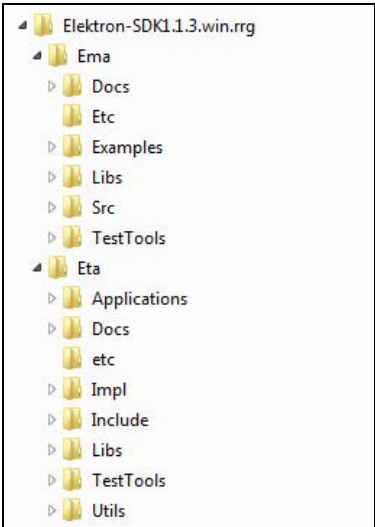
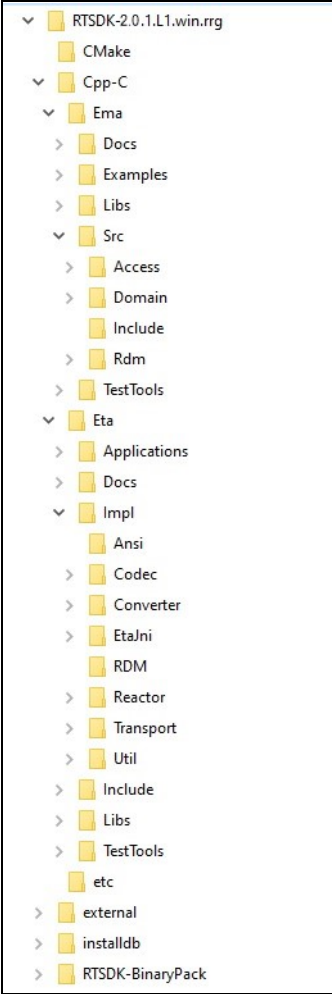
RTSDK C/C++ VERSION 1.1.3 PACKAGE	RTSDK C/C++ 2.0.6.L1 PACKAGE
<p>Prior to the version1.2 release, the RTSDK C/C++ package used the following high-level structure (included here for the sake of comparison):</p>  <pre>Elektron-SDK1.1.3.win.rtg ├── Ema │ ├── Docs │ ├── Etc │ ├── Examples │ ├── Libs │ ├── Src │ └── TestTools ├── Eta │ ├── Applications │ ├── Docs │ ├── etc │ ├── Impl │ ├── Include │ ├── Libs │ ├── TestTools │ └── Utils</pre>	<p>The following diagram illustrates the top-level directory structure for the RTSDK C/C++ 2.0.6.L1 release:</p>  <pre>RTSDK-2.0.1.L1.win.rtg ├── CMake ├── Cpp-C ├── Ema │ ├── Docs │ ├── Examples │ ├── Libs │ └── Src │ ├── Access │ ├── Domain │ ├── Include │ ├── Rdm │ └── TestTools ├── Eta │ ├── Applications │ ├── Docs │ └── Impl │ ├── Ansi │ ├── Codec │ ├── Converter │ ├── EtaJni │ ├── RDM │ ├── Reactor │ ├── Transport │ ├── Util │ ├── Include │ ├── Libs │ ├── TestTools │ └── etc ├── external ├── installldb └── RTSDK-BinaryPack</pre>

Table 1: RTSDK C/C++ Package Structures

4.1 Starting in Version 1.2

- 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 the Package.
- 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**.

4.2 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**.

4.3 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 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** files¹ (on Windows), or **Makefile** files (on Linux) in a build directory that you specify. After this process, you can then 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, Refinitiv supports only the use of CMake version 3.14 or greater. You can download CMake from <https://cmake.org/download/>.

1. CMake refers to such files as 'targets'

5.1 Building with CMake on Windows

► To run CMake in a Windows environment:

1. Obtain the RTSDK (for details, refer to Section 3).
2. Extract the contents of the RTSDK package as needed.
3. Note the name of the top-level extracted directory (i.e., on Windows, the name might be something like **RTSDK1.3.0.L1.win.rrg** or if this is a GitHub clone, the name might be **RTSDK**).

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.
- **VisualStudioVersion** is the Visual Studio version (e.g., **Visual Studio 11 2012 Win64**). Valid values for **VisualStudioVersion** are:
 - "Visual Studio 16 2019" -A x64
 - "Visual Studio 15 2017 Win64"
 - "Visual Studio 14 2015 Win64"
 - "Visual Studio 12 2013 Win64"
 - "Visual Studio 11 2012 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 5.4).

The **cmake** command builds all needed **Solution** and **vcxproj** files (and other related files) in the CMake build tree. Compiled output (after running make or from visual studio make) is located in its associated 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.

5.2 Building with CMake on Linux

Refinitiv 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 5.4). For supported OS and compilers, refer to the Compatibility Matrix.

Note: For Linux builds with RedHat-based distributions (RHEL, CentOS, Oracle Linux), the CMake scripts require **lsb_release** software. On Red Hat Enterprise Linux and CentOS, when logged in as root, you can install **lsb_release** using the following command: `yum install redhat-lsb-core`

► To run CMake in a Linux environment:

1. Obtain the RTSDK (for details, refer to Section 3).
2. Extract the contents of the RTSDK package as needed.
3. Note the name of the top-level extracted directory (i.e., on Linux, the name might be something like **RTSDK1.3.0.L1.linux.rrg** or if this a GitHub clone, the name might be **RTSDK**).

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 -HsourceDir -BbuildDir -DCMAKE_BUILD_TYPE=Debug`

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).
- **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 5.4).

The **cmake** command builds all needed **Makefile** files (and related dependencies) in the CMake build tree in their associated directories (i.e., example executables are in the **Executables** directory and libraries (e.g., **libema.lib**, **librssl.lib**) in the **Libs** directory). You open these files and build all libraries and examples in the same fashion as you did with prior RTSDKs.

5.3 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 5.1 (for building on Windows) and Section 5.2 (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 5.4.

5.4 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 2: 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	<p>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.</p> <p>Note: This is used only for forcing 32-bit Linux builds.</p> <p> Tip: To force a 32-bit build in Windows, leave out the Win64 specification in the generator statement.</p>	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 2: CMake Command Options

5.5 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 UI². After configuring the **CMakeCache.txt** file, for ease of use, Refinitiv 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
```

5.6 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 5.4).

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>).

© 2018 - 2022 Refinitiv. All rights reserved.
Republication or redistribution of Refinitiv content, including by framing or similar means, is prohibited without the prior written consent of Refinitiv. 'Refinitiv' and the Refinitiv logo are registered trademarks and trademarks of Refinitiv and its affiliated companies.

RTSDK C/C++ v2.0.6.L1 Installation Guide
Document Version: 2.0.6
RTSC206IP.220

