(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
22 December 2011 (22.12.2011)

PCT

(10) International Publication Number
WO 2011/159605 A1

(51) International Patent Classification:
H04N 21/84 (2011.01) *H04N 7/24* (2011.01)
H04N 21/85 (2011.01)

(21) International Application Number:
PCT/US2011/040168

(22) International Filing Date:
13 June 2011 (13.06.2011)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/354,422 14 June 2010 (14.06.2010) US
61/354,424 14 June 2010 (14.06.2010) US

(71) Applicant *(for all designated States except US)*: **TECHNICOLOR USA INC** [US/US]; 101 W. 103rd Street, INH-3340, Indianapolis, Indiana 46290 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **WU, Zhenyu** [CN/US]; 9016 Tamarron Drive, Plainsboro, New Jersey 08540 (US). **ZHU, Li Hua** [CN/US]; 333 Andover Drive, # 207, Burbank, California 91504 (US).

(74) Agents: **SHEDD, Robert, D.** et al.; Thomson Licensing LLC, 2 Independence Way, Suite #200, Princeton, New Jersey 08540 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report (Art. 21(3))*

(54) Title: METHOD AND APPARATUS FOR ENCAPSULATING CODED MULTI-COMPONENT VIDEO

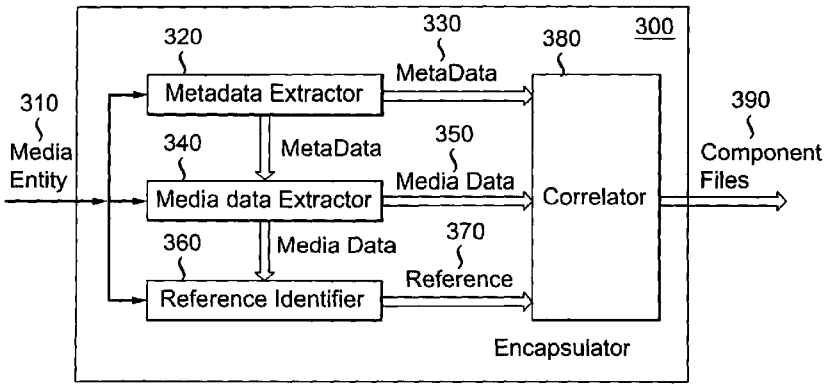

FIG. 3

(57) Abstract: A method and a device for encapsulating a media entity containing more than one layer into multiple component files, each for one layer, are described along with the corresponding method and device for component file reading. A new box for ISO BMFF and extensions to the Extractor data structure of SVC / MVC file formats are proposed. The new box enables access of referenced component files in parallel with the processing of the current component file. The extractor extensions of the invention allow NAL units referencing across different component files. The present invention enables adaptive HTTP streaming of the media files.

WO 2011/159605 A1

METHOD AND APPARATUS FOR ENCAPSULATING CODED MULTI-COMPONENT VIDEO

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application for patent claims the benefit of priority from U.S. Provisional Patent Application Serial No. 61/354,422, entitled "Extension to the Extractor data structure of SVC/MVC file formats," and filed on June 14, 2010, and U.S. Provisional Patent Application Serial No. 61/354,424, entitled "Some extensions for ISO Base Media File Format for HTTP streaming," and filed on June 14, 2010. The teachings of the above-identified provisional patent applications are expressly incorporated herein by reference.

The present application is related to the following co-pending, commonly owned U.S. Patent Application Serial No. __/____ entitled "*Method and Apparatus for Encapsulating Coded Multi-component Video*", filed concurrently herewith (Attorney Docket No. PU100140). The teachings of the non-provisional patent applications identified immediately above are expressly incorporated herein by reference.

TECHNICAL FIELD

The present invention relates generally to HTTP streaming. More specifically, the invention relates to encapsulating a media entity for coded multi-component video streams such as scalable video coding (SVC) steams and multi-view coding (MVC) streams for HTTP streaming.

BACKGROUND OF THE INVENTION

In HTTP streaming applications, an encoded video is often encapsulated and stored at the server side as a file that is compliant with BMFF, such as an MP4 file. Moreover, to realize adaptive HTTP streaming, the file is usually divided into multiple movie fragments and these fragments are further grouped into segments, which are addressable by client URL requests. In practice, different encoded representations of the video content are stored in these segments, so that a client can dynamically choose the desired representation to download and playback during a session.

Encoded layered video, such as an SVC or MVC bitstream, provides natural support for such bitrate adaptation by enabling different operating points, i.e., representations, in terms of temporal / spatial resolutions, quality, views, etc., by decoding different subsets of the bitstream. However, existing ISO Base Media File Format (BMFF) standards, such as the MP4 file format, do not support separate access of each layer or representation, and thus are not applicable to the HTTP streaming application. As shown in Fig. 1, in MP4 file format, the metadata for all the layers or representations for one media file are stored in the *moov* Movie Box, while the media content data for all the layers or representations are stored in the *mdat* Movie Box. In HTTP streaming, when the client requests one layer, the whole file has to be sent since all the layers or representations are mixed together and the client does not know where to find the required layer or representation.

As will be seen later, in adaptive HTTP streaming applications, it is desirable to be able to reference media data samples, such as network abstract layer (NAL) units, across movie fragment or component file boundaries. In SVC/MVC context, such a reference may be built by using mechanisms like "Extractor". Extractor is an internal file data structure defined in the SVC / MVC Amendments to the AVC file format extension of BMFF: *Information Technology – coding of audio-visual objects – Part 15: Advanced Video Coding (AVC) file format, Amendment 2: File format support for Scalable Video Coding, 2008, pages 15-17.* Extractor is designed to enable extraction of NAL units from other tracks by reference, without copying. Here *track* is a timed sequence of related samples in an ISO base media file. For media data, a track corresponds to a sequence of images or sampled audio. The syntax of Extractor is shown below:

```
class aligned(8) Extractor () {
    NALUnitHeader( );
    unsigned int(8)        track_ref_index;
    signed int(8)   sample_offset;
    unsigned int ((lengthSizeMinusOne + 1) * 8)
      data_offset;
    unsigned int ((lengthSizeMinusOne + 1) * 8)
      data_length;
}
```

The semantics of the Extractor data structure are:

NALUnitHeader: The NAL unit structure as specified in *ISO/IEC 14496-10 Annex G* for NAL units of type 20:

nal_unit_type shall be set to the extractor NAL unit type (type 31).

forbidden_zero_bit, reserved_one_bit, and reserved_three_2bits shall be set as specified in *ISO/IEC 14496-10 Annex G.*

Other fields (nal_ref_idc, idr_flag, priority_id, no_inter_layer_pred_flag, dependency_id, quality_id, temporal_id, use_ref_base_pic_flag, discardable_flag, and output_flag) shall be set as specified in B.4 of *Information technology – Coding of audio-visual objects – part 15: Advanced Video Coding (AVC) file format, Amendment 2: File format support for Scalable Video Coding, ISO/IEC 14496-15:2004/Amd.2:2008, page 17.*

track_ref_index specifies the index of the track reference of type 'scal' to use to find the track from which to extract data. The sample in that track from which data is extracted is temporally aligned or nearest preceding in the media decoding timeline, i.e. using the time-to-sample table only, adjusted by an offset specified by sample_offset with the sample containing the Extractor. The first track reference has the index value 1; the value 0 is reserved.

sample_offset gives the relative index of the sample in the linked track that shall be used as the source of information. Sample 0 (zero) is the sample with the same, or the closest preceding, decoding time compared to the decoding time of the sample containing the extractor; sample 1 (one) is the next sample, sample -1 (minus 1) is the previous sample, and so on.

data_offset: The offset of the first byte within the reference sample to copy. If the extraction starts with the first byte of data in that sample, the offset takes the value 0. The offset shall reference the beginning of a NAL unit length field.

data_length: The number of bytes to copy. If this field takes the value 0, then the entire single referenced NAL unit is copied (i.e. the length to copy is taken from the length field referenced by the data offset, augmented by the additional_bytes field in the case of Aggregators).

Further details can be found in *Information technology – Coding of audio-visual objects – part 15: Advanced Video Coding (AVC) file format, Amendment 2: File format support for Scalable Video Coding, ISO/IEC 14496-15:2004/Amd.2:2008.*

Currently extractors are only able to extract, by reference, the NAL units from other tracks, but within the same movie box/fragment. In other words, it is not possible to use extractors to extract NAL units from a different segment or file. This restriction limits the use of extractors in the above use case.

If a client has already downloaded one or more content component of a piece of media content from the server, and is in the process to download another content component, the client needs to know whether the previously downloaded content components are among the set of the dependent components of the new one, so that it can make other requests as necessary to download the complete component set. This use case also requires a mechanism to signal an external dependent content component and its location information.

In BMFF, there is a box type called "tref", which is used to provide a reference from the containing track to another track in the presentation. This box can be used to describe the dependencies among tracks, however, the dependency is limited to tracks within the same media file.

One approach is to signal such information using some out-of-band mechanism. For example, for an HTTP streaming application, the server can send a manifest file to the client before a session starts. The manifest file is a file that contains dependency and location information of each content component of a requested media content. Then the client is able to request all the necessary component files. However, this out-of-band approach is not applicable for local file playback, where no manifest file is available.

Prior solutions to the problems mentioned above have not adequately been established in the art. It would be desirable to provide the ability to parse and encapsulate layers without sacrificing speed and transport efficiency. Such results have not heretofore been achieved in the art.

SUMMARY OF THE INVENTION

This invention directs to methods and apparatuses for encapsulating component files from a media entity containing more than one layer and for reading a component file.

According to an aspect of the present invention, there is provided a method for encapsulating and creating component files from a media entity containing more than one layer. The method extracts metadata and media data corresponding to the extracted metadata for each layer from the media entity. The extracted media data and metadata are associated to enable the creation, for each layer, of a component file containing the extracted metadata and the extracted media data.

According to another aspect of the present invention, there is provided a file encapsulator. The file encapsulator includes an extractor for extracting metadata and media data corresponding to the extracted metadata for each layer from the media entity; and a correlator for associating the extracted media data with the extracted metadata to enable creation, for each layer, of a component file.

BRIEF DESCRIPTION OF THE DRAWINGS

The above features of the present invention will become more apparent by describing in detail exemplary embodiments thereof with reference to the attached drawings in which:

Figure 1 shows an example MP4 file format.

Figure 2 shows one embodiment of the current invention to encapsulate a media entity.

Figure 3 shows the structure of an Encapsulator used to encapsulate or create component files from a media entity which contain multiple layers/representations,

Figure 4 shows an example of associating additional media data with component files based on dependency relationship.

Figure 5 shows an example to extract an NAL unit, by reference, from a movie box/fragment that is different from the one that the extractor is within.

Figure 6 shows the involved encapsulation operations for an SVC / MVC type video bitsteam into multiple component files using one of the invented new extractor data structures.

Figure 7 shows the structure of a file reader used to read the component files.

Figure 8 shows the process of reading an encapsulated component file for a video decoder involving one embodiment of the present invention.

Figure 9 shows the encapsulation operations for an SVC / MVC type video bitstream into multiple movie fragments using another preferred new extractor data structures.

Figure 10 shows the process of reading an encapsulated component file for a video decoder involving another embodiment of the present invention.

DETAILED DESCRIPTION

In present invention, a media entity, such as a media file or a set of media files or a streaming media, is divided or encapsulated into multiple movie component files, which are addressable by client URL requests. Here, a component file is used in a broader sense that it represents a fragment, a segment, a file and other equivalent terms thereto.

In one embodiment of the present invention, a media entity containing multiple representations or components is parsed to extract metadata and media data for each representation/component. Examples of the representation/component includes layers, such as layers with various temporal/spatial resolutions and quality in SVC, and views in MVC. In the following, layers are also used to refer to representations/components, and these terms are used interchangeably. The metadata describes, for example, what is contained in the media entity for each representation and how to use the media data contained therein. The media data contain media data samples required for serving the purpose of the media data, e.g. decoding of the content, or any necessary information on how to obtain the required data samples. The extracted metadata and media data for each representation or layer are associated/correlated and stored together for user access. The storing operation can be done physically on a hard drive or other storing media, or can be performed virtually through a relationship management mechanism so that the metadata and media data appear to be stored together when interfacing with other applications or modules when they indeed are actually located in different places on storing media. Fig. 2 illustrates an example of this embodiment. In Fig. 2, a media entity contains three layers: base layer, enhancement layer 1 and enhancement layer 2. The media entity is parsed to extract the metadata and media data for each of the three layers, and those data are stored separately as component files with the metadata and corresponding media data associated together.

Fig. 3 shows the structure of a preferred encapsulator 300 used to encapsulate and create component files from a media entity which contain multiple layers, such as SVC encoded

videos. The inputs media entity 310 is passed to a metadata extractor 320 and a media data extractor 340. The metadata extractor 320 extracts the metadata 330 for each layer. The media data extractor 340 takes in the metadata 330 and extracts the corresponding media data 350. Note that in a different embodiment, the metadata extractor 320 and the media data extractor 340 are implemented as one extractor. Both data, metadata 330 and media data 350, are feed into a correlator 380 which associates these two types of data and creates the output component files 390, one component file for each layer.

A layered video, such as a video encoded by AVC extensions of SVC or MVC, contains multiple media components (scalable layers or views). Such an encoded bitstream can provide different operating points, i.e., representations or layers, in terms of temporal / spatial resolutions, quality, views, etc., by decoding different subsets of the bitstream. Furthermore, there exist coding dependencies among the layers of the bitstream, i.e., the decoding of a layer may depend on other layers. Therefore, to request one of such a bitstream's representations may require retrieving and decoding one or more components or media data from the encapsulated video file. To facilitate the extraction process for different representations, an encoded layered video is often encapsulated into an MP4 file in a way that each layer is stored separately in different segments or component files. In this case, it needs to be taken into account that certain media data samples, such as NAL units, of the bitstream are required by, or related to, multiple segments or component files, due to the decoding dependencies described above or other dependencies based on the application.

In another embodiment of the present invention, additional media data required by a segment or a component file are extracted and associated with the segment or component file. Figure 4 shows an example of this embodiment. In the figure, an SVC bitstream has three spatial layers, HD1080p, SD and QVGA. Three movie fragments or component files are formed corresponding to the three operating points, and each is addressable by a different URL. Inside each movie fragment or component file, all the media data samples, NAL units in this example, required for decoding are copied and stored as media samples contained in the "mdat" box. So, when a client requests a particular operating point or representation by using a proper URL, the server can retrieve the corresponding movie fragment or component file and forwarded to the client. In this embodiment, the media data extractor 340 in Fig. 3 further extracts, for each layer, from the input media entity 310 additional media data related

to the extracted media data for each of the layers. Correlator 380 further associates the additional extracted media data for each layer to create corresponding component files.

For the sake of storage space saving, it is desirable to be able to reference media data samples, such as NAL units, across movie fragment or component file boundaries, without actually duplicating the same data in each component file. However, ISO Base Media File Format (BMFF) and its extensions currently do not support this feature. To solve this problem, in a further embodiment of the present invention, a reference is identified and built for those additional media data that are related to or required by the media data of a movie fragment or a component file. The reference, rather than those additional media data, is associated with the component file along with the metadata and media data thereof. One can embed the references into the extracted media data for each layer, and then associate the extracted metadata and extracted media data for each layer for creating corresponding component files.

In this embodiment, a Reference Identifier 360 is added to the structure of the Encapsulator 300. The Reference Identifier 360 identifies, from input media entity 310, references 370 to those additional media data that are related to extracted media data 350 for each layer. Then references 370 are associated, via correlator 380, with extracted metadata 330 and extracted media data 350 for each layer, e.g. by embedding said references into said extracted media data 350, for creating corresponding component files 390.

As discussed earlier, in SVC/MVC context, such a reference may be built by using mechanisms like "Extractor". Currently extractors are only able to extract, by reference, the NAL units from other tracks, but within the same movie box/fragment. In other words, it is not possible to use extractors to extract NAL units from a different segment or file. This restriction limits the use of extractors in other cases. Hereafter, an extension to the extractor data structure is disclosed, where the extension is aimed to support efficient encapsulation of SVC / MVC type layered video content into multiple component files as described before.

The extension is added to provide the extractor data structure with the extra capability to reference to NAL units that reside in a different movie box/fragment or component file other than the one in which extractor resides.

The extended Extractor is defined as the following:

Syntax:

```
aligned (8) class DataEntryUrlBox (bit (24) flags)
        extends FullBox ('url', version = 0, flags) {
        string location;
}
aligned (8) class DataEntryUrnBox (bit (24) flags)
        extends FullBox ('urn', version = 0, flags) {
        string name;
        string location;
}
class aligned (8) Extractor () {
    NALUnitHeader ( );
    DataEntryBox (entry_version, entry_flags)  data_entry; // added extension
    unsigned int(8)   track_ref_index;
    signed int(8)     sample_offset;
    unsigned int ((lengthSizeMinusOne + 1) * 8)
      data_offset;
    unsigned int ((lengthSizeMinusOne + 1) * 8)
      data_length;
}
```

Semantics:

data_entry is a Uniform Resource Locator (URL) or Uniform Resource Name (URN) entry. *Name* is a URN, and is required in a URN entry. *Location* is a URL, and is required in a URL entry and optional in a URN entry, where it gives a location to find the resource with the given name. Each is a null-terminated string using UTF-8 characters. If the self-contained flag is set, the URL form is used and no string is present; the box terminated with the entry-flags field. The URL type should be of a service that delivers a file. Relative URLs are permissible and are relative to the file containing the Movie Box/Fragment that contains the track that the Extractor belongs to.

Other fields have the same semantics as the original Extractor described before.

With the extended extractor, it is now possible to extract a NAL unit, by reference, from a movie box/fragment that is different from the one the extractor is within. Figure 5 shows such an example, with the same SVC bitstream as Figure 4 but using the new extended Extractor data structure. As can be seen from the figure, now the SD movie fragment can reference to the NAL units from the QVGA movie fragments. Likewise, the HD1080p movie fragment can use the extractors to reference NAL units from both QVGA and SD movie fragments. Compared to Figure 4, no NAL units are duplicated across these movie fragments, thus storage space is saved.

Figure 6 shows the involved encapsulation operations for an SVC / MVC type video bitsteam into multiple movie fragments or component files using the invented new extractor data structure. The process starts at step 601. Each NAL unit is read in one by one in step 610. If the end of the bitstream is reached in step 620, the process stops at 690; otherwise, the process proceeds to the next step 630. Decision step 630 determines if the current NAL unit depends on NAL units from other track for decoding. If the determination is that the current NAL unit does not depend on NAL units from other tracks for decoding, the control is then transferred to step 640, wherein a sample is formed using the current NAL unit and is placed in the current track. If the determination from step 630 is that there is dependency between the current NAL unit and NAL units from other track, the process goes on to step 650. Decision step 650 further determines if the track from which NAL units are required by the current NAL unit resides within the same movie fragment. If the determination is that the track resides in the same movie fragment, step 670 is employed to fill in an extended Extractor to reference to the NAL unit from that other track. If the determination is that the track resides in a different movie fragment, the URL or URN of such a movie fragment is identified in step 660 and the process proceeds to step 670 with the identified URL and URN to be filled in an extended Extractor. After such an extended Extractor is filled in, it is embedded into the current track in step 680. Then, the process starts over with the next NAL unit in step 610.

In a different embodiment, references 370 are embedded into extracted metadata 330 and indices to reference 370 are added to extracted media data 350 via correlator 380, which further associates the metadata and the media data for each layer for creating corresponding component files 390. In the context of the ISO Media Base File Format, a box called HTTP

Streaming Information Box is disclosed. This box contains the information that can assist the HTTP streaming of the ISO file. It is preferred that the HTTP Streaming Information Box be placed as early as possible in the component files, e.g. at the beginning of the files. The box can also serve as a source by the server when forming a manifest file for the client. Another type of box called Media Reference Box which is contained in the HTTP Streaming Information Box is also disclosed. This box contains the information about the external dependent files. The extractor structure is further extended so that it can reference media samples across different component files. The information contained in Media Reference Box can be utilized by extractors to save signaling overhead.

The detailed definition for the proposed HTTP streaming information box, media reference box and further improved extractors are as follows.

- HTTP Streaming Information Box

Definition:

Box Type: 'hsin'
Container: File
Mandatory: No
Quantity: Zero or one

The HTTP Streaming Information Box aids the HTTP streaming operation of an ISO media file. It contains relevant information about HTTP streaming delivery of the file, including Media Reference Box as defined below, among other possible types of boxes. The HTTP Streaming Information Box is preferably placed as early as possible in files, for maximum utility.

Syntax:

```
aligned(8) class HTTPStreamingInfoBox extends Box ('hsin') {
}
```

- Media Reference Box

Definition:

Box Type: 'mref'
Container: 'hsin'

Mandatory: No
Quantity: Zero or one

Media Reference Box is contained in HTTP Streaming Information Box, and it contains a table of the data references in the form of URL that declare the locations of the external files that each track included in the box is dependent on. By reading this box, the file reader is able to identify the external dependent file sources, such as external component files, of a track in the file, as well as means to retrieve them.

Syntax:

```
aligned(8) class DataEntryUrlBox ( bit(24) flags ) extends Box ( 'url') {
    string location;
}

aligned(8) class MediaReferenceBox extends Box ('mref' ) {
    unsigned int(16)     entry_count;
    for ( i = 1; i <= entry_count; i++) {
            unsigned int(32)     track_ID;
            unsigned int(16)     dependent_source_count;
            for ( j = 1; j <= dependent_source_count; j++){
                    DataEntryUrlBox    data_entry;
            }
    }
}
```

Semantics:

entry_count: is an integer that counts the actual entries;

track_ID: is an integer that uniquely identifies the track in the file upon which the box is applied;

dependent_source_count: is an integer that counts the external media sources that the track in the file with track_ID that is dependent on;

data_entry: is a URL entry that points to one external media source the designated track is dependent on. Each is a null-terminated string using UTF-8 characters. The URL type should be of a service that delivers a file. Relative URLs are permissible and are relative to the file that contains this media reference box.

Media Reference Box, as defined above, is designed to facilitate, in a number of ways, HTTP streaming of a media entity containing more than one layer.

First, it can explicitly signal the dependency relationship among component files at the beginning of a component file through the reference table. Thus, once the client has downloaded a small portion of the component file, it is able to know all the related external component files of its track(s), and make corresponding requests to obtain the complete set(s) for playback through the references contained in the table, if necessary.

Second, the in-file information from the box can be easily extracted to be included in a manifest file. Such information in the manifest can help the client, before actual HTTP streaming, discover relevant service information and perform the corresponding service initialization, such as requesting all the associated component files, allocating necessary buffer resources, etc.

Third, when the client requests a different representation of some multi-component media content, which has another representation already been delivered as a component file, the client can check the corresponding Media Reference Box in the file to see if the file contains any of the dependent components of the new representation that can be reused.

Finally, the box helps reduce the signaling overhead of the extended Extractor structure as defined below.

- Extractors

Extractor is further proposed to extend its capability of referencing data from tracks of external media files.

Extended syntax:

```
class aligned(8) Extractor ( ) {
    NALUnitHeader ( );
    unsigned int(16)    media_reference_index;
```

```
        unsigned int(8)     track_ref_index;
        signed int(8)       sample_offset;
        unsigned int ((lengthSizeMinusOne + 1) * 8)
                    data_offset;
        unsigned int ((lengthSizeMinusOne + 1) * 8)
                    data_length;
    }
```

Semantics:

media_reference_index: specifies an index of the entry to the reference table contained in the Media Reference Box that has the same associated track_ID value as the track that contains the extractor. If media_reference_index equals to 0, the extractor references to the data from another track but within the same file as the extractor. In this case, there shall not be a reference table in Media Reference Box that has the same track_ID value as the track. If media_reference_index is between 1 and the value of dependent_source_count from the reference table associated with the track from the Media Reference Box, the URL referenced by media_reference_index from the reference table points to an external file, which contains a track from which the extractor extracts data.

The semantics of other fields remain the same as the original extractor definition.

With the further extended extractor structure, it is now possible to use extractors to link to and extract data from a track that belongs to an external component file. It is especially useful when content components from an encoded piece of multi-component media content, such as encoded by SVC or MVC, are encapsulated into different component files. With the extended extractors, extraction can take place across file boundaries. This avoids duplicating the same data in different component files.

Figure 9 shows the involved encapsulation operations for an SVC / MVC type video bitsteam into multiple movie fragments or component files using the disclosed HTTP Streaming Information Box and Media Reference Box as well as the further extended extractor data structure. This process is similar to the process shown in Fig. 6 with a few modifications due to the above described boxes, and the further extension of the extractor. After location information URL/URN is indentified in step 660, the location information is

used to fill in the reference table in the mref box (Media Reference Box) in step 965. Step 970 further fills in an Extractor with the indices to the location information of the reference table. The extractor is then embedded into the current track. When the end of the bitstream is reached at step 620, mref box and its container hsin box (HTTP Streaming Information Box) are embedded into the metadata of the component file.

To read a component file, a file reader 700 shown in Fig. 7 is employed. A parser 710 first parses the component file to get metadata and media data, and a reference if available. If, according to the decoded reference, the media data are related to media data of other component files such as through decoding dependency, a retriever 720 retrieves the related media data from other component files as indicated in the reference. A processor 730 further processes the metadata and media data obtained from the component file as well as the additional media data if available. The parsing operation by the parser 710 includes various necessary operations to obtain the metadata, the media data that are ready for the processor 730, and the reference ready for the retriever 720. It will include further parsing the metadata and/or the media data when necessary. In one embodiment, the reference is embedded in the media data, and thus the reference is obtained by parsing the media data. If a reference is available, the parsing step further includes analyzing the syntax of the reference and decoding the reference. The processor 730 can contain a video decoder if the component file contains video content. In a different embodiment, the parser and the retriever can be incorporated in the processor.

Figure 8 shows the process of reading an SVC / MVC type video bitstream for a video decoder involving the present invention. Step 801 accesses a component video file whose metadata and media data for each layer are identified in step 805. The identified metadata and media data are parsed in step 810 and each NAL unit of the media data is read in one by one in step 815. For the current NAL unit, a decision is first made at step 820 to determine if the end of the bitstream is reached, and the process ends at step 825 if the answer is “Yes”. Otherwise, the process proceeds to decision step 830 to determine if the current NAL unit is an extractor. If it is not an extractor, which means it is a normal NAL unit containing decoding data, the NAL unit is sent to decoder at step 835. If the current NAL unit is an extractor, it is determined at step 840 that whether the current NAL unit depends on a NAL unit outside the same component file or not. If the required NAL unit is within the same

component file, it is retrieved from the current file in step 845 and sent to the decoder at step 835. If the required NAL unit is from another component file, the NAL unit is located using the reference information *Data_entry* in the extractor in step 850, retrieved from the remote file in step 855 and then sent to the decoder in step 835.

In another embodiment, the reference is identified in the parser 710 by parsing the media data to get the embedded reference indices, and to obtain corresponding reference according to the reference indices. The corresponding process of reading an SVC / MVC type video bitstream for a video decoder is shown in Fig. 10, which is similar to the process of Fig. 8. At step 810, since the reference is placed at the beginning of the component file according to a preferred embodiment, the parsing of the metadata in step 810 enables the analysis of the reference contained therein in parallel with the parsing of the media data. When analyzing the reference, other component files that are referenced to are identified in step 1014. Retrieving of those other component files are started in step 1012 in parallel with the remaining steps of the process. After accessing the location information of the component file that the current NAL unit depends on in step 850, local storage, such as media buffer, is checked for availability of such a component file. If the required component file is available locally, then the NAL unit of the local copy is retrieved; otherwise, the NAL unit from remote file is retrieved. Note that the local copy of the component file can be obtained by the parallel retrieving in step 1012, or it can be obtained from a previous request of such component file.

Although preferred embodiments of the present invention have been described in detail herein, it is to be understood that this invention is not limited to these embodiments, and that other modifications and variations may be effected by one skilled in the art without departing from the scope of the invention as defined by the appended claims.

CLAIMS

1. A method for creating component files from a media entity containing more than one layer, the method comprising the steps of:

extracting metadata for each layer from said media entity;

extracting media data from said media entity corresponding to the extracted metadata for each layer of said media entity; and

associating said extracted media data with said extracted metadata to enable creation, for said each layer, of a component file containing said extracted metadata and said extracted media data.

2. The method of claim 1, wherein said component file is at least one of a movie box, a movie fragment, a segment and a file.

3. The method of claim 1, further comprising the steps of:

extracting, for said each layer, from said media entity additional media data related to said extracted media data for said each layer; and

associating said extracted media data and said additional media data for each layer for creating corresponding component files.

4. The method of claim 1, further comprising the steps of:

identifying references to additional media data related to said extracted media data for each layer; and

associating said references with said extracted metadata and extracted media data for each layer for creating corresponding component files.

5. The method of claim 4, wherein said media data and additional media data comprise data samples.

6. The method of claim 5, wherein a data sample comprises a network abstract layer unit.

7. The method of claim 6, wherein said references contain at least one of a uniform resource locator and a uniform resource name of said network abstract layer units in said additional media data.

8. The method of claim 4, further comprising the steps of:

embedding said references into said extracted metadata for each layer; and

adding indices to said references in said extracted media data.

9. The method of claim 8, wherein said references are placed at a beginning of said component file for each layer.

10. The method of claim 8, wherein said references are filled into media reference boxes and said indices are filled into extractors.

11. A file encapsulator for creating component files from a media entity containing more than one layer, the encapsulator comprising:

an extractor for extracting metadata for each layer from said media entity and for extracting media data from said media entity corresponding to said extracted metadata for each layer of said media entity; and

a correlator for associating said extracted media data with said extracted metadata to enable creation, for said each layer, of a component file containing said extracted metadata and said extracted media data.

12. The method of claim 11, wherein said component file is at least one of a movie box, a movie fragment, a segment and a file.

13. The file encapsulator of claim 11, wherein said extractor further extracts, for said each layer, from said media entity additional media data related to said extracted media data for each layer; and said correlator further associates said extracted media data and said additional media data for each layer for creating corresponding component files.

14. The file encapsulator of claim 11, further comprising:

a reference identifier for identifying a reference to additional media data, from said media entity, related to said extracted media data for each layer, wherein said reference is associated, via said correlator, with said extracted metadata and extracted media data for each layer for creating corresponding component files.

15. The file encapsulator of claim 14, wherein said media data and additional media data comprise data samples.

16. The file encapsulator of claim 15, wherein a data sample comprises a network abstract layer unit.

17. The file encapsulator of claim 16, wherein said references contain at least one of a uniform resource locator and a uniform resource name of said network abstract layer units in said additional media data.

18. The file encapsulator of claim 14, wherein said correlator further embeds said reference into said extracted metadata for each layer and adds indices to said references in said extracted media data.

19. The file encapsulator of claim 18, wherein said correlator places said references at the beginning of said component file for each layer.

20. The file encapsulator of claim 19, wherein said references are filled into media reference boxes and said indices are filled into extractors.

21. A method for reading a component file, comprising the steps of:

parsing said component file to obtain metadata, media data and references; and

if, according to said references, said media data of said component file are related to media data of other component files, retrieving said related media data from said other component files using said references.

22. The method of claim 21, wherein said media data of said component file are related to media data of other component files according to coding dependency.

23. The method of claim 21, wherein said media data and said related media data comprise data samples.

24. The method of claim 23, wherein a data sample comprises a network abstract layer unit.

25. The method of claim 21, further comprising the step of parsing said metadata to obtain said references.

26. The method of claim 25, further comprising the steps of:

parsing said media data to get reference indices embedded therein; and

obtaining corresponding references according to said reference indices.

27. The method of claim 25, wherein the retrieving step comprises:

retrieving said other component files according to said references in parallel.

28. The method of claim 27, wherein the retrieving step further comprising:

checking local file storage;

if said local file storage contains said other component files, retrieving said other component files from said local storage.

29. A file reader, comprising:

a parser for parsing a component file to obtain metadata, media data and a reference;

a retriever for retrieving media data related to said media data from other component files according to said reference; and

a processor for processing said metadata, media data and said retrieved media data from other component files.

30. The file reader of claim 29, wherein said media data of said component file are related to media data of other component files in terms of coding dependency.

31. The file reader of claim 29, wherein said media data and said related media data comprise data samples.

32. The file reader of claim 31, wherein a data sample comprises a network abstract layer unit.

33. The file reader of claim 29, wherein said processor comprises a video decoder.

34. The file reader of claim 29, wherein said parser further comprises means for obtaining said references.

35. The file reader of claim 34, wherein said parser further parses said media data to get reference indices embedded therein, and obtains corresponding references according to said reference indices.

36. The file reader of claim 34, wherein said retriever further retrieves said other component files according to said obtained references in parallel.

37. The file reader of claim 36, wherein the retriever further checks local file storage, and if said local file storage contains said other component files, retrieves said other component files from said local storage.

5

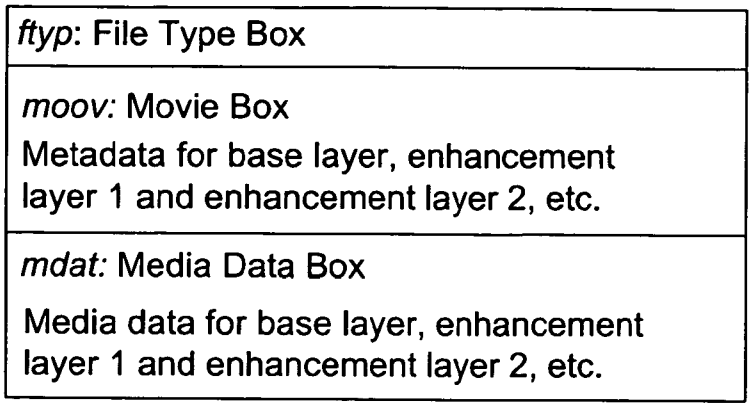
ftyp: File Type Box
moov: Movie Box
Metadata for base layer, enhancement layer 1 and enhancement layer 2, etc.
mdat: Media Data Box
Media data for base layer, enhancement layer 1 and enhancement layer 2, etc.

FIG. 1

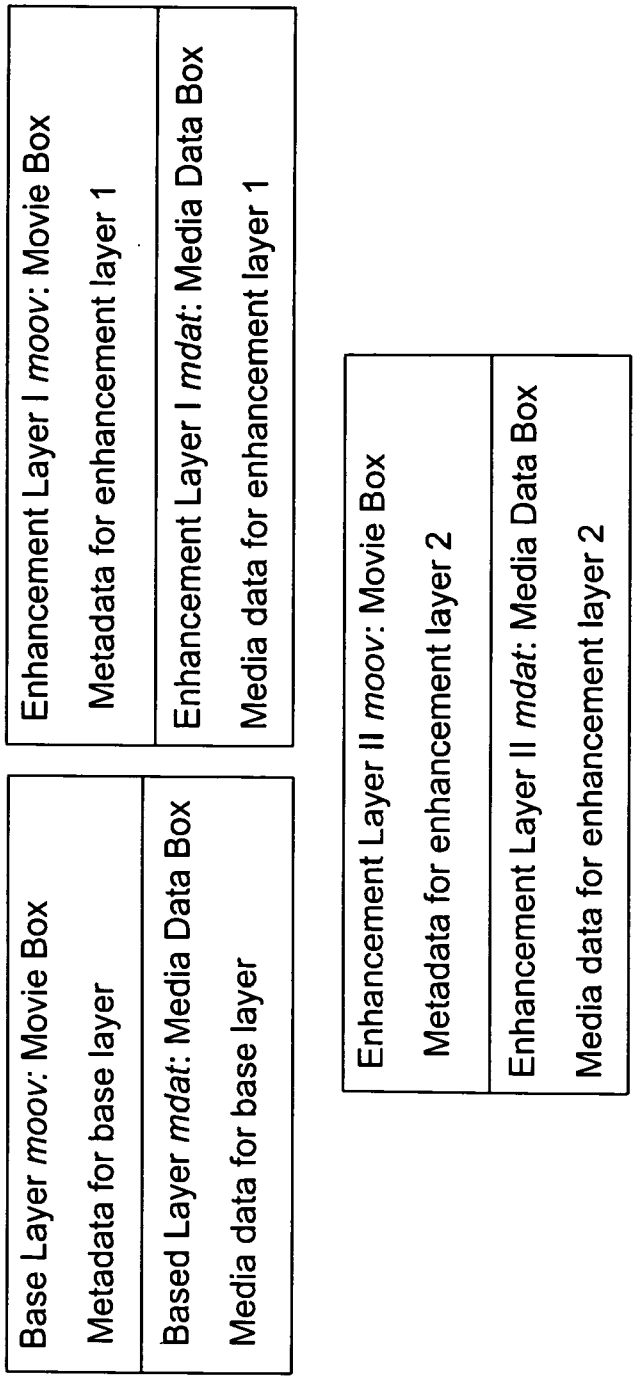
Base Layer moov: Movie Box
Metadata for base layer
Based Layer mdat: Media Data Box
Media data for base layer
Enhancement Layer I moov: Movie Box
Metadata for enhancement layer 1
Enhancement Layer I mdat: Media Data Box
Media data for enhancement layer 1
Enhancement Layer II moov: Movie Box
Metadata for enhancement layer 2
Enhancement Layer II mdat: Media Data Box
Media data for enhancement layer 2

FIG. 2

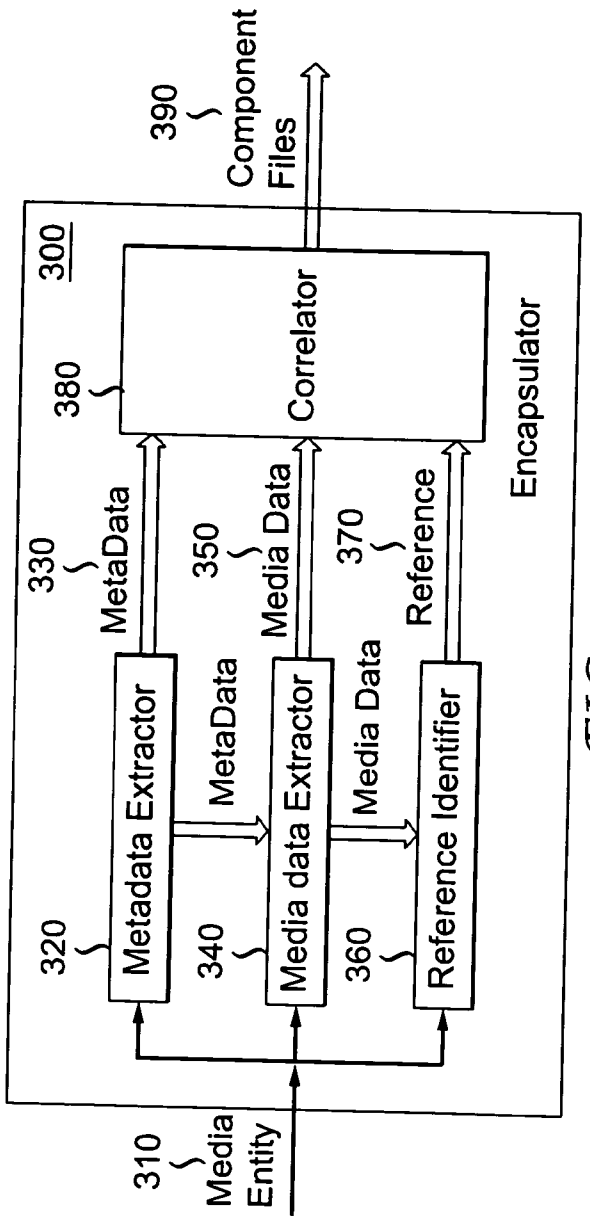
390
Component Files
300
380
Correlator
Encapsulator
330
MetaData
350
Media Data
370
Reference
Metadata Extractor
MetaData
Media data Extractor
Media Data
Reference Identifier
320
340
360
310
Media Entity

FIG. 3

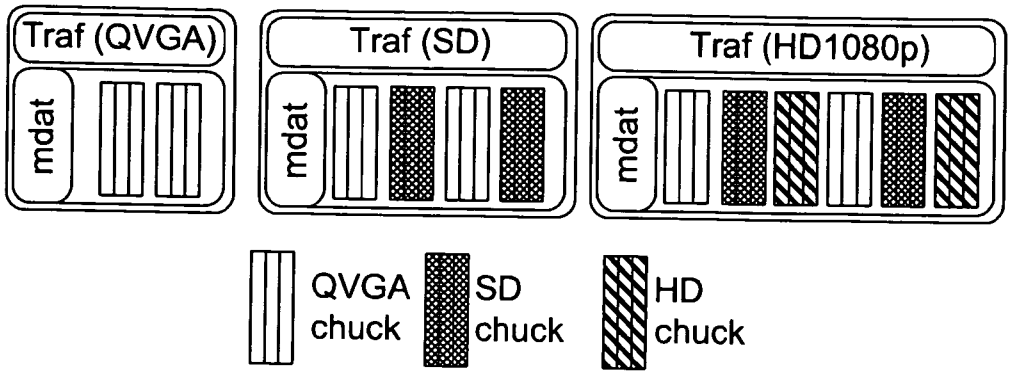
Traf (QVGA)
mdat
Traf (SD)
mdat
Traf (HD1080p)
mdat
QVGA chuck
SD chuck
HD chuck

FIG. 4

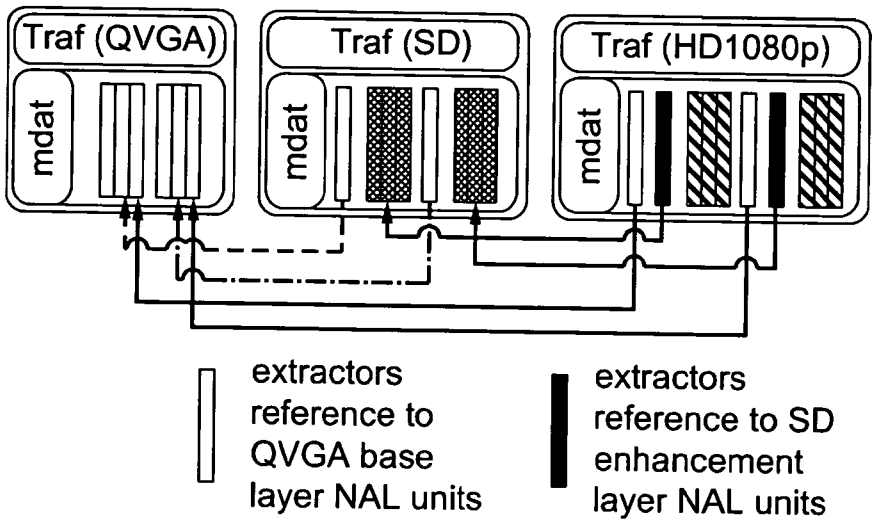
Traf (QVGA)
mdat
Traf (SD)
mdat
Traf (HD1080p)
mdat
extractors reference to QVGA base layer NAL units
extractors reference to SD enhancement layer NAL units

FIG. 5

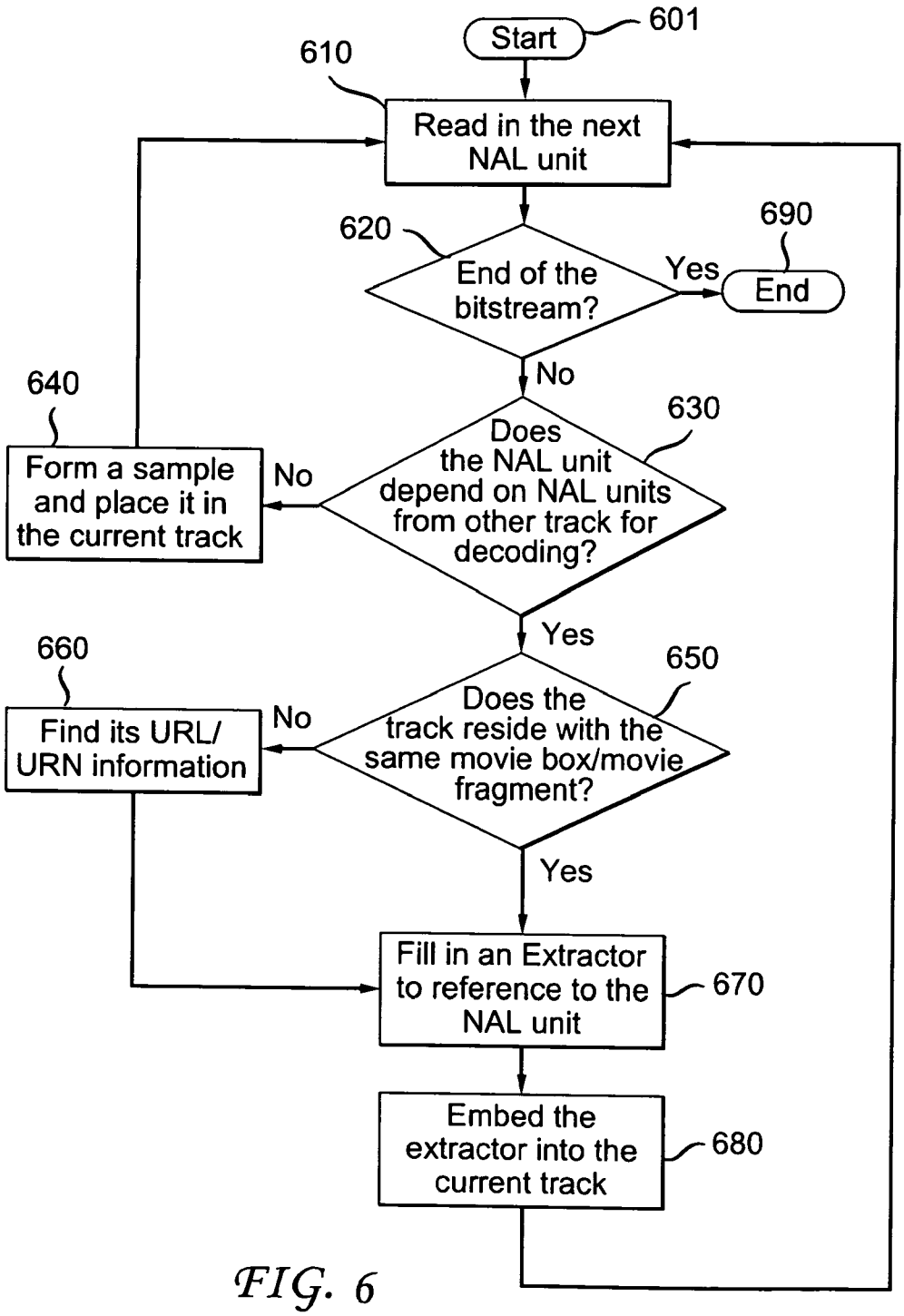
Start
601
610
Read in the next NAL unit
620
End of the bitstream?
Yes
690
End
No
640
630
Does the NAL unit depend on NAL units from other track for decoding?
Form a sample and place it in the current track
No
Yes
660
650
Does the track reside with the same movie box/movie fragment?
Find its URL/ URN information
No
Yes
Fill in an Extractor to reference to the NAL unit
670
Embed the extractor into the current track
680

FIG. 6

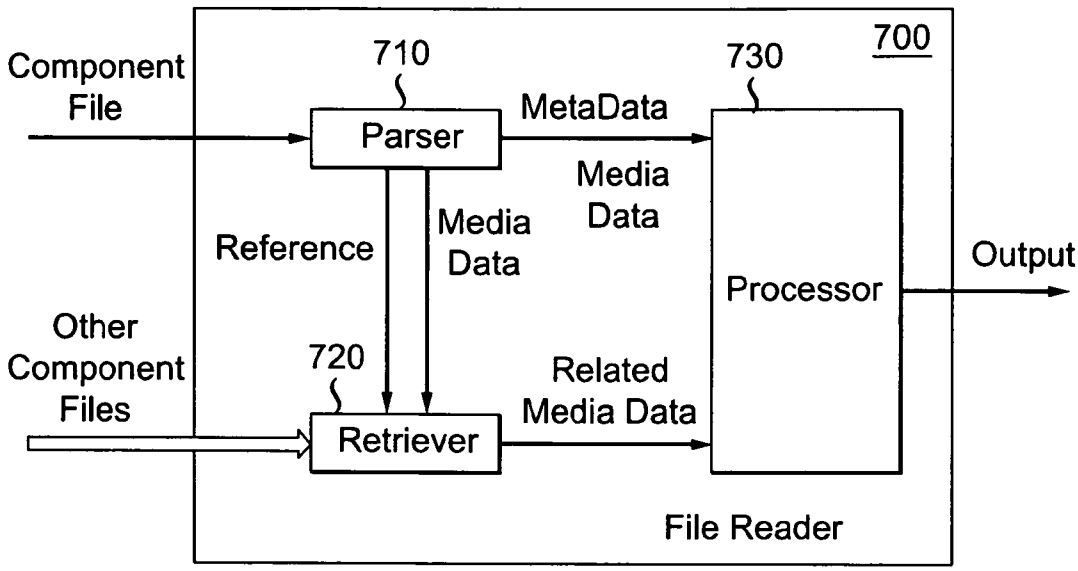
Component File
710
Parser
MetaData
730
700
Media Data
Reference
Media Data
Processor
Output
Other Component Files
720
Related Media Data
Retriever
File Reader

FIG. 7

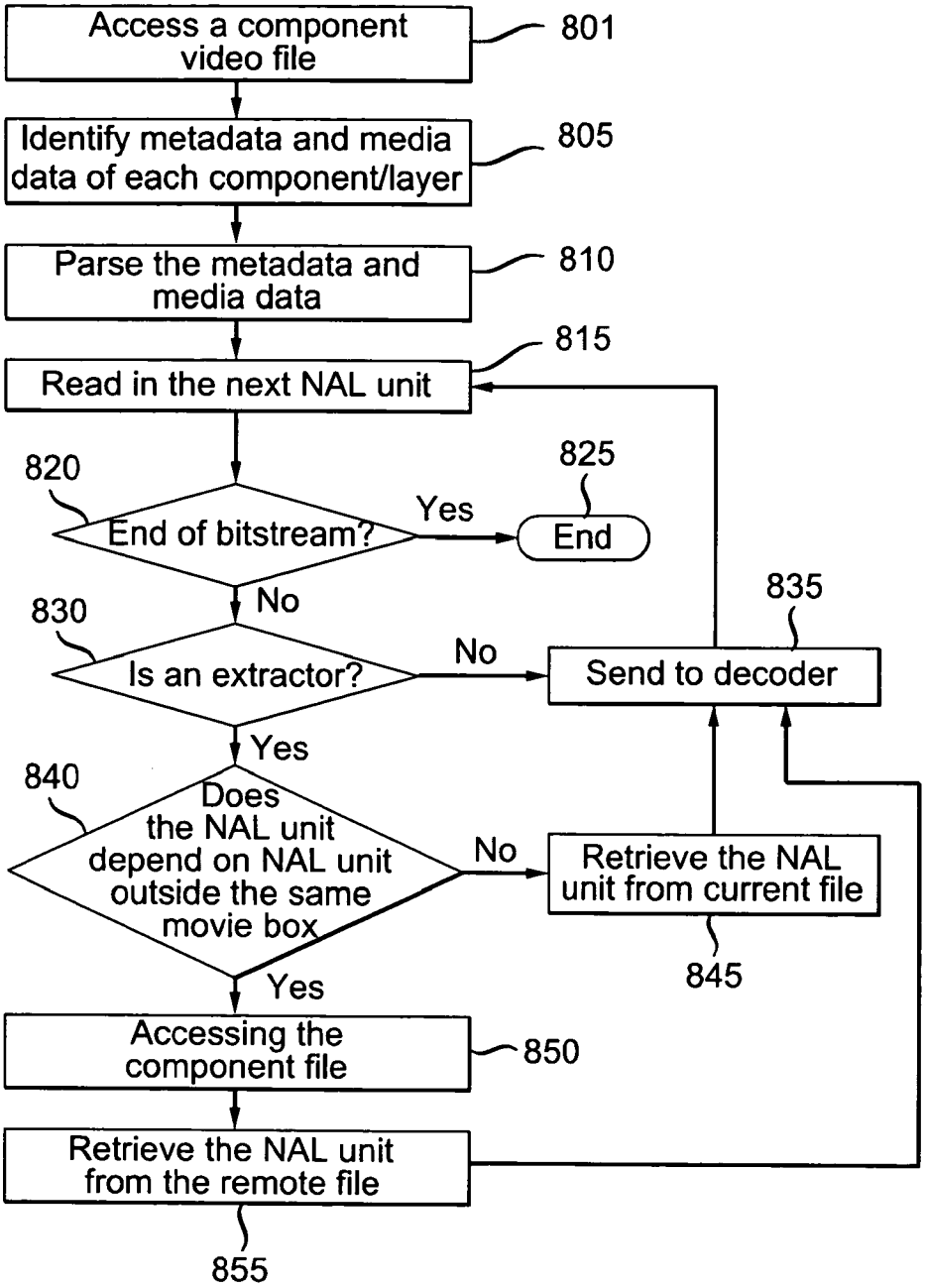
Access a component video file
801
Identify metadata and media data of each component/layer
805
Parse the metadata and media data
810
Read in the next NAL unit
815
820
End of bitstream?
Yes
825
End
No
830
Is an extractor?
No
835
Send to decoder
Yes
840
Does the NAL unit depend on NAL unit outside the same movie box
No
Retrieve the NAL unit from current file
845
Yes
Accessing the component file
850
Retrieve the NAL unit from the remote file
855

FIG. 8

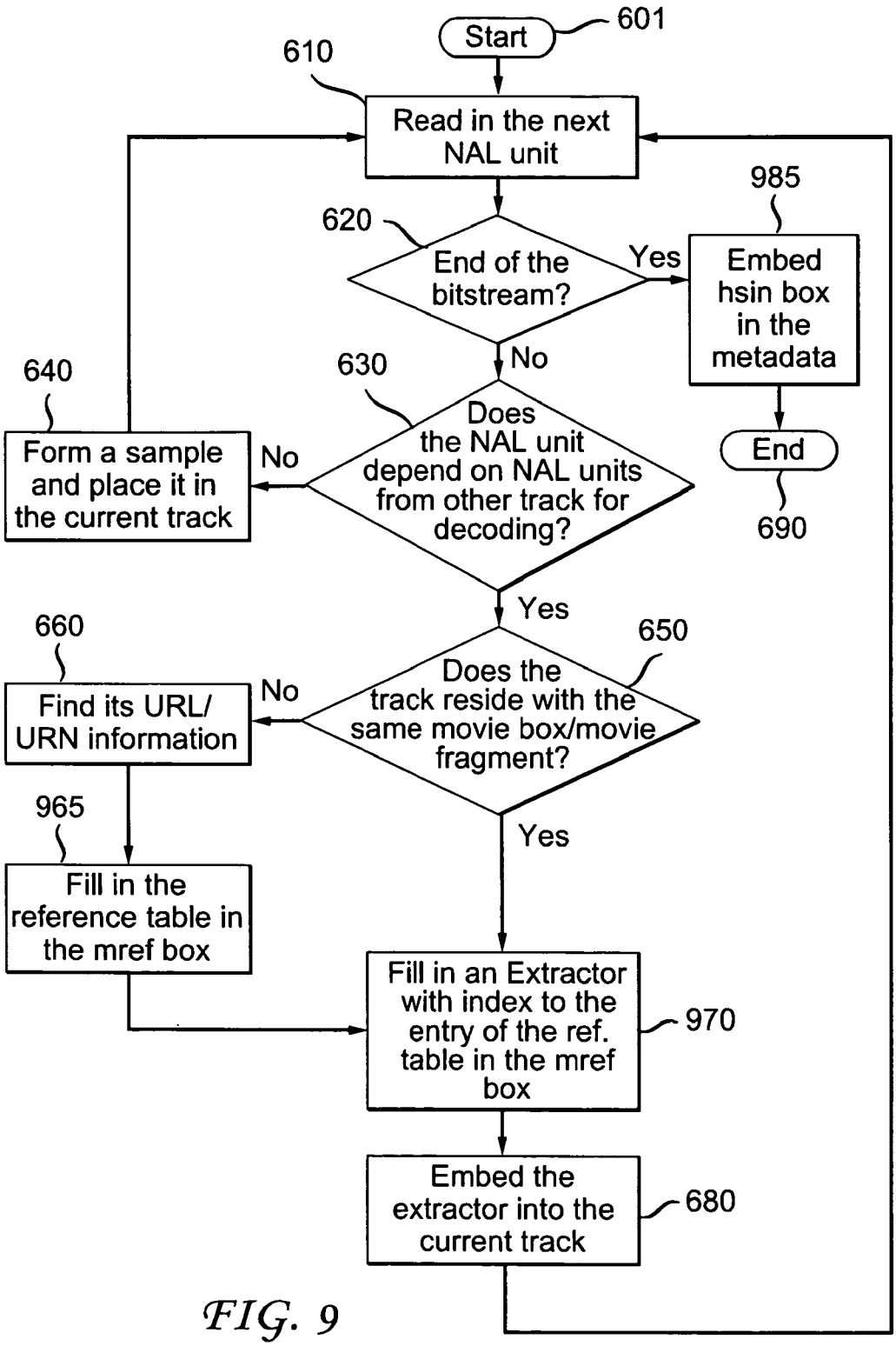
Start
601
610
Read in the next NAL unit
985
620
End of the bitstream?
Yes
Embed hsin box in the metadata
No
640
630
Does the NAL unit depend on NAL units from other track for decoding?
Form a sample and place it in the current track
No
End
690
Yes
660
650
Does the track reside with the same movie box/movie fragment?
Find its URL/ URN information
No
965
Yes
Fill in the reference table in the mref box
Fill in an Extractor with index to the entry of the ref. table in the mref box
970
Embed the extractor into the current track
680

FIG. 9

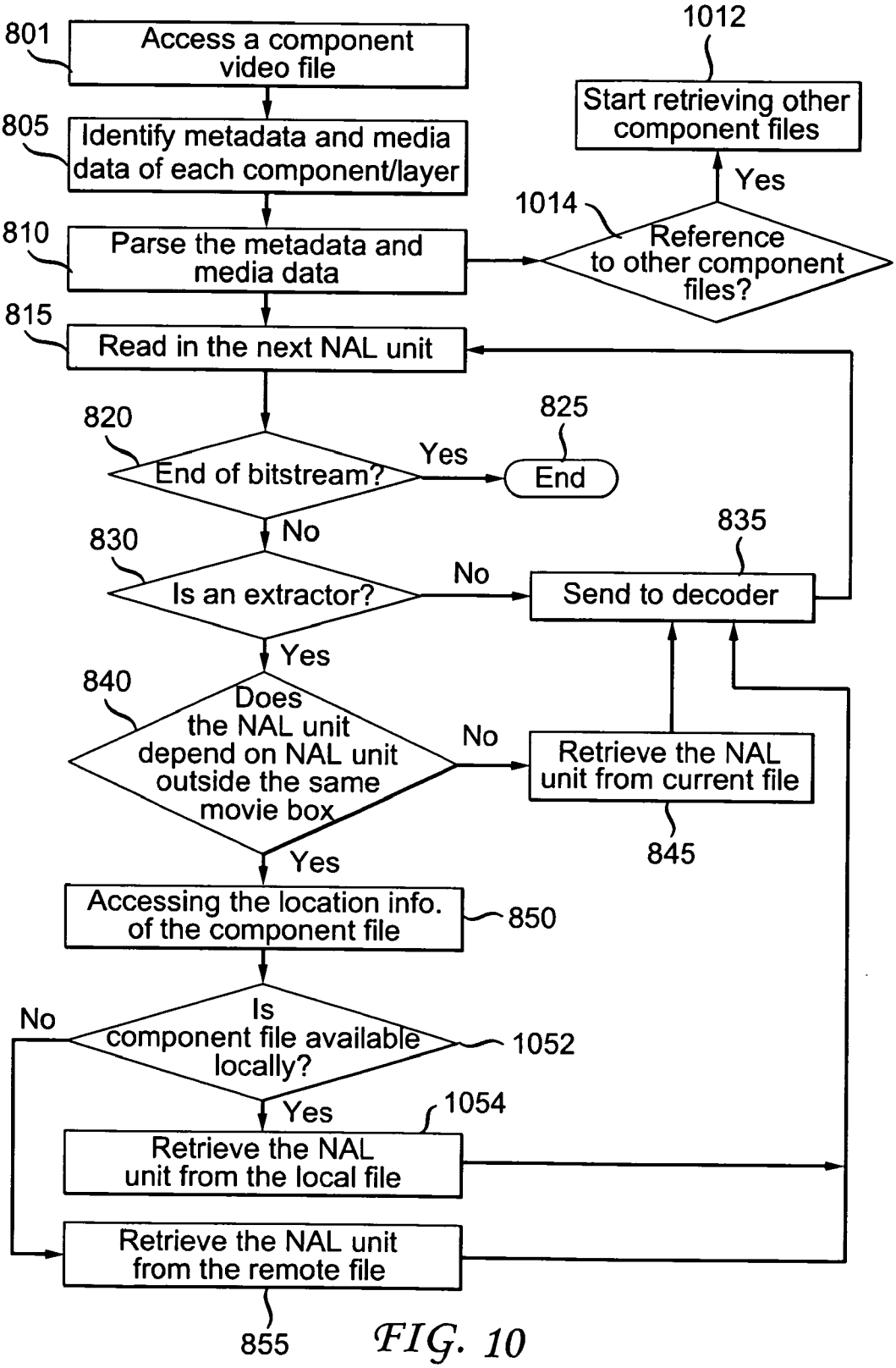
801
Access a component video file
805
Identify metadata and media data of each component/layer
810
Parse the metadata and media data
815
Read in the next NAL unit
820
End of bitstream?
Yes
825
End
No
830
Is an extractor?
No
835
Send to decoder
Yes
840
Does the NAL unit depend on NAL unit outside the same movie box
No
Retrieve the NAL unit from current file
845
Yes
Accessing the location info. of the component file
850
No
Is component file available locally?
1052
Yes
1054
Retrieve the NAL unit from the local file
Retrieve the NAL unit from the remote file
855
1012
Start retrieving other component files
Yes
1014
Reference to other component files?

FIG. 10

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2011/040168

A. CLASSIFICATION OF SUBJECT MATTER
INV. H04N21/84 H04N21/85 H04N7/24
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
H04N

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	GRÜNEBERG K ET AL: "Deliverable D3.2 MVC/SVC storage format", , no. Project No: FP7-ICT-214063 29 January 2009 (2009-01-29), pages 1-34, XP002599508, Retrieved from the Internet: URL:http://www.ist-sea.eu/Public/SEA_D3.2_HHI_FF_20090129.pd [retrieved on 2010-09-01] section 2.2.2 and 2.2.6 figures 2-4 - 2-10 ----- -/--	1-37

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
25 August 2011	31/08/2011

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Arpaci, Mutlu

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2011/040168

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X,P	ZHENYU WU ET AL: "Some extensions to ISO Base Media File Format and MPEG-2 Transport Stream to support multi-component media content HTTP Streaming", 93. MPEG MEETING; 26-7-2010 - 30-7-2010; GENEVA; (MOTION PICTURE EXPERT GROUP OR ISO/IEC JTC1/SC29/WG11),, no. M17894, 22 July 2010 (2010-07-22), XP030046484, section 1 section 2.1	1-37
A	DAVID SINGER: "Editor's draft of the Part12 file format amendment", 86. MPEG MEETING; 13-10-2008 - 17-10-2008; BUSAN; (MOTION PICTURE EXPERT GROUP OR ISO/IEC JTC1/SC29/WG11),, no. M15812, 20 October 2008 (2008-10-20), XP030044409, section 8.7.2	1-37
A	ANONYMOUS: "Text of ISO/IEC 14496-15/FDAM2 SVC File Format Extension", 83. MPEG MEETING;14-1-2008 - 18-1-2008; ANTALYA; (MOTION PICTURE EXPERT GROUP OR ISO/IEC JTC1/SC29/WG11),, no. N9682, 12 March 2008 (2008-03-12), XP030016176, page 1, first paragraph	1-37
A	YE-KUI WANG ET AL: "Comments to the MVC file format draft", 88. MPEG MEETING; 20-4-2009 - 24-4-2009; MAUI; (MOTION PICTURE EXPERT GROUP OR ISO/IEC JTC1/SC29/WG11),, no. M16444, 17 April 2009 (2009-04-17), XP030045041, page 1, first paragraph	1-37
A	AMON P ET AL: "File Format for Scalable Video Coding", IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, IEEE SERVICE CENTER, PISCATAWAY, NJ, US, vol. 17, no. 9, 1 September 2007 (2007-09-01), pages 1174-1185, XP011193013, ISSN: 1051-8215, DOI: 10.1109/TCSVT.2007.905521 abstract	1-37