Thomson Reuters Enterprise Platform

ESDK C/C++ 1.2

MIGRATION GUIDE

1 Overview

ESDK packages are specific to the product language (C/C++ or Java) and include both the ETA and EMA products. This *ESDK C/C++ 1.2 Migration Guide* describes migrating from any prior ESDK C / C++ version to Version 1.2 (or later). Because migration steps are specific to the ESDK package, migration steps are identical for both ETA and EMA.

With this release, 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 Obtaining the Package

In addition to obtaining the C/C++ ESDK via the GSG, you have the following alternative options in obtaining the SDK:

- You can download it from the Developer Community Portal at the following URL: https://developers.thomsonreuters.com/elektron/elektron-sdk-cc/downloads
- You can clone it from the GitHub repository at: https://github.com/thomsonreuters/Elektron-SDK.

3 Package Directory Changes

The following tables illustrates the ESDK package directory structure of Version 1.1.3 as compared against the new directory structure introduced in Version 1.2.

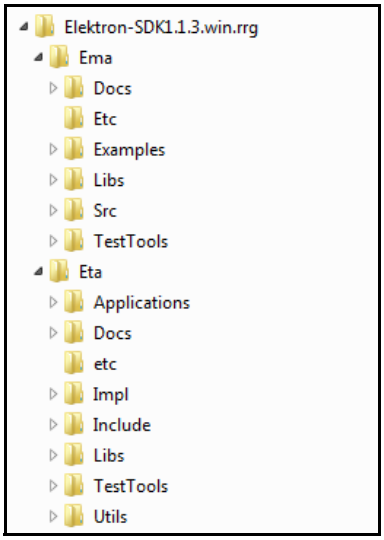
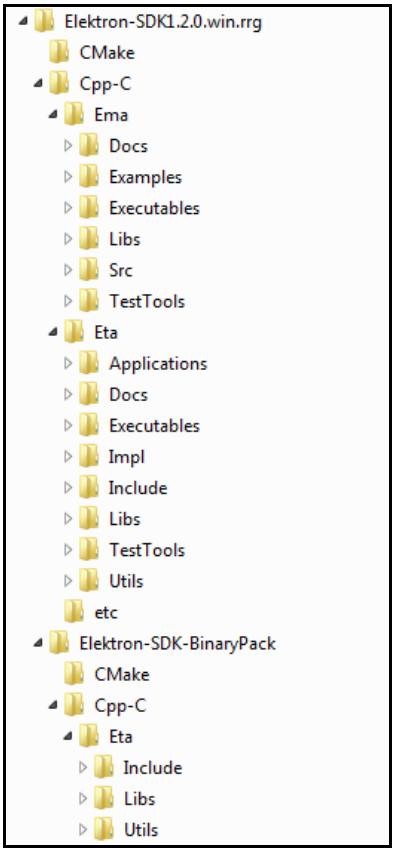
Elektron SDK C/C++ Version 1.1.3 Package	Elektron SDK C/C++ Version 1.2 Package
The ESDK C/C++ package prior to the version1.2 release, used the following high-level structure (included here for the sake of comparison): 	Starting with Version 1.2, the ESDK C/C++ package uses a new directory structure that differs significantly from previous versions. The following diagram illustrates the new top-level directory structure: 

Table 1: ESDK C/C++ Package Structures

In Version 1.2:

- 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.
- 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.

4 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) or you can further configure your CMake output by customizing the CMake cache file named **CMakeCache.txt**. For details on configuring CMake output, refer to Section 4.4.

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'

4.1 Building with CMake on Windows

To run CMake in a Windows environment:

1. Obtain the ESDK package (for details, refer to Section 2).
2. Extract the contents of the ESDK package.
3. Note the name of the top-level extracted directory (i.e., on Windows, the name might be something like **Elektron-SDK1.2.0.win.rrg**).

   You will use this name in Step 5 as the ***sourceDir***.
4. Open a command window: on the **Windows** menu, in **Search programs and files**, type **cmd** and press ENTER.
5. Issue the command:

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

   Where:

   - ***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 binary directory (for the CMake build tree).
   - ***VisualStudioVersion*** is the Visual Studio generator (e.g., **Visual Studio 11 2012**).[2]
   - ***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 4.3).

   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). You open these files and build all libraries and examples in the same fashion as you did with prior ESDKs.

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

4.2 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 4.3). For supported OS and compilers, refer to the Compatability Matrix.

To run CMake in a Linux environment:

1. Obtain the ESDK package (for details, refer to Section 2).
2. Extract the contents of the ESDK package.
3. Note the name of the top-level extracted directory (i.e., on Linux, the name might be something like **Elektron-SDK1.2.0.linux.rrg**).

   You will use this name in the following steps as the ***sourceDir***.
4. Run the **LinuxSoLink** script: at a command prompt (e.g., in a terminal window) from the ***sourceDir*** directory, issue the command:

```
./LinuxSoLink
```

5. 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 4.3).

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.

4.3 CMake Build Configuration Options

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

Note: By default, all options are active except for **BUILD_WITH_PREBUILT_ETA_EMA_LIBRARIES**. Turning off certain options have a cascading affect on other options (for example, setting **DBUILD_UNIT_TESTS=OFF** in the command line also switches off the options **BUILD_EMA_UNIT_TESTS** and **BUILD_ETA_UNIT_TESTS**. To see whether a relationship exists between options, refer to the following option descriptions in Table 2.

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_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
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

Table 2: CMake Command Options

Option	Description	Default Setting
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
BUILD_WITH_PREBUILT_ETA_EMA_LIBRARIES	Builds applications with the distributed (prebuilt) EMA and ETA libraries (as packaged with the ESDK obtained from the Developer Community Portal or GSG). Do not use this option if you obtained your ESDK via GitHub. Turning this option on turns off the `BUILD_EMA_LIBRARY` option.	Off

Table 2: CMake Command Options

4.4 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, and on Linux you access this UI via the `ccmake` command.

4.5 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 4.3).

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

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