We are now Refinitiv, formerly the Financial and Risk business of Thomson Reuters. We've set a bold course for the future – both ours and yours – and are introducing our new brand to the world.

As our brand migration will be gradual, you will see traces of our past through documentation, videos, and digital platforms.

Thank you for joining us on our brand journey.

REFINITIV™

ESDK C/C++ 1.3.x

INSTALLATION GUIDE

1 Overview

ESDK packages are specific to the product language (C/C++ or Java) and include both the ETA and EMA products. This *Installation Guide* describes the procedures to install and build ESDK C / C++. These instructions apply to ESDK versions 1.2 and higher. Because the steps to install are specific to ESDK, they apply to both ETA and EMA.

Starting in Version 1.2, the ESDK supports open sourcing and uses more standards-based, freely-available open source tools to provide additional flexibility and benefit.

In versions prior to 1.2, the ESDK APIs were built without a CMake harness (i.e., developers used the static build files with other utilities such as Visual Studio or Linux make to build the APIs). With the open-source version 1.2 ESDK release, developers use CMake to dynamically generate the build files.

Note: Version 1.2 (and later) ESDK applications are more memory-use intensive when initializing the ETAC library and when loading the dictionary.

2 Requirements and Limitations

The ESDK C/C++ package uses Google Test in its unit tests. While the ESDK automatically downloads Google Test whenever you run its unit tests, Google Test requires Python. So if you want to run the ESDK unit tests, you must ensure your also have Python on your machine.

The ESDK C/C++ version 1.3 package requires CMake.

Thomson Reuters does not support 32-bit builds in EMA.

Starting with version ESDK 1.3, you must install openSSL if you intend to use encrypted connections.

If you downloaded the ESDK package from Github and run CMake, CMake automatically attempts to download needed libraries from GitHub and build the Elektron-SDK binary pack. You must have Internet access 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.

THOMSON REUTERS

3 Obtaining the Package

You have the following options in obtaining the ESDK:

- You can download the package from the Developer Community Portal at the following URL:
  https://developers.refinitiv.com/elektron/elektron-sdk-cc/downloads

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

- You can clone the ESDK from the GitHub repository (at https://github.com/Refinitiv/Elektron-SDK) by using the following command:

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

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

Tip:	You can also download the package from GitHub via the browser: • Browse to the URL https://github.com/Refinitiv/Elektron-SDK/releases • Each release will have the following options listed beneath it's release name: zip tar.gz • To download a compressed package, click **zip** or **tar.gzip**.

If you need to manually clone the binary pack, do so according to the preceding tip or by using the following command (clone the binary pack into the **Elektron-SDK** directory).

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

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

4 Package Directory Changes

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

Elektron SDK C/C++ Version 1.1.3 Package	ESDK C/C++ 1.3.x Package
Prior to the version1.2 release, the ESDK C/C++ package used the following high-level structure (included here for the sake of comparison): ▲ 📁 Elektron-SDK1.1.3.win.rrg 　▲ 📁 Ema 　　▷ 📁 Docs 　　　📁 Etc 　　▷ 📁 Examples 　　▷ 📁 Libs 　　▷ 📁 Src 　　▷ 📁 TestTools 　▲ 📁 Eta 　　▷ 📁 Applications 　　▷ 📁 Docs 　　　📁 etc 　　▷ 📁 Impl 　　▷ 📁 Include 　　▷ 📁 Libs 　　▷ 📁 TestTools 　　▷ 📁 Utils	The following diagram illustrates the top-level directory structure for the ESDK C/C++ 1.3.x release: ▲ 📁 Elektron-SDK1.3.0.L1.win.rrg 　📁 CMake 　▲ 📁 Cpp-C 　　▲ 📁 Ema 　　　▷ 📁 Docs 　　　▷ 📁 Examples 　　　▷ 📁 Executables 　　　▷ 📁 Libs 　　　▷ 📁 Src 　　　▷ 📁 TestTools 　　▲ 📁 Eta 　　　▷ 📁 Applications 　　　▷ 📁 Docs 　　　▷ 📁 Executables 　　　▷ 📁 Impl 　　　▷ 📁 Include 　　　▷ 📁 Libs 　　　▷ 📁 TestTools 　　　📁 etc 　▲ 📁 Elektron-SDK-BinaryPack 　　📁 CMake 　　▲ 📁 Cpp-C 　　　▲ 📁 Eta 　　　　▷ 📁 Include 　　　　▷ 📁 Libs 　　　　▷ 📁 Utils 　▲ 📁 external 　　📁 dlcache 　▲ 📁 installdb 　　▷ 📁 WIN_64_VS110 　　▷ 📁 WIN_64_VS120 　　▷ 📁 WIN_64_VS140 　　▷ 📁 WIN_64_VS150

Table 1: ESDK C/C++ Package Structures

In Versions 1.2 and later:

- The **CMake** directory contains modules to support the CMake build harness

- The **Elektron-SDK-BinaryPack** presents libraries (prebuilt from non-open source code) as targets for the rest of the ESDK 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 **Elektron-SDK-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 **Elektron-SDK-BinaryPack/Cpp-C/Eta/Utils**.

5 CMake

Prior versions of the ESDK provided the static build files **Solution** and **vcxproj** for Windows, and **Makefile** for Linux. However, ESDK Version 1.2 has changed to instead include CMake configuration files (**CMakeLists.txt**) in strategic directories. You must now 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 ESDK product.

The ESDK 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 then compile your ESDK in the same way as previous ESDK versions (i.e., by running Make on Linux or by using Visual Studio on Windows). For details on configuring the ESDK with CMake, refer to Section 5.5.

For both Windows and Linux, Thomson Reuters supports the use of CMake version 3.10 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 ESDK (for details, refer to Section 3).

2. Extract the contents of the ESDK package as needed.

3. Note the name of the top-level extracted directory (i.e., on Windows, the name might be something like **Elektron-SDK1.3.0.L1.win.rrg** or if this a GitHub clone, the name might be **Elektron-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.

- **VisualStudioVersion** is the Visual Studio version (e.g., **Visual Studio 11 2012 Win64**).[2] For example: **"Visual Studio 15 2017 Win64"**. For a list of generator types, you can run **cmake --help**.

  Note: If you do not explicitly specify **Win64**, by default cmake builds the 32-bit version.

- **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 ESDK 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 (**esdk.sln** in the specified **buildDir**). After loading the full solution from **esdk.sln**, you can begin building individual projects.

---

2. For details on Visual Studio generators and a list of available generators, refer to:
https://cmake.org/cmake/help/v3.10/manual/cmake-generators.7.html?highlight=visual%20studio#visual-studio-generators

5.2   Building with CMake on Linux

Thomson Reuters 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 Compatability Matrix.

▶ **To run CMake in a Linux environment:**

1. Obtain the ESDK (for details, refer to Section 3).

2. Extract the contents of the ESDK package as needed.

3. Note the name of the top-level extracted directory (i.e., on Linux, the name might be something like **Elektron-SDK1.3.0.L1.linux.rrg** or if this a GitHub clone, the name might be **Elektron-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 ESDK 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 ESDK 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 ESDKs.

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

The ESDK 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 ESDK 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 ESDK API libraries in the following ways:

- If you need to rebuild the ETA and/or EMA 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**):

```
-DESDK_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.

```
-DESDK_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 ETA library, turn off the following options: `BUILD_ETA_APPLICATIONS`, `BUILD_EMA_LIBRARY`, and `BUILD_EMA_EXAMPLES`

Option	Description	Default Setting
BUILD_ELEKTRON-SDK-BINARYPACK	Downloads needed libraries (as a tarball) from Github and builds the Elektron-SDK-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 Elektron-SDK-BinaryPack.	On
BUILD_EMA_DOXYGEN	Builds EMA 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 Ema library (**libema**)	On

Table 2: CMake Command Options

Option	Description	Default Setting
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 EMA (located in **Cpp-C/Ema/Examples/Test/UnitTest**).	On
BUILD_ETA_APPLICATIONS	The top-level control option for all ETA Applications. Turning this option off also turns off `BUILD_ETA_EXAMPLES`, `BUILD_ETA_PERFTOOLS`, and `BUILD_ETA_TRAINING`.	On
BUILD_ETA_DOXYGEN	Builds ETA 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 ETA (located in **Cpp-C/Eta/TestTools/UnitTests**).	On
BUILD_UNIT_TESTS	Builds all unit test programs for both EMA (located in **Cpp-C/Ema/Examples/Test/UnitTest**) and ETA (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 ETA and ETA examples that do not require the Binary Pack (thus VA examples such as VACons, VAProv, VANIProv, and WatchlistCons are not built). Also turns off EMA 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
ESDK_OPT_BUILD_WITH_PREBUILT_ETA_EMA_LIBRARIES	Available only if you downloaded the ESDK from the [Developer Community Portal](). This option sets CMake to build the ESDK package using prebuilt ETA and EMA libraries. This option does not rebuild the libraries themselves.	ON

Table 2: CMake Command Options

Option	Description	Default Setting
ESDK_OPT_BUILD_ETA_EMA_LIBRARIES	Available only if you downloaded the ESDK from the [Developer Community Portal](). This option sets CMake to rebuild the ETA and EMA libraries, the examples, and the applications, and then rebuild the ESDK package. To build external project libraries, CMake uses the tarballs from the local download cache (**dlcache**) in the ESDK distribution.	OFF
ESDK_OPT_REBUILD_ALL	Available only if you downloaded the ESDK from the [Developer Community Portal](). This option sets CMake to rebuild the entire ESDK 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 UI[3]. After configuring the **CMakeCache.txt** file, for ease of use, Thomson Reuters 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
```

---

3. 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.

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 ESDK-specific targets in `stdout`.[4] You can use CMake build configuration options to control the specific set of ESDK 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 ESDK successfully completes the CMake configuration, any target can be built directly if it is included with the configuration (e.g., `make ConsPerf`).

---

4. For non-ESDK targets, refer to CMake's documentation and broader CMake developer community (both accessed from https://cmake.org/documentation).