EBU MXF SDK:
Phase #1, EBUCore metadata functionality

This document describes the functionality and structure of the EBUCore processing, serialization and extraction part of the EBU MXF SDK. It also describes how to use the EBUCore functions in derivative projects.

1 Functionality

The EBU MXF SDK offers functionality for processing EBUCore metadata, and in particular, dealing with the inclusion and extraction of EBUCore metadata in Material Exchange Format (MXF) audiovisual essence container files.

1.1 Concepts of EBUCore Processing

This section describes the EBUCore processing functionality of the SDK (illustrated in Section ❶ in Figure 1). The SDK can read and write two representations of EBUCore; the XML variant is read from and written to XML documents that conform to the EBUCore XML schema[1], the MXF variant is read and written to KLV packets, the native encoding of information units in MXF files, that conform to the Class 13 SMPTE metadata element and group dictionary register for EBUCore. For both XML and MXF representations, the EBUCore metadata is read (or written to) an in-memory representation (i.e., an instantiated object model) first and then translated to the other representation through a bi-directional mapping that translates peculiarities between both sides.

The source code that defines the object model for both representations has been generated automatically. It should be again generated when modifications are done the source definitions, either to the EBUCore XML schema or the MXF-KLV metadata dictionary. Instructions as to how this is done are provided in Appendix 1.

1.1.1 Writing EBUCore metadata into MXF files

Concerning the serialization of KLV EBUCore packets into MXF files, the SDK can function in one of two modes; as part of the wrapping process of raw essence into a new MXF file, and as part of the extension process of an existing MXF file.

Firstly, and illustrated in Section ❷ in Figure 1, the SDK can operate in a mode where a new MXF file is constructed from a number of essence tracks (delivered in separate files) and the EBUCore KLV packets are mixed in with the newly constructed metadata (incl., track structure and essence characteristics) of the MXF file. In this case, the metadata is written as part of the header partition, which is marked closed and complete. There is no need for metadata to be written to any of the body or footer partitions.

The **raw2bmx** example program in ebu-bmx/apps/raw2bmx demonstrates the use of this first mode of operation. Along with the parsing of raw essence and inclusion of its structural metadata in a

[1] The EBU SDK employs the EBU_CORE_20120917.xsd version of the EBUCore XML schema.

newly written MXF file, the EBUCore metadata is processed from an XML document, hooked onto the MXF file metadata model, and then written to the file along with the other KLV packets in the MXF container.

In the second mode of operation, the SDK writes EBUCore metadata into an existing MXF file, the path depicted in Section ❸ in Figure 1. This mode requires more complex application logic, as the existing file must 1) be modified as efficiently as possible, and 2) the existing metadata must be modified in such a way as to remain fully compliant with the MXF file format specification.
MXF files may carry multiple instances of the file's metadata (each new file partition can contain an updated set of metadata). This way, streaming and growing file scenarios can be supported in which increasingly accurate metadata is continuously inserted as the file being is extended, resulting in an MXF file that contains the most complete metadata in its footer partition. Partitions marked as open and incomplete can instruct MXF interpreters to ignore early sets of metadata and only consider a final closed and complete metadata set as the definitive MXF file structure description.
Unless explicitly instructed otherwise, the SDK uses this mechanism to append the updated metadata in the footer partition of the MXF file. This involves a rewrite of only the footer partition, which is requires only limited writing operations since footer partitions contain no essence. Most of the header and (bulky) body partitions remain unchanged, except for an update of the small partition header KLV pack to signal an – as of now – open and incomplete metadata set. Note that, when selecting the metadata to extend, the SDK also interprets partition flags to select only the finalized metadata for extension with EBUCore elements.

Considering the complexity of the MXF file format specification, it is not unlikely that certain implementations of MXF interpreters will lack support for selection of metadata beyond the header partition, and will expect this partition to contain only a single complete metadata set. To support these systems, the SDK can be overridden to write the EBUCore metadata to the header partition, at the expense of a byte shift operation across the remainder of the MXF file.

The **ebu2mxf** example program in ebu-bmx/apps/ebu2mxf demonstrates the use of this second mode of operation. An existing MXF file is opened, and the EBUCore metadata is appended to its most appropriated (closed and complete metadata where available) set of metadata. This mode uses an identical EBUCore processing path as in the first operation mode, but only attempts to modify only the file metadata without rewriting its essence.

To use the SDK in this mode, use one of the `EBUCore::EmbedEBUCoreMetadata()` functions that are documented in the Doxygen documentation.

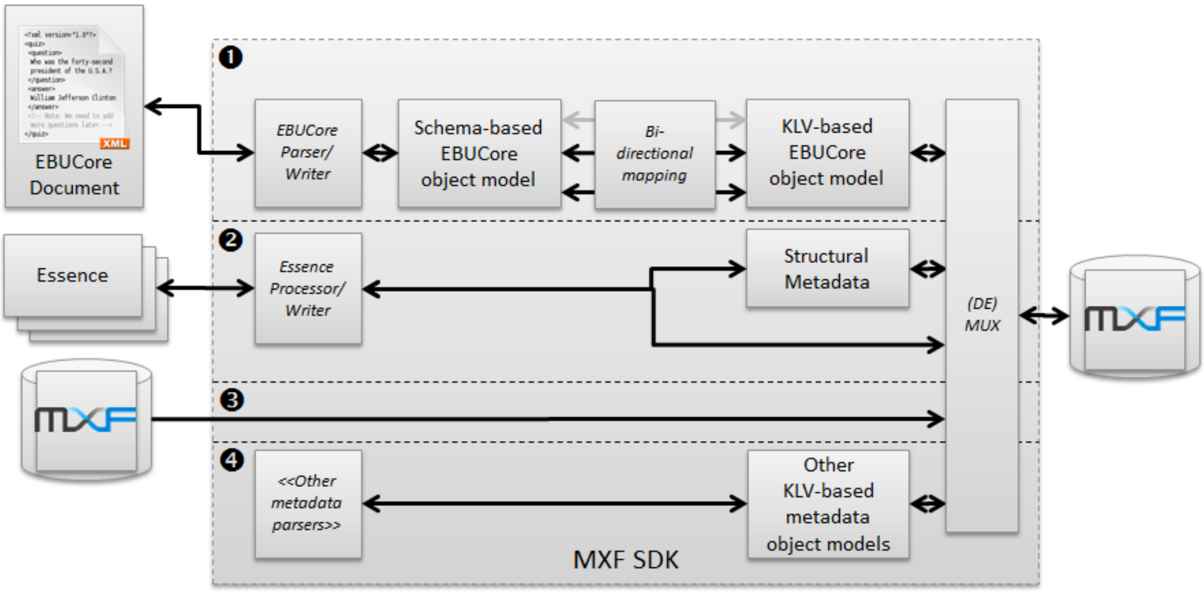

Figure 1: EBU SDK block diagram.

1.1.2 Methods of EBUCore serialization

Apart from the overall operation mode of the SDK (writing a new file, or modifying an existing one) and the location where to write the metadata (forcibly in the header partition, or more efficiently in the footer), there are three ways in which the SDK can write actual EBUCore metadata into an MXF file. EBUCore can be embedded as a full object tree of KLV packets, as a minimal number of KLV packets that refer to the external file as side-car metadata, or the XML representation can be inserted into a single 'dark' KLV packet at the end of the header metadata. All methods are discussed in the following paragraphs.

1.1.2.1 KLV Serialization

EBUCore can be written into the MXF file using a comprehensive tree of KLV packets, formatted according to the MXF specification best practices and the EBUCore dictionary registers; each logical object is contained in a single local set-formatted KLV packet and MXF metadata strong references hold these objects together. Also, each EBUCore KLV object extends the standard MXF *Interchange Object* such that all written metadata objects can, amongst others, participate in the MXF modification tracking mechanism.

Additionally, the EBUCore metadata is inserted into the MXF metadata in such a way that in properly interacts with the timeline model of the MXF file format, as illustrated in Figure 2. Alongside the timeline tracks that describe the essence, a Descriptive Metadata static track is inserted, which contains a reference to the entry point of the EBUCore metadata: an *ebucoreMainFramework* instance. *Part* definitions described in more complex EBUCore documents are also properly modelled on the MXF timeline by using a Descriptive Metadata *Event Track* on which temporal segments with a reference to an *ebucorePartFramework* object instance are assigned for each of the *Part* elements.

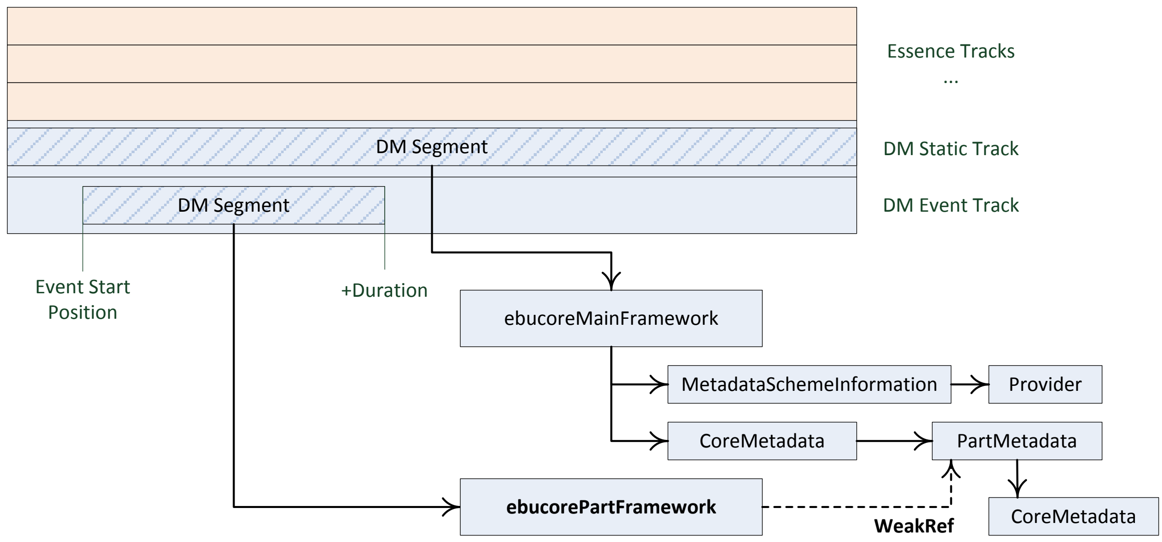

Figure 2: EBUCore metadata and the MXF timeline.

1.1.2.2 Side-car serialization

A second method of serialization writes only a minimal set of KLV packets to the MXF file, and uses a reference to an external file in which the actual metadata is stored. This method can be used in scenarios where metadata updates will occur and the recurring modification of MXF files is not feasibly. The downside of this approach is that metadata file must be transferred and kept together with the MXF file throughout the production and distribution process.

With respect to the KLV packets stored in this case, only the timeline elements (the *DM Static Track*, and its segments) and the *ebucoreMainFramework* are written. Also, no *MetadataSchemeInformation* or *CoreMetadata* objects are present. The *ebucoreMainFramework* only contains a *DocumentLocator* field into which the location of the side-car file is written.

1.1.2.3 'Dark' serialization

The third method of EBUCore serialization involves embedding a single 'dark' KLV packet in which the EBUCore XML metadata document is written as-is. The KLV packet is inserted as the last packet at the end of the regular header metadata and is identified by a specific EBUCore 'dark' metadata key. No further modifications are done to the MXF file metadata.

1.1.3 Extraction of EBUCore metadata from MXF files

The EBU MXF SDK also operates in the reverse direction of the serialization functionality explained above. EBUCore metadata can be extracted from MXF KLV packets and translated to the XML representation, by following the reverse path of Section ❶ in Figure 1. Just as in the serialization direction, the EBUCore metadata is extracted from the most appropriate header metadata (closed and complete where available). The EBUCore metadata is then located by searching through the structural metadata and MXF timeline model, as described in the next section.

When parsing the MXF file, the SDK automatically detects the EBUCore metadata from each of the used serialization methods. It first attempts to read fully KLV-encoded metadata, then tries to locate a side-car metadata file (if applicable) and finally searches for the EBUCore dark metadata packet. The first set of metadata found is employed.

The **mxf2ebu** example program in ebu-bmx/apps/mxf2ebu demonstrates the use of EBUCore metadata extraction. An existing MXF file is opened, and the EBUCore metadata is read from its metadata.

To use the SDK for EBUCore extraction, use one of the `EBUCore::ExtractEBUCoreMetadata()` functions that are documented in the Doxygen documentation.

2 SDK Structure and Dependencies

In this section, a short description of given of the individual components of the SDK, and the limited set of software components it depends on, as shown in Figure 3. Basic MXF manipulation and parsing functionality has been inherited from the BBC's libMXF (the C library) and libMXF++ (a C++ wrapper library of the libMXF functions). libMXF++ (and as such also libMXF) are used by the BMX higher-level library for MXF manipulations for specific Application Specifications (e.g., AS-02, AS-11, …). This BMX library is in turn also used by the EBU SDK code, along with libMXF++. At the moment, as separate code repository is used for libMXF, libMXF++ and BMX+EBU SDK.

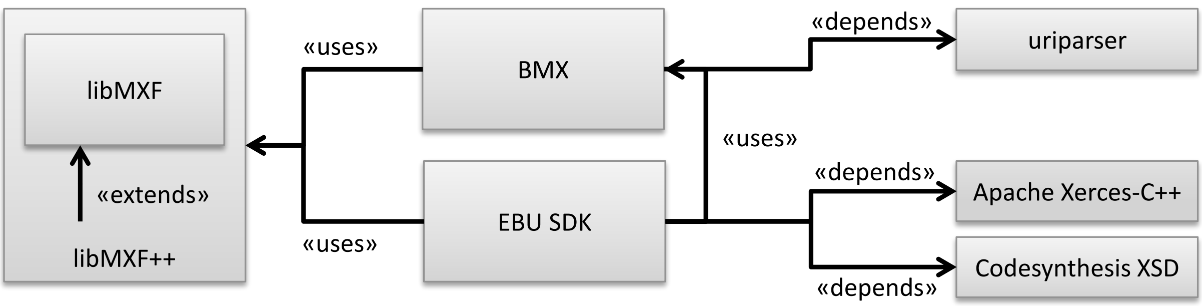

Figure 3: Dependencies in the EBU SDK.

The SDK depends on the following software components.

- uriparser: a library for parsing URIs;
- Apache Xerces-C++: a library for processing XML documents and XML schema validation;
- Codesynthesis XSD: a library for generating XML schema-based parsers and serializers.

For this version of the SDK, programs are use the SDK statically link to the functionality of the SDK, no explicit provisions have been made yet to produce a dynamically linkable library.

2.1 Code Structure

This section describes where in the repository the source code for each of the functional elements of the SDK can be found. For illustration, these elements have been identified in Figure 4.

ebu-bmx\apps	Example applications that use functionality of the EBU SDK (and the bmx libraries). This includes the **mxf2ebu** and **ebu2mxf** tools mentioned earlier.
ebu-bmx\EBUCoreProcessor	Functionality specific to the EBU SDK.
ebu-bmx\EBUCoreProcessor**src\metadata*** ebu-bmx\EBUCoreProcessor**include\metadata***	Contains the EBUCore source files that represent the EBUCore KLV object model. This code is generated by the generate_ebucore_classes tool.

	Also contains code (EBUCoreDMS++.cpp) that deals with registering the EBUCore metadata extensions with a standard MXF metadata model. This implements the *KLV-based EBUCore object model* block ❹ in Figure 4.
ebu-bmx\EBUCoreProcessor**\src\xsd***, ebu-bmx\EBUCoreProcessor**\include\xsd***	Contains the source files that represent the C++ object model of the EBUCore XML schema. This code is generated by the Codesynthesis XSD schema compiler. One file is present for each version of the XSD schema used (i.e., v1.3 and the latest version of 1.4 available at time of writing). This code implements (along with the external XSD and Xerces-C++ libraries) the *EBUCore Parser/Writer* block ❶ and the *Schema-based EBCore object model* block ❷ in Figure 4.
ebu-bmx\EBUCoreProcessor ↵ **\src\EBUCoreMapping.cpp**, ebu-bmx\EBUCoreProcessor ↵ **\src\EBUCoreReverseMapping.cpp**	Bi-directional mapping code for conversion between the XML Schema-based EBUCore object model and the KLV-based object model. These files implement the *Bi-directional mapping* block ❸ in Figure 4.
ebu-bmx\EBUCoreProcessor ↵ **\src\EBUCoreProcessor.cpp**	Contains EBUCore-specific code for handling EBUCore serialization and extraction in MXF files. As such, it implements the EBUCore-specific part of the *(DE)MUX* block ❺ in Figure 4.
ebu-bmx\EBUCoreProcessor ↵ **\src\MXFCustomMetadata.cpp**	Contains generic code for handling custom metadata serialization and extraction in MXF files. As such, it implements the generic part of the *(DE)MUX* block ❺ in Figure 4. This generic part also uses various functions and objects of the bmx and lower-level libMXF and libMXF++ libraries.
ebu-bmx\EBUCoreProcessor ↵ **\src\XercesUtils.cpp**	Utility code for interaction between the Xerces-C++ functions and the functionality of MXFCustomMetadata.cpp, e.g., for the serialization of parsed XML documents into an MXF KLV packet.

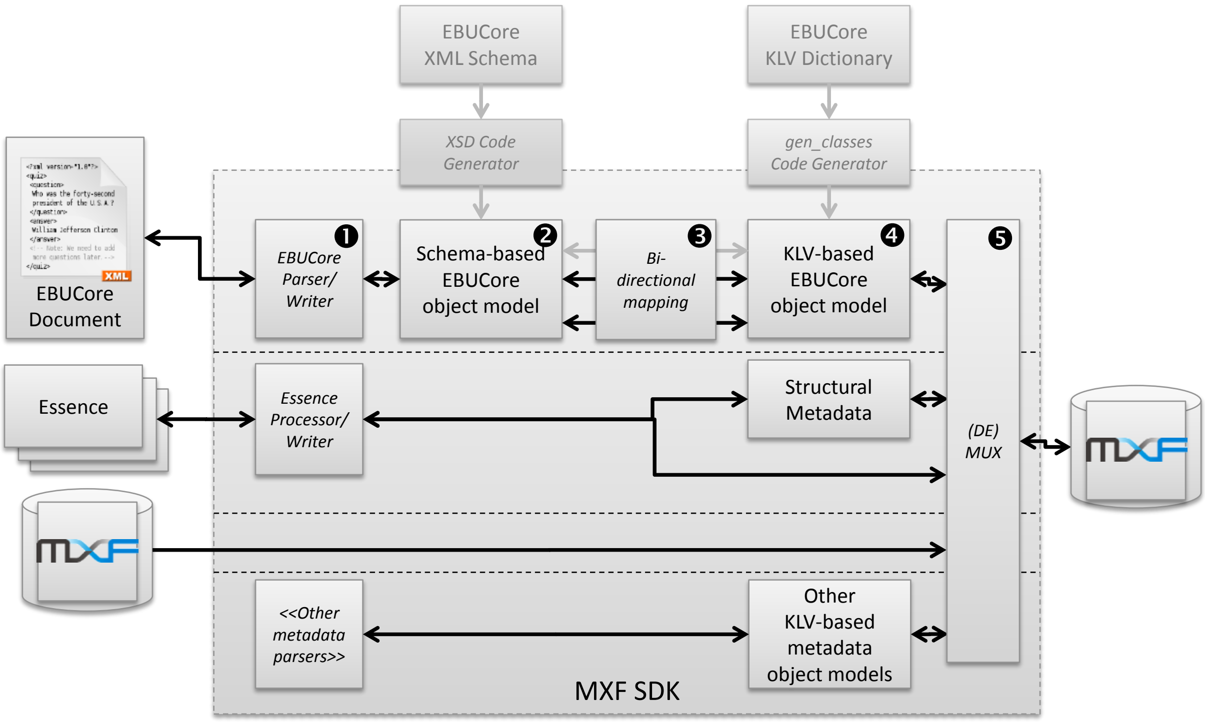

Figure 4: Functional code blocks of the EBU MXF SDK.

3 EBU SDK Source Code

The source code for the SDK consists of three repositories, cloned from the BMX libraries and related code by BBC. It is publicly available at the following locations:

http://sourceforge.net/p/bmxlib/libmxf/	This is the original BBC repository, as no changes have been made here yet. Contains the base MXF functionality written in C, including code for reading KLV units, partitions, metadata, …
https://github.com/Limecraft/ebu-libmxfpp	C++ wrappers for the libmxf library, with a number of extensions for the EBU SDK.
https://github.com/Limecraft/ebu-bmx	Contains the SDK-specific functionality is here, along with the original high-level BMX libraries by BBC.

3.1.1 Building the source code on UNIX-type systems

The SDK inherits the build system from the BBC BMX libraries, and uses a sequence of automatically generated configuration files that automate the build process, incl. tools such as GNU automake and GNU autoconf.

The code depends on a number of software frameworks that must be present before building.

uriparser	Clone the publicly available repository into the directory where the three EBU SDK repositories have been checked out, from the following location: git://uriparser.git.sourceforge.net/gitroot/uriparser/uriparser Make sure the repository is checked out into a directory named “uriparser”. Then use the uriparser build instructions to make and install the library.
Codesynthesis XSD (version 3.3.0)	Depending on the system specifics, this component could be present in the system package/software management repository and can be installed from there. Otherwise binary packages and source archives are available at: http://www.codesynthesis.com. Note: The EBU SDK requires only header files from the XSD component. No binary libraries need to be linked with SDK-produced software.
Apache Xerces-C++ (version 3.1.1)	Depending on the system specifics, this component is likely to be present in the system package/software management repository and can be installed from there. Otherwise binary packages and source archives are available at: http://xerces.apache.org/xerces-c/.

The following commands build the SDK, and install the code it provides to a system-wide accessible location for other software to use.

Apart from the dependencies, this document assumes that all code repositories are checked out into a common parent directory from which the following commands should be executed.

```
cd libmxf
./gen_scm_version.sh
./autogen.sh
./configure
make
make install
```

```
cd libmxfpp
./gen_scm_version.sh
./autogen.sh
./configure
make
make install
```

```
cd ebu-bmx
./gen_scm_version.sh
./autogen.sh
./configure
make
make install
```

3.2 Building the source code on Windows systems

The source code includes Solution and Project files for the Microsoft Visual Studio 2010 development environment. From these project files, available in each repository's /msvc_build/vs10 directory, the project can be built directly, provided that the location of the include directories for the dependencies are updated or added in the Visual Studio projects.

The code depends on a number of software frameworks that must be present before building.

uriparser	Clone the publicly available repository into the directory where the three EBU SDK repositories have been checked out, from the following location: git://uriparser.git.sourceforge.net/gitroot/uriparser/uriparser Make sure the repository is checked out into a directory named "uriparser". Then use the uriparser build instructions to make and install the library, thereby possibly upgrading the solution and project files to a newer Visual Studio format.
Codesynthesis XSD (version 3.3.0)	The binary archive of this component can be found in the ebu-bmx repository: \dependencies\xsd-3.3.0-i686-windows.zip. This archive must be extracted, and references (in include files and library directories) from the EBU SDK must be updated to the proper location. Note: The EBU SDK requires only header files from the XSD component. No binary libraries need to be linked with SDK-produced software.
Apache Xerces-C++ (version 3.1.1)	The binary archive of this component can be found in the

	ebu-bmx repository: \dependencies\ xerces-c-3.1.1-x86-windows-vc-10.0.zip. This archive must be extracted, and references (in include files and library directories) from the EBU SDK must be updated to the proper location.

4 Appendix 1: Generating new EBUCore definition code

Whenever the definition of a metadata representation (the XML Schema or KLV registers) of EBUCore changes due to revisions of the standard parts of the code of the EBU SDK can be regenerated to reflect those changes. This Appendix describes how this is done.

Note that for the generation of some EBUCore program code, a working runtime of the Groovy language is required. Groovy is a high-level programming language, inspired by Java and Ruby, which runs on the Java Runtime Environment. Groovy can be obtained from http://groovy.codehaus.org/.

4.1 Generating new EBUCore XSD schema definition code

The EBUCore XML schema is compiled into C++ code that represents the model of the schema such that it can be manipulated using program code. Whenever the XML schema changes, the code must be regenerated using the Codesynthesis XSD schema compiler. A windows batch file is provided in the ebu-bmx/EBUCoreProcessor/schema/compilexsd.bat to perform this operation. Update the location of the xsd.exe compiler in this file and then execute the batch script to generate updated source files (which are copied into the proper directory automatically).

4.2 Generating new EBUCore KLV-based definition code

Whenever the definition of the KLV-based representation of EBUCore changes, the program code that implements it must also be regenerated, using a combination of tools provided with the EBU MXF SDK. The sequence of execution is illustrated in Figure 5.

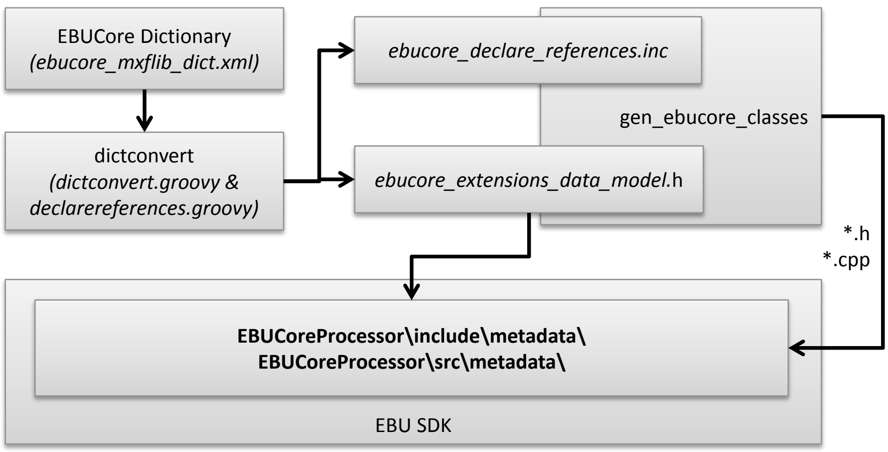

Figure 5: Sequence of KLV-based model code generation.

First, the dictionary (*EBUCoreProcessor/dictionary/ebucore_mxflib_dict.xml*) must be updated to reflect the changes of the KLV register specifications, which is a trivial operation (the existing elements in the dictionary serve as an exhaustive example).

The dictionary is then converted to a C header file (*ebucore_extensions_data_model.h*) which contains the low-level definitions of the EBUCore KLV-based model as an extension of the standard MXF data model defined in the libMXF project. This conversion is done using a Groovy script called from a batch file (*EBUCoreProcessor/dictionary/dictconvert.bat*; update the first line of the batch script with the correct Groovy installation location).

The generated header file (along with the *ebucore_declare_references.inc* file which defines how MXF metadata sets link with each other) is used by the *gen_ebucore_classes* in the libMXF++ project to create the metadata definition code. Recompile the gen_ebucore_classes project with the updated files and then execute it with the destination directory of where to create the source files as the single argument (make sure the destination is empty when doing so). The resulting C++ header and source files (along with the *ebucore_extensions_data_model.h* header file) can then be copied to the respective *EBUCoreProcessor/include/metadata* and *EBUCoreProcessor/src/metadata* directories where they can be employed by the EBU SDK.

When new classes are added to the metadata model, the *EBUCoreProcessor/include/metadata/EBUCoreDMS++.h* and *EBUCoreProcessor/src/metadata/EBUCoreDMS++.cpp* file should be updated, respectively with an #include directive for the newly created header file in the metadata directory and with a declaration of the object factory for the new object type (existing entries present in the file can serve as examples).

Finally, note that while the object model code can be generated for both the XML Schema and KLV-based representations, the bi-directional mapping between them (in EBUCoreProcessor\src\EBUCoreMapping.cpp and EBUCoreProcessor\src\EBUCoreReverseMapping.cpp) must still be updated manually.