



(12) 发明专利

(10) 授权公告号 CN 101427571 B

(45) 授权公告日 2010. 10. 20

(21) 申请号 200380105014. X

G06F 15/16(2006. 01)

(22) 申请日 2003. 11. 29

(56) 对比文件

(30) 优先权数据

10/309, 571 2002. 12. 04 US

W0 02/058388 A1, 2002. 07. 25, 全文.

US 2003/0061369 A1, 2003. 03. 27, 全文.

US 6381273 B1, 2002. 04. 30, 全文.

(85) PCT申请进入国家阶段日

2005. 06. 03

US 2003/0110297 A1, 2003. 06. 12, 说明书第

0028 段第 1-10 行.

(86) PCT申请的申请数据

PCT/US2003/038138 2003. 11. 29

审查员 康凯

(87) PCT申请的公布数据

W02004/051424 EN 2004. 06. 17

(73) 专利权人 国际商业机器公司

地址 美国纽约

(72) 发明人 威廉·拉肯

(74) 专利代理机构 中国国际贸易促进委员会专

利商标事务所 11038

代理人 付建军

(51) Int. Cl.

H04N 7/12(2006. 01)

G06F 7/00(2006. 01)

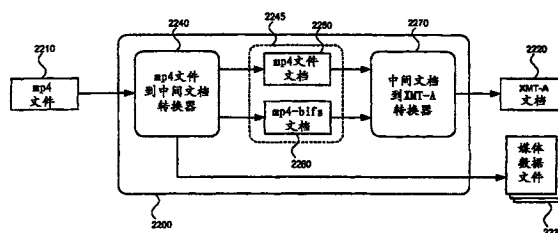
权利要求书 2 页 说明书 97 页 附图 44 页

(54) 发明名称

从 MPEG-4 中间格式创建 MPEG-4 文本表示的方法

(57) 摘要

一种用于将二进制 MPEG-4(mp4) 二进制文件 (2210) 转换为可扩展 MPEG-4 文本 (XMT) 文件 (2220) 的方法、系统和计算机程序产品 (2200)。本发明包括生成至少一个表示 mp4 二进制文件 (2210) 的中间结构化文档 (2245)。一种转换器被配置为输入中间文档 (224) 并生成 XMT 结构化文档 (2220)。然后使用 XMT 串行器 (2270) 基于 XMT 结构化文档 (2245) 创建 XMT 文件 (2220)。



1. 一种用于将 MPEG-4 即 mp4 二进制文件转换为可扩展 MPEG-4 文本即 XMT 文件的方法,该方法包括:

基于 mp4 二进制文件创建临时数据结构;

使用该临时数据结构生成至少一个表示 mp4 二进制文件的组织和结构、表示 mp4 二进制文件的对象描述符流以及表示 mp4 二进制文件的场景描述符流的中间结构化文档;

将至少一个中间结构化文档转换为 XMT 结构化文档;以及

基于 XMT 结构化文档创建 XMT 文件。

2. 根据权利要求 1 所述的方法,其中,转换中间结构化文档的过程包括生成 mp4-file 结构化文档和表示场景描述流的 mp4-bifs 结构化文档。

3. 根据权利要求 2 所述的方法,其中,mp4-file 结构化文档包括表示 moov 原子和一个或多个 mdat 原子的数据结构,moov 原子包括 iods 原子和一个或多个 trak 原子。

4. 根据权利要求 3 所述的方法,进一步包括基于表示 mdat 原子和 trak 原子的数据结构生成外部媒体文件。

5. 根据权利要求 2 所述的方法,其中,mp4-file 结构化文档基于可扩展标记语言。

6. 根据权利要求 2 所述的方法,其中,mp4-file 结构化文档包括 moov 结构化元素和一个或多个 mdat 结构化元素,moov 结构化元素表示 moov 原子,每一个 mdat 结构化元素都表示由中间结构化文档表示的 mdat 原子中的一个。

7. 根据权利要求 6 所述的方法,其中,moov 结构化元素包括表示中间结构化文档的对应的部分的 mp4fiods 元素和一个或多个 trak 元素。

8. 根据权利要求 7 所述的方法,其中,mp4-bifs 结构化文档是与 mp4-file 结构化文档分开的不同的文档。

9. 根据权利要求 2 所述的方法,其中,mp4-file 结构化文档包括表示对象描述符流的元素。

10. 根据权利要求 2 所述的方法,其中,mp4-bifs 结构化文档包括表示场景描述流的元素。

11. 根据权利要求 1 所述的方法,其中,生成中间结构化文档的过程包括:

生成表示 mp4 二进制文件的数据结构集;以及

将数据结构集转换为中间结构化文档。

12. 一种用于将 MPEG-4 即 mp4 二进制文件转换为可扩展 MPEG-4 文本即 XMT 文件的系统,该系统包括:

二进制文件解码器,该解码器被配置为输入 mp4 二进制文件,并使用基于 mp4 二进制文件创建的临时数据结构生成表示 mp4 二进制文件的组织和结构、表示 mp4 二进制文件的对象描述符流以及表示 mp4 二进制文件的场景描述符流的至少一个中间结构化文档,

转换器,该转换器被配置为输入中间结构化文档,并生成 XMT 结构化文档;以及

XMT 串行器,该串行器被配置为输 XMT 结构化文档,并生成 XMT 文件。

13. 根据权利要求 12 所述的系统,其中,至少一个中间结构化文档包括 mp4-file 结构化文档和表示场景描述流的 mp4-bifs 结构化文档,mp4-file 结构化文档包括基于数据结构集的一个或多个 mdat 元素、moov 元素以及 mp4fiods 元素。

14. 根据权利要求 13 所述的系统,其中,mp4-file 结构化文档包括表示 moov 原子和一

个或多个 mdat 原子的数据结构。

15. 根据权利要求 14 所述的系统,其中,表示 moov 原子的数据结构包括表示 iods 原子和一个或多个 trak 原子的数据结构。

16. 根据权利要求 15 所述的系统,进一步包括基于表示 mdat 原子和 trak 原子的数据结构的一个或多个外部媒体文件。

17. 根据权利要求 13 所述的系统,其中,mp4-file 结构化文档表示对象描述符流。

18. 根据权利要求 13 所述的系统,其中,mp4-bifs 结构化文档是与 mp4-file 结构化文档分开的不同的结构化文档。

从 MPEG-4 中间格式创建 MPEG-4 文本表示的方法

技术领域

[0001] 一般来说,本发明涉及多媒体信息的数据表示,具体来说,涉及将被称为“MPEG-4 中间格式”的一种形式的多媒体信息转换为被称为“MPEG-4 文本表示”的另一种形式的多媒体信息表示。

背景技术

[0002] 计算机通常被用来呈现各种数字媒体,包括图像、音频采样(声音),视频媒体,以及文本和几何形状。这些媒体类型中的每一种类型都可以分别地呈现,或者许多这样的媒体元素可以以所谓的复合多媒体内容一起呈现。

[0003] 创建和分发复合多媒体内容的能力对于传播基于各种媒体类型的信息非常重要。此外,已经创建了表示复合多媒体内容的标准化手段,以使许多作者创建可以在各种计算机平台上(如个人计算机、机顶盒及其他设备)再现的内容。

[0004] 运动图像专家组(MPEG)开发的复合多媒体内容的两个著名的标准化格式是可扩展 MPEG-4 文本(XMT)格式和二进制编码 MPEG-4(mp4)格式。XMT 格式最适合用于创作复合多媒体内容,而 mp4 格式最适合用于复合多媒体内容的压缩存储和传输。如此,应该有效地将 XMT 格式的内容转换为 mp4 格式的内容。

发明内容

[0005] 如下面详细描述,本发明是用于将二进制编码的 MPEG-4(mp4)格式转换为可扩展 MPEG-4 文本(XMT)格式的方法、系统和设备。本发明利用包括相对少量的软件并只需要少量的资源实现从 mp4 格式到 XMT 格式的复合多媒体内容转换的有效方法。

[0006] 如此,本发明的一个方面涉及用于将 MPEG-4(mp4)二进制文件转换为可扩展 MPEG-4 文本(XMT)文件的方法。该方法包括生成表示 mp4 二进制文件的数据结构集的生成操作。第一个转换操作将数据结构集转换为至少一个中间结构化文档,而第二个转换操作将中间结构化文档转换为 XMT 结构化文档。然后,创建操作基于 XMT 结构化文档创建 XMT 文件。

[0007] 本发明的另一个方面是用于将 MPEG-4(mp4)二进制文件转换为可扩展 MPEG-4 文本(XMT)文件的系统。该系统包括二进制文件解码器,该解码器被配置为输入 mp4 二进制文件,并生成表示 mp4 二进制文件的数据结构集。第一个转换器被配置为输入数据结构集,并创建至少一个中间结构化文档。第二个转换器被配置为输入中间结构化文档,并生成 XMT 结构化文档。XMT 串行器被配置为输入 XMT 结构化文档,并生成 XMT 文件。

[0008] 本发明的又一个方面是用于将 MPEG-4(mp4)二进制文件转换为可扩展 MPEG-4 文本(XMT)文件的包含在有形介质中的计算机程序产品。计算机程序执行生成表示 mp4 二进制文件的数据结构集,将数据结构集转换为至少一个中间结构化文档,将中间结构化文档转换为 XMT 结构化文档,基于 XMT 结构化文档创建 XMT 文件的操作。

[0009] 从下面比较具体的参考附图对本发明的各种实施例的描述中,本发明的前述的及

其他特点,实用程序和优点将变得显而易见。

附图说明

- [0010] 图 1A 显示了本发明的一个实施例利用的典型的 XMT-A 文档。
- [0011] 图 1B 显示了典型的 XMT-A 初始对象描述符。
- [0012] 图 2A 显示了典型的 XMT-A par 元素。
- [0013] 图 2B 显示了典型的 XMT-A odsm 命令元素。
- [0014] 图 3A 显示了典型的 XMT-A Insert 命令。
- [0015] 图 3B 显示了典型的 XMT-A Delete 命令。
- [0016] 图 3C 显示了典型的 XMT-A Replace 命令。
- [0017] 图 4 显示了典型的 XMT-A BIFS Node 元素。
- [0018] 图 5A 显示了典型的 XMT-A BIFS Node。
- [0019] 图 5B 显示了典型的重复使用的 XMT-A BIFS Node。
- [0020] 图 6A 显示了典型的 XMT-A ObjectDescriptor。
- [0021] 图 6B 显示了典型的 XMT-A ES_Descriptor。
- [0022] 图 6C 显示了典型的 sdsm 的 DecoderSpecificInfo (BIFS)。
- [0023] 图 7A 显示了本发明的一个实施例生成的典型的 mp4 二进制文件。
- [0024] 图 7B 显示了典型的 mdat 原子。图 7C 显示了典型的块。图 7D 显示了典型的 moov 原子。
- [0025] 图 8A 显示了典型的 mp4 文件 iods 原子。
- [0026] 图 8B 显示了典型的 Mp4flnitObjectDescr。
- [0027] 图 8C 显示了典型的 ES_ID_Inc。
- [0028] 图 9A 显示了典型的 trak 原子。
- [0029] 图 9B 显示了典型的采样表原子。
- [0030] 图 10A 显示了典型的二进制 ES 描述符。
- [0031] 图 10B 显示了典型的解码器配置描述符。
- [0032] 图 10C 显示了典型的解码器特定信息描述符。
- [0033] 图 10D 显示了典型的二进制 SL 配置描述符。
- [0034] 图 11A 显示了典型的 sdsm 二进制块。
- [0035] 图 11B 显示了典型的 sdsm 命令帧。
- [0036] 图 12A 显示了典型的 BIFS 插入命令。
- [0037] 图 12B 显示了典型的 BIFS 删除命令。
- [0038] 图 12C 显示了典型的 BIFS 替换命令。
- [0039] 图 12D 显示了典型的 BIFS scene 替换命令。
- [0040] 图 13A 显示了典型的 Node 插入命令。
- [0041] 图 13B 显示了典型的 IndexedValue 插入命令。
- [0042] 图 13C 显示了典型的 Route 插入命令。
- [0043] 图 14A 显示了典型的 Node 删除命令。
- [0044] 图 14B 显示了典型的 IndexedValue 删除命令。

- [0045] 图 14C 显示了典型的 Route 删除命令。
- [0046] 图 15A 显示了典型的 Node 替换命令。
- [0047] 图 15B 显示了典型的 Field 替换命令。
- [0048] 图 15C 显示了典型的 IndexedValue 替换命令。
- [0049] 图 15D 显示了典型的 Route 替换命令。
- [0050] 图 16 显示了典型的 BIFS Scene。
- [0051] 图 17A 显示了典型的 SFNode(重复使用的)。
- [0052] 图 17B 显示了典型的 SFNode(屏蔽节点)。
- [0053] 图 17C 显示了典型的 SFNode(列表节点)。
- [0054] 图 17D 显示了典型的 MFField(列表形式)。
- [0055] 图 17E 显示了典型的 MFField(矢量形式)。
- [0056] 图 18A 显示了典型的 Routes(列表形式)。
- [0057] 图 18B 显示了典型的 Routes(矢量形式)。
- [0058] 图 18C 显示了典型的 Route。
- [0059] 图 19A 显示了典型的 odsm 二进制块。
- [0060] 图 19B 显示了典型的 odsm 二进制示例。
- [0061] 图 20A 显示了典型的 ObjectDescriptorUpdate 命令。
- [0062] 图 20B 显示了典型的 ObjectDescriptorRemove 命令。
- [0063] 图 21A 显示了典型的二进制对象描述符。
- [0064] 图 21B 显示了典型的二进制 EsIdRef 描述符。
- [0065] 图 22 显示了本发明设想的典型的 MPEG-4 中间到 XMT-A 文件转换器。
- [0066] 图 23A 显示了典型的 mp4file 文档。
- [0067] 图 23B 显示了典型的 mp4fiods 元素。
- [0068] 图 24A 显示了典型的 mdat 元素。
- [0069] 图 24B 显示了典型的 sdsm 元素。
- [0070] 图 24C 显示了典型的 odsm 元素。
- [0071] 图 24D 显示了典型的 mediaFile 元素。
- [0072] 图 25A 显示了典型的 odsmChunk 元素。
- [0073] 图 25B 显示了典型的 odsmSample 元素。
- [0074] 图 25C 显示了 odsm-command 元素。
- [0075] 图 26A 显示了典型的 trak 元素。
- [0076] 图 26B 显示了典型的 stbl 元素。
- [0077] 图 27 显示了典型的 ES Descr。
- [0078] 图 28A 显示了典型的 mp4bifs 文档。
- [0079] 图 28B 显示了典型的 mp4bifs commandFrame 元素。
- [0080] 图 29A 显示了典型的 mp4bifs bifsCommand 元素。
- [0081] 图 29B 显示了典型的 mp4bifs ReplaceScene 元素。
- [0082] 图 30A 显示了典型的 mp4bifs original Node 元素。
- [0083] 图 30B 显示了典型的 mp4bifs Conditional Node 元素。

- [0084] 图 30C 显示了典型的 mp4bifs Reused Node 元素。
- [0085] 图 31A 显示了典型的 QtInfo 结构。
- [0086] 图 31B 显示了典型的 Mp4fInitObjectDescr 元素。
- [0087] 图 31C 显示了典型的 mdat 结构。
- [0088] 图 31D 显示了典型的 MoovHeader 元素。
- [0089] 图 32A 显示了典型的 track 结构。
- [0090] 图 32B 显示了典型的 TrackHeader 结构。
- [0091] 图 32C 显示了典型的 TrackMedia 结构。
- [0092] 图 32D 显示了典型的 TrackEdit 结构。
- [0093] 图 32E 显示了典型的 EditList 结构。
- [0094] 图 32F 显示了典型的 EditSegment 结构。
- [0095] 图 32G 显示了典型的 TrackReference 结构。
- [0096] 图 32H 显示了典型的 Mpod 结构。
- [0097] 图 32I 显示了典型的 UserData 结构。
- [0098] 图 32J 显示了典型的 cpri 结构。
- [0099] 图 31D 显示了典型的 MoovHeader 元素。
- [0100] 图 33A 显示了典型的 MediaHeader 结构。
- [0101] 图 33B 显示了典型的 Handler 结构。
- [0102] 图 33C 显示了典型的 MediaInfo 结构。
- [0103] 图 33D 显示了典型的 MediaDataInfo 结构。
- [0104] 图 33E 显示了典型的数据表结构。
- [0105] 图 33F 显示了典型的数据表结构。
- [0106] 图 34A 显示了典型的 SampleTables 结构。
- [0107] 图 34B 显示了典型的采样表结构。
- [0108] 图 34C 显示了典型的 SampleSizeTable 结构。
- [0109] 图 34D 显示了典型的 SampleToChunk 结构。
- [0110] 图 34E 显示了典型的 TimeToSample 结构。
- [0111] 图 34F 显示了典型的 SampleDescription 结构。
- [0112] 图 35 显示了典型的 mp4 文件解释流程图。
- [0113] 图 36 显示了典型的 mdat 原子处理流程图。
- [0114] 图 37 显示了典型的 moov 原子处理流程图。
- [0115] 图 38 显示了典型的 trak 原子处理流程图。
- [0116] 图 39 显示了典型的 trak 媒体原子处理流程图。
- [0117] 图 40 显示了典型的媒体信息原子处理流程图。
- [0118] 图 41 显示了典型的采样表原子处理流程图。
- [0119] 图 42 显示了构成以 QtInfo 数据结构表示的媒体数据原子的 xml 表示的典型流程图。
- [0120] 图 43 显示了典型的媒体数据处理流程图。
- [0121] 图 44 显示了典型的查找流式类型的流程图。

- [0122] 图 45 显示了典型的查找解码器配置描述符的流程图。
- [0123] 图 46 显示了典型的获取标记大小流程图。
- [0124] 图 47 显示了从 odsm 数据流创建 xml 表示的典型流程图。
- [0125] 图 48 显示了典型的解码 odsm 块流程图。
- [0126] 图 49 显示了典型的解码 OdUpdate 流程图。
- [0127] 图 50 显示了典型的 decodeEsIdRef 过程流程图。
- [0128] 图 51 显示了创建媒体文件和表示 mdat 原子的典型流程图。
- [0129] 图 52 显示了典型的 SdsmDataToXml 过程流程图。图 53 显示了典型的解码 odsm 块过程流程图。
- [0130] 图 54 显示了典型的解码插入命令过程流程图。
- [0131] 图 55 显示了典型的解码删除命令过程流程图。
- [0132] 图 56 显示了典型的解码替换命令过程流程图。
- [0133] 图 57 显示了典型的解码 SFNode 过程流程图。
- [0134] 图 58 显示了创建 moov 原子的 xml 表示的典型流程图。
- [0135] 图 59 显示了创建 XMT-A 文档的典型流程图。
- [0136] 图 60 显示了创建 XMT-A 标头的典型流程图。
- [0137] 图 61 显示了处理 EsIdInc 元素的典型流程图。
- [0138] 图 62A 显示了典型的 streamType 转换表。
- [0139] 图 62B 显示了典型的 objectType 到 objectTypeIndication 表。
- [0140] 图 63 显示了创建表示 Object Descriptor 命令的“par”元素的典型流程图。
- [0141] 图 64 显示了创建表示 commandFrame 命令的“par”元素的典型流程图。
- [0142] 图 65 显示了创建节点编号值和 nodeId 名称的表的典型流程图。
- [0143] 标题表
- [0144] MPEG-4 文本表示 1.0
- [0145] MPEG-4 中间格式文件 2.0
- [0146] 场景描述流 (sds) 3.0
- [0147] 对象描述符流 (odsm) 4.0
- [0148] mp4-file 文档 5.0
- [0149] mp4-bifs 文档 6.0
- [0150] mp4 到 xmta 转换器 7.0

具体实施方式

[0151] 本发明是用于将可扩展 MPEG-4 文本 (XMT) 格式 (这里也称为“XMT-A 文档”和“MPEG-4 文本表示”) 转换为二进制编码的 MPEG-4 (mp4) 格式 (也称为“MPEG-4 中间二进制格式”) 的方法、系统和计算机程序。本发明利用了实现从 XMT-A 到 mp4 的转换的新颖的方法, 该方法需要相对少量的软件并只需要少量的资源。这里将参考图 1-65 对本发明进行描述。

[0152] 1.0MPEG-4 文本表示

[0153] “MPEG-4 文本表示”包括表示多媒体内容的结构的“文本文件”。多媒体内容包括声

音、静止图像、视频剪辑及其他元素的同步组合或序列。文本文件是包含字母、数字、以及标点的二进制代码的序列的电子数据结构。文本文件通常可以使用通常被称为“文本编辑器”的软件来进行解释。有许多文本编辑器的例子,包括基于**Windows ®**操作系统的计算机的被称为“NotePad. exe”软件,以及使用统称为 UNIX 的各种操作系统的计算机的“vi”。Windows 是位于华盛顿州 Redmond 的微软公司的注册商标。包括“MPEG-4 文本表示”的文本文件类型被称为“XMT-A”文件。

[0154] 在文本文件范围内,XMT-A 文件是可扩展标记语言(XML)文件的示例。XML 文件是基于 World Wide Web Consortium 指定的原理的结构化文档(请参见 <http://www.w3.org/TR/2000/REC-XML-20001006>)。如国际标准化组织和国际电工委员会所说明的,XMT-A 文件代表了 XML 文件的特定实施例(请参见 TS0/IfCdocument 14496-1:2000 Amd. 2, October 2000, 网址是:http://mpeg.telecomitalialab.com/working_documents.htm and International Organization for Standardization(ISO),1, rue deVarembe, Case postal 56, CH-12" Geneva 20, Switzerland)。XMT-A 的每一部分的完整的说明非常长。如此,下面对 XMT-A 文件的描述限于描述本发明所需要的那部分规范。有关 XMT-A 文件结构的完整的说明,读者应该查阅引用的 XMT 规范文档。

[0155] 类似于任何 XML 文件,XMT-A 文件包括“元素”的层次集合。每一个元素都可以包含被称为“子元素”的从属元素。此外,每一个元素还可以拥有被称为“属性”的数据值集。每一个属性都有名称和值。特定属性名称和任何特定元素所拥有的可能的子元素取决于元素的类型。对于每一个属性值的解释取决于相应的属性名称和拥有该属性的元素。

[0156] 如图 1A 所示,XMT-A 文件 100 包括两个主要部分,Header 元素 110 和 Body 元素 120。Header 元素 110 包含被定义为 InitialObjectDescriptor 元素 130 的单个子元素。Body 元素 120 包含一个或多个“par”元素 140 作为子元素。

[0157] InitialObjectDescriptor 130 有一个属性,Obj ectDescriptorID(ODID),其值是字符串。如图 1B 所示,此元素具有两个子元素,Profiles 元素 150 和 Descr 元素 160。Profiles 元素 150 没有子元素。Profiles 元素 150 拥有多个属性,包括“includeInclineProfileLevelFlag”、“sceneProfileLevelIndication”、“ODProfileLevelIndication”、“audioProfileLevelIndication”、“visualProfileLevelIndication”和“graphicsProfileLevelIndication”。

[0158] Descr 元素 160 可以具有多种类型的子元素。对本发明必不可少的唯一类型是单个“esDescr”元素 170。esDescr 元素 170 可以拥有一个或多个“ES_Descriptor”子元素 180,190。ES_Descriptor 元素指定了“基本流”的某些属性,这是 MPEG-4 文档中定义的概念。下面指出了 ES_Descriptor 元素的结构。

[0159] 从属于 InitialObjectDescriptor 元素 130 的 esDescr 元素 110 可以拥有一个或两个 ES_Descriptor 元素 180,190。在每一种情况下,都应该有被定义为“sdsM”或“场景描述流”的基本流的 ES_Descriptor 180。此外,还可以有被定义为“odsM”或“对象描述符流”的基本流的第二个 ES_Descriptor 190。odsM 的 ES_Descriptor 元素 190 只对于依赖于音频数据、视频数据,或 sdsM 内未指定的其他类型的媒体数据的 XMT-A 文件才需要。

[0160] 如图 2A 所示,每一个 par 元素 140,200 都包含一个或多个“par-child”元素 210。“par-child”元素可以是另一个 par 元素、odsM 命令或 bifs 命令。每一个 par 元素还包含

名称为“begin”的属性。begin 属性的值指定将要执行 par 元素内的 odsm 或 bifs 命令的时间。par 元素的 begin 属性确定的时间值是相对于由任何父元素暗示的时间值计算的，Body 元素 120 暗示开始时间为 0。

[0161] par-child 元素 210 可以包含两种类型的 odsm 命令元素的实例，如图 2B 所示。这些实例包括 ObjectDescriptorUpdate 元素 220 和 ObjectDescriptorRemove 元素 250。ObjectDescriptorUpdate 元素 220 包含单个 OD 子元素 230，而 OD 元素 230 包含单个 ObjectDescriptor 子元素 240。下面将比较详细地描述 ObjectDescriptor 元素 240。ObjectDescriptorRemove 元素 250 具有一个属性，没有子元素。ObjectDescriptorRemove 元素 250 的属性叫做“ODID”。

[0162] par-child 元素 210 可以包含三种类型的 bifs 命令元素的实例，如图 3 所示。这些实例包括 Insert 元素 300、Delete 元素 310 和 Replace 元素 320。如图 3A 所示，Insert 元素 300 可以具有“xmtaBifsNode”子元素 330 或“ROUTE”子元素 340。Delete 元素 310 没有子元素。Replace 元素 320 可以具有“xmtaBifsNode”子元素 350，“ROUTE”子元素 360 或“Scene”子元素 370。Scene 元素具有“xmtaTopNode”子元素 380。Scene 元素还可以具有一个或多个 ROUTE 子元素 390。

[0163] ROUTE 元素 340, 390 没有子元素。ROUTE 元素 340, 390 的属性包括“fromNode”、“fromField”、“toNode”和“toField”。

[0164] 术语“xmtaBifsNode 元素”330 代表大致 100 个定义的 BIFSNODE 元素中的任何一个。这些元素种的每一个元素都具有图 4 所示的一般结构 400。每一个 xmtaBifsNode 元素 400 都代表 BIFSNODE，这是 MPEG-4 系统规范 ISO-IfC 文档 ISO/IfC14496-1: 2001, 2001 年 8 月第 9 章中定义的二进制数据结构。有关此文档的信息位于 <http://mpeg.telecomitalialab.com/documents.htm> 和 International Organization for Standardization (ISO), 1, rue de Varembe, Case postale 56, CH-12 " Geneva 20, Switzerland。每一个 xmtaBifsNode 元素 400 的元素标记都基于 MPEG-4 系统规范中定义的对应的 NodeName。某些类型的 xmtaBifsNode 元素可以具有基于对应的 BIFS Node 的某些属性的从属（子）元素。这些子元素被称作 nodeField 元素 410。每一个 nodeField 元素都可以具有一个或多个从属元素，而这些从属元素又可以进一步地包括 xmtaBifsNode 元素 420。此方案可以递归地重复，以描述 BIFS 节点的层次树。对此层次结构的深度没有限制。

[0165] 每一个 BIFS 节点每一个都具有许多叫做“字段”的属性。这些字段中的每一个字段都具有定义的字段名称（字符串）和字段数据类型（布尔、整数、浮点等等）。其中一个字段数据类型是“Node”。Node 之外的所有字段数据类型都由 xmtaBifsNode 元素 400 的名称类似的属性代表。具有类型“节点”的每一个字段都由 xmtaBifsNode 元素 400 的名称类似的子元素 410 代表。xmtaBifsNode 元素 400 的每一个子元素 410 都可以具有一个或多个 xmtaBifsNode 元素 420 作为子元素（xmtaBifsNode 父元素 400 的孙子元素）。

[0166] XMT-A BIFS Node 的 XML 表示如图 5 所示。每一个 XMT-A BIFS Node 元素都由 NodeName 标记 500, 570 来标识，这些标记唯一地标识 100 个以上的可能的 XMT-A BIFS 节点类型中的某一种类型。每一个节点元素都可以是原始节点元素 500 或重新使用的节点元素 570。在原始节点元素 500 的情况下，可以使用可选属性“DEF”510 来提供特定节点的唯一

的字母数字描述。如果提供了此属性,则该节点被分为“可重复使用的”类。

[0167] 原始XMT-A BIFS节点元素还拥有一组字段属性520,为类型NodeName的节点定义的每一个属性字段都具有一个字段属性,并具有“node”或“buffer”之外的节点数据类型。这些属性被标识为图5A中的“field0”、“field2”、“field3”和“field5”。这些属性中的每一个属性的实际名称都由在MPEG-4系统规范为“NodeName”类型的节点定义的对应的属性字段名称确定。指定给这些属性中的每一个属性的值都必须代表具有在MPEG-4系统规范中定义的节点数据类型的数据值(布尔、整数、浮点等等)。

[0168] 此外,原始XMT-A BIFS节点元素500可以具有一个或多个字段值子元素530,540,并具有对应于属性字段(具有数据类型“node”或“buffer”)的字段名的元素标记。每一个这样的字段值元素都具有开始标记530和结束标记540。这样的字段值元素530,540的示例由图5A中的元素标记<field1>...</field1>和<field4>...</field4>代表。

[0169] 在具有数据类型“node”的属性字段的情况下,字段值元素可以包含对应于BIFS-Node元素550的一个或多个子元素。这样的BIFS-Node子元素的示例由元素标记<NodeName1.../>、<NodeName2.../>和<NodeName3.../>代表。

[0170] 在具有数据类型“buffer”的属性字段的情况下,字段值元素可以包含对应于BIFS命令元素300、310、320的一个或多个子元素。

[0171] 如果XMT-A BIFS Node元素包括任何字段值子元素,根据标准XML原则,Node元素将由</NodeName>结束标记560结束。

[0172] XMT-A BIFS节点元素的上述定义递归地应用到每一个从属BIFS节点元素(<NodeName1>,等等),从而允许创建节点的层次树。对于此XMT-A BIFS节点元素的树的深度没有限制。

[0173] 在重复使用的节点570情况下,节点元素只具有一个属性,而没有子元素。唯一的属性是“USE”属性580,其值590是节点ID字符串。作为USE属性的值提供的节点ID字符串必须匹配作为具有相同NodeName的原始节点元素500的DEF属性510指定的节点ID字符串。

[0174] 术语“xmtaTopNode”代表允许作为Scene元素的子元素的xmtaBifsNode元素的定义的子集中的一个元素。

[0175] 如图6A所示,ObjectDescriptor元素240,600(ObjectDescriptorUpdate元素220的孙子元素)类似于上文所描述的InitialObjectDescriptor元素130。与InitialObjectDescriptor元素130不同,ObjectDescriptor元素240,600没有Profiles子元素150。类似于InitialObjectDescriptor元素130,ObjectDescriptor元素240,600具有“ObjectDescriptorID”(ODID)属性606。典型的ObjectDescriptor元素240,600具有单个Descr子元素610,Descr元素610具有单个esDescr子元素620,esDescr元素620具有单个ES_Descriptor子元素630。Descr元素610,esDescr元素620,以及ES_Descriptor元素630类似于InitialObjectDescriptor元素130的对应的子元素160、170、180、190。

[0176] ES_Descriptor元素180、190、630可以包含在ObjectDescriptor元素600或InitialObjectDescriptor元素130内。不论是哪一种情况,ES_pescriptor元素180、190、630都具有如图6B所示的结构640。ES_Descriptor元素640的“ES_ID”属性636的值是字母数字字符串,该字符串对于每一个流是唯一的。ES_Descriptor 640

元素始终具有 decConfigDescr 子元素 646 和 slConfigDescr 子元素 660。如果 ES_Descriptor 元素 630、640 从属于 ObjectDescriptor 元素 600, 则 ES_Descriptor 元素 630、640 还具有 StreamSource 子元素 670。如果 ES_Descriptor 元素 180、190、640 从属于 InitialObjectDescriptor 元素 130, 则 ES_Descriptor 元素 180、190、640 没有 StreamSource 子元素 670。

[0177] decConfigDescr 元素 646 具有 DecoderConfigDescriptor 子元素 650。DecoderConfigDescriptor 元素 650 具有多个属性, 包括“streamType”和“objectTypeIndication”, 表示父 ES_Descriptor 元素 640 是否代表音频、视频、sdsm、odsm 或其他类型的媒体。根据 streamType 和 objectTypeIndication 的值, DecoderConfigDescription 元素 650 还可以具有 decSpecificInfo 子元素 656。

[0178] 在 sdsm(场景描述流)的 ES_Descriptor 元素 180 的情况下, DecoderConfigDescriptor 650 元素具有 decSpecificInfo 子元素 656。如图 6C 所示, decSpecificInfo 680 元素具有 BIFSConfig 子元素 686。BIFSConfig 元素 686 拥有多个指定如何对 BIFS Nodes 进行编码的属性。BIFSConfig 元素 686 还拥有 commandStream 元素 690, 而 commandStream 元素 690 拥有“size”元素 696。

[0179] SlConfig 元素 660 具有 SlConfigDescriptor 子元素 666。SlConfigDescriptor 元素 666 具有一个名为“predefined”的属性, 没有子元素。“预定义”属性的值始终为“2”。

[0180] StreamSource 元素 670 具有一个属性“url”, 没有子元素。url 属性的值指定文件名或 Internet 地址(url, 统一资源定位器), 该地址指出包含音频数据、视频数据或定义特定流的实际声音、图像等等的其他数据的媒体数据文件的位置。StreamSource 元素 670 对于 sdsm(场景描述流)或 odsm(对象描述符流), 因为这些流都是由 XMT-A 文件确定的。

[0181] 2.0MPEG-4 中间格式文件

[0182] MPEG-4 中间格式文件是具有 MPEG-4 Systems specifications document ISO-IfC document ISO/IfC 14496-1:2001, August 2001 的第 13 章中定义的结构和组成的电子数据的一种形式。这种形式的电子数据结构是通常被称为“二进制文件”的示例, 因为它由二进制数据值的序列组成, 这些二进制数据值不局限于字母、数字和标点的表示。这就允许比诸如 XMT-A 文件之类的典型的文本文件所提供的更加紧凑的数据结构。具有由 MPEG-4 中间格式定义的结构的数据的电子数据的存储的形式被称作“mp4 二进制文件”。与 XMT-A 文件不同, mp4 二进制文件不能被大多数文本编辑软件解释。

[0183] MPEG-4 中间格式是从由 Apple Computers, Inc. 于 1996 年定义的 **QuickTime®** 文件格式派生而, 并可以从 <http://developer.apple.com/techpubs/quicktime/qtdevdocs/REF/refFileFormat96.htm> 和 <http://developer.apple.com/techpubs/quicktime/qtdevdocs/PDF/QTFileFormat.pdf> 下载。QuickTime 是 Apple Computer, Inc. 的注册商标。

[0184] > 由于其 QuickTime 继承性, MPEG-4 中间格式保留了许多从 QuickTime 规范派生的特征。这些特征包括“原子”作为数据结构的单元的概念。每一个原子都具有两个部分, 标头和主体。标头包含原子大小值, 该值指定构成原子的字节的数量, 包括标头。标头还包含指定原子的类型的 atomId。原子的正文主体包含原子所携带的数据。此数据可以包括从属原子。在其基本形式中, 原子具有由四个字节组成的原子大小值(无符号整数)和也包

括四个字节的 atomId(字符)。在 MPEG-4 规范中还定义了具有原子大小值和具有 4 个以上字节的 atomId 值的原子的扩展形式。

[0185] 如图 7A 所示, mp4 二进制文件 700 由一个或多个“mdat”原子 706 和一个“moov”原子 712 组成。moov 原子 712 可以在 mdat 原子 706 前面,也可以在其后面。如图 7B 所示,每一个 mdat 原子 718 都包括原子大小值 724,后面有四字节 atomId “mdat”730 和叫做“块”736 的数据块序列。如图 7C 所示,每一个块 742 都由媒体数据“样本”748 的序列组成。每一个样本 748 都指定了与单个媒体流的特定时间点关联的数据块。单个块内的所有样本都代表相同的媒体数据流。从 mdat 原子 700 的检查不一定可能标识单个样本 748 或块 736、742。每一个样本 748 和块 736、742 都可以使用存储在 mp4 二进制文件内某处的表来进行标识。

[0186] 如图 7D 所示, moov 原子 754 包括原子大小值 760, 后面有四字节 atomId “moov”766, 和包括“mvhd”(moov 标头)原子 772、“iods”(初始对象描述符)原子 778, 以及一个或多个“trak”原子 790 的多个从属原子。“moov”原子 712、754 包括每一个数据流的一个“trak”原子 790, 包括 sdsm(场景描述流)和 odsm(对象描述符流), 如果存在的话。“moov”原子 712、754 还可以包括可选的“udta”(用户数据)原子 784。“udta”原子 784 可以用来在 MP4 二进制文件中嵌入诸如版权消息之类的可选信息。

[0187] mvhd 原子 772 包括原子大小值, 后面有四字节 atomId “mvhd”和许多数据值, 包括时间和日期戳, 时间标度值, 以及文件持续时间值。mvhd 原子的原子大小的值始终为 108。时间和日期戳表示创建文件的时间。时间标度值指出用于表示文件的时间值的每秒钟的滴答声的数量。文件持续时间值指出呈现文件在中的材料所需要的总时间(采用时间标度值指定的时间单位)。

[0188] 如图 8A 所示, iods 原子 800 包括原子大小值 804, 后面有四字节 atomId “iods”808、8- 位版本值 812、24- 位标志值 816, 以及 Mp4fInitObj_Descr 数据结构 820。如图 8B 所示, Mp4fInitObj_Descr 数据结构 824 包括一字节 Mp4_IOD_TAG 值 828、序列数据块中的字节的数量 832、10- 位 ObjectDescriptorID 836、两个标志位 840、844, 四个预留的位 848 和五个配置文件级别指示值 852、856、860、864、868。配置文件级别指示值后面有一个或两个 MPEG-4ES_ID_Inc 数据结构 872。一个 ES_ID_Inc 结构指出对应于 sdsm(场景描述流)的 trak 原子 790 的 ES_ID 值。第二个 ES_ID_Inc 结构(如果存在的话)指出对应于 odsm(对象描述符流)的 trak 原子 790 的 ES_ID。第二个 ES_ID_Inc 结构只有在存在 odsm 的情况下才存在。如图 8C 所示, 每一个 ES_ID_Inc 数据结构都包括一字节 ES_ID_IncTag 值 880、序列数据中的字节的数量(始终为 4)和 32- 位 ES_ID 值 888。

[0189] 如图 9A 所示, 每一个 trak 原子 900 包括原子大小值 903, 后面有四字节 atomId “trak”906、“tkhd”(轨道标头)原子 910, 以及“mdia”(媒体)原子 912。在表示 odsm(对象描述符流)的 trak 原子的情况下, trak 原子还包括“tref”(轨道引用)原子 940。在具有延迟启动的轨道的情况下, 提供了“edts”(编辑列表)原子 945, 以指出何时开始轨道。mdia 原子 912 包括“mdhd”(媒体标头)原子 915、“hdlr”(处理程序)原子 918、“minf”(媒体信息)原子 920、“stbl”(样本表)原子 933, 以及媒体信息标头原子 936。标记“* mhd”代表多个媒体信息标头原子类型中的任何一个, 包括“nmhd”(对于 sdsm 和 odsm 轨道)、“smhd”(对于音频轨道)、“vmhd”(对于视频轨道)等等。

[0190] tkhd 原子 910、mdhd 原子 915,以及 hdlr 原子 918 包含许多数据值,包括 trackId 编号、时间和日期戳、媒体时间标度和媒体持续时间值。每一个轨道都具有其自己的时间标度,该标度可以不同于在 mvhd 原子 772 中指定的全局时间标度。

[0191] 如图 9B 所示,样本表原子 950 包括原子大小值 954,后面有四字节 atomId“stbl”957 和一系列样本表原子 960、963、966、970、974、978。各种样本表原子可以采取任何顺序。其中包括“stsc”(样本到块表)原子 960、“stts”(时间到样本表)原子 963、“stco”(块偏移表)原子 966、“stsz”(样本大小表)原子 970、可能的“stss”(同步样本表)原子 974,以及“stsd”(样本描述表)原子 978。这些样本表原子中的每一个都包含关联的 mdat 原子 706、718 中存储的二进制媒体数据 736、748 的数据描述属性。

[0192] 样本到块表原子(stsc 原子)960 包括原子大小值、四字节 atomId(“stsc”)、32-位无符号整数(numStscEntries),以及样本到块数据记录的序列。numStscEntries 的值指定了样本到块数据记录的序列中的条目的数量。每一个样本到块数据记录都包括三个 32-位无符号整数,它们指定了开始块编号、每个块的样本数量,以及样本描述索引。样本描述索引是对样本描述表 978 中的条目的索引。每个块的样本的数量指定了由开始块编号指定的块 736,以及下一个条目指定的开始块前面的所有随后的块中的样本 748 的数量。

[0193] 时间到样本表原子(stts 原子)963 包括原子大小值、四字节 atomId(“stts”)、32-位无符号整数(numSttsEntries),以及时间到样本数据记录的序列。numSttsEntries 的值指定了时间到样本数据记录的序列中的条目的数量。每一个时间到样本数据记录都包括两个 32-位无符号整数,这些整数指定样本计数和以轨道时间刻度单位表示的样本持续时间。样本计数值指定了具有对应的样本持续时间的连续的样本 748 的数量。

[0194] 块偏移表原子(stco 原子)966 包括原子大小值、四字节 atomId(“stco”)、32-位无符号整数(numStcoEntries),以及块偏移值的序列。numStcoEntries 的值指定了块偏移值的序列中的条目的数量。此序列中的每一个条目都包括一个 32 位无符号整数,该整数指定了 mp4 文件的开始和 mdat 原子 719 内的对应的块 736 的开始之间的字节的数量。

[0195] 样本大小表原子(stsz 原子)970 包括原子大小值、四字节 atomId(“stsz”)和 32-位无符号整数(iSampleSize),该整数指定了与此 trak 原子 900 关联的所有媒体数据样本 748 的大小。如果与此 trak 原子关联的媒体数据样本的大小不全部相等,则 iSampleSize 的值被指定为零,后面有 32 位无符号整数(numStszEntries)和样本大小值的序列。numStszEntries 的值指定了样本大小值的序列中的条目的数量。此序列中的每一个条目都包括一个 32 位无符号整数,该整数指定了与此 trak 原子 900 关联的媒体数据 718 内的对应的样本 748 中的字节的数量。

[0196] 块偏移表原子(stco 原子)966 包括原子大小值、四字节 atomId(“stco”)、32-位无符号整数(numStcoEntries),以及块偏移值的序列。numStssEntries 的值指定了样本索引值的序列中的条目的数量。此序列中的每一个条目都包括一个 32 位无符号整数,该整数指定了“随机访问样本”的样本索引。随机访问样本索引标识对应于此 trak 原子 900 关联的媒体数据中的媒体播放器可以开始处理媒体数据而不考虑任何前面的样本的点的媒体数据样本 748。此表中的样本索引值必须单调地增大。

[0197] 样本描述表原子(stsd 原子)978 包括原子大小值、四字节 atomId(“stsd”)、32-位无符号整数(numStsdEntries),以及样本描述数据记录的序列。numStsdEntries 的

值指定了样本描述数据记录的序列中的条目的数量。每一个样本描述数据记录都指定用于对由样本到块数据记录中的对应的索引标识的媒体数据样本进行编码的手段。样本描述数据记录的序列通常具有单个条目 ($\text{numStdEntries} = 1$), 该条目指定媒体的类型 (音频、视频、sds、ods), 用于音频和视频样本的压缩算法等等。每一个样本描述表条目都包含在“mp4*”原子 982 内, 其中“mp4*”是“mp4s” (对于 sds 和 ods 样本)、“mp4a” (对于音频样本) 和“mp4v” (对于视频样本) 的通用替代。每一个“mp4*”原子 982 都包含“esds” (基本流描述符) 原子 986, 而每一个 esds 原子 986 都包含 MPEG-4 基本流描述符 (Es_Descr) 数据结构 990。

[0198] MPEG-4 基本流描述符 1000 的结构如图 10A 所示。此数据结构包括一字节标记 (ES_DescrTag) 1004, 后面有数据结构的其余部分的字节数量的指示 1008、16 位 ES_ID 值 (通常为零) 1012、三个 1 位标志 (streamDependenceFlag、URL_Flag 和 OCRstreamFlag) 1016, 以及 5- 位流优先值。1020。如果三个标志 1016 中的任何一个都为非零, 那么其他数据值 (没有显示) 可以跟在流优先值 1020 之后。这些可选数据值不是本发明所需的。

[0199] 流优先值 1020 后面有解码器配置描述符数据结构 1024 和同步层配置描述符数据结构 1028。解码器配置描述符 1024、1032 的结构如图 10B 所示。此数据结构包括一字节标记 (DecoderConfigDescrTag) 1036, 后面有数据结构的其余部分的字节数量的指示 1040, 和一系列数据值: objectType 1044、streamType 1048、upStream 位 1052、预留位 1056、bufferSizeDB 1060、maxBitrate 1064 和 avgBitrate 1068。这些值后面可以有依赖 streamType 和 objectType 的解码器特定的信息数据结构 1072。解码器特定的信息数据结构 1072 是 sds 所必需的, 但不是 ods 所必需的。大多数音频和视频媒体数据流在解码器配置描述符 1032 内还具有解码器特定的信息数据结构。

[0200] 解码器特定的信息数据 1072、1076 的结构如图 10C 所示。此数据结构包括一字节标记 (DecoderSpecificInfoTag) 1080, 后面有数据结构的其余部分的字节数量的指示 1084。其余字节取决于 objectType 和 streamType。在 sds (场景描述流或 BIFS) 的情况下, 解码器特定的信息 1072、1076 包括用于对 nodeID 值进行编码的位数和用于对 routeID 值进行编码的位数的指示。这些值中的每一个值都由 5- 位无符号整数代表。

[0201] 同步层配置描述符 1028、1088 的结构如图 10D 所示。此数据结构包括一字节标记 (SLConfigDescrTag) 1090, 后面有数据结构的其余部分的字节数量的指示 1094 (始终为 1), 单个数据字节 (值 2 “预定义”) 1098, 该字节指出预定义的配置将用于同步层。

[0202] 3.0 场景描述流 (sds)

[0203] 无论是 XMT-A 规范还是 MPEG-4 中间文件规范都没有确定用于对特定类型的音频和视频数据流进行编码或解码的手段, 因此, 这里将不涉及如何对这些流进行编码的细节。XMT-A 文档包含有关流描述流 (sds) 和对象描述流 (ods) 的内容的详细信息。因此, 每一个 XMT-A 文档都与 MPEG-4 中间文件内包含的 sds 和 ods 流密切相关。本节将描述 sds 数据的结构, 下一节将描述 ods 数据的结构。

[0204] 类似于任何其他媒体数据流, sds 数据由一个或多个块组成, 而每一个块每一个都由一个或多个样本组成。如图 11A 所示, sds 二进制块 1100 内的每一个样本都被定义为“命令帧”1110。每一个命令帧 1110 都是字节对齐的。如图 11B 所示, 每一个命令帧 1110 都

包括一个或多个“BIFS 命令”1120。“BIFS”代表“BIInary Format forStreams”(流的二进制格式)。每一个 BIFS 命令 1120 后面都有继续位 1130、1140。如果继续位的值是 (1) 1130, 则另一个 BIFS 命令跟随其后。否则 1140, 继续位后面跟足够数量的空值填充位 1150, 以完成最后一个字节。命令帧中的第一个 BIFS 命令除外, 单个 BIFS 命令 1120 通常不是字节对齐的。

[0205] 如图 12 所示, 有四种类型的 BIFS 命令: “insertion”、“deletion”、“replacement”以及“scene replacement”。BIFS 插入命令 1200 包括两位插入代码 (值=“00”) 1206, 后面有两位参数类型代码 1210 和插入命令数据 1216。BIFS 删除命令 1220 包括两位删除代码 (值=“01”) 1226, 后面有两位参数类型代码 1230 和删除命令数据 1236。BIFS 替换命令 1240 包括两位替换代码 (值=“10”) 1244, 后面有两位参数类型代码 1250 和替换命令数据 1260。BIFS 场景替换命令 1270 包括两位场景替换代码 (值=“11”) 1280, 后面有 BIFS 场景数据结构 1290。

[0206] 如图 13 所示, 有三种类型的 BIFS 插入命令, (a) NodeInsertion 命令, (b) Indexed Value Insertion 命令, 以及 (c) RouteInsertion 命令。插入命令的类型是由参数类型值 1210 确定的。Node Insertion 命令 1300 包括两位插入代码 (值=“00”) 1304, 后面有两位参数类型代码 (值=“01”, 类型=Node) 1308、nodeID 值 1312、两位插入位置代码 1316, 以及 SFNode 数据结构 1324。如果插入位置代码 1316 的值为零, 则 8 位位置值 1320 跟随在插入位置代码 1316 之后。nodeID 值 1312 指定 BIFS 命令中的某处定义的一组可更新的节点中的一个节点。用于对此 nodeID 及其他 nodeID 值进行编码的位数在 sdsM 流的解码器特定的信息 1072 中指定。下面将说明 SFNode 数据结构 1324 的结构。

[0207] Node Insertion 命令 1300 包括两位插入代码 (值=“00”) 1304, 后面有两位参数类型代码 (值=“01”, 类型=Node) 1308、nodeID 值 1312、两位插入位置代码 1316, 以及字段值数据结构 1356。如果插入位置代码 1316 的值为零, 则 8 位位置值 1352 跟随在插入位置代码 1316 之后。nodeID 值 1340 指定 BIFS 命令中的某处定义的一组可更新的节点中的一个节点。用于对 nodeID 值 1340 进行编码的位数在 sdsM 流的解码器特定的信息 1072 中指定。inFieldID 值 1344 标识由 nodeID 1340 的值指定的 BIFS 节点的其中一个数据字段。用于对 inFieldID 值 1344 进行编码的位数取决于 MPEG-4 系统规范中所包含的表。

[0208] 字段值数据结构的内容取决于指定的 BIFS 节点的指定的数据字段的字段数据类型 (布尔、整数、浮点、字符串、节点等等)。在简单时可以是一位, 复杂时可以是 SFNode 数据结构。每一个 BIFS 节点的每一个数据字段的字段数据类型在 MPEG-4 系统规范中包含的表中指定。

[0209] Route Insertion 命令 1360 包括两位插入代码 (值=“00”) 1364, 后面有两位参数类型代码 (值=“11”, 类型=Route) 1368、“isUpdateable”位 1372、departureNodeID 值 1380、departureFieldID 值 1384、arrivalNodeID 值 1388, 以及 arrivalFieldID 值 1392。如果“isUpdateable”位的值为 (1), 则 routeID 值 1376 跟随在“isUpdateable”位 1372 之后。用于对 departureNodeID 值 1380 和 arrivalNodeID 值 1388 进行编码的位数在 sdsM 流的解码器特定的信息 1072 中指定。用于对 departureFieldID 值 1384 进行编码的位数和用于对 arrivalFieldID 值 1392 进行编码的位数取决于 MPEG-4 系统规范中包含的表。

[0210] 如图 14 所示, 有三种类型的 BIFS 删除命令: (a) NodeDeletion 命令, (b) Indexed

Value Deletion 命令,以及 (c.RouteDeletion 命令。删除命令的类型是由参数类型值 1230 确定的。NodeDeletion 命令 1400 包括两位删除代码 (值=“00”)1406,后面有两位参数类型代码 (值=“00”,类型=Node)1412 和 nodeID 值 1418。nodeID 值 1418 指定 BIFS 命令中的某处定义的一组可更新的节点中的一个节点。用于对 nodeID 值 1418 进行编码的位数在 sdsM 流的解码器特定的信息 1072 中指定。

[0211] Indexed Value Deletion 命令 1424 包括两位删除代码 (值=“01”)1430,后面有两位参数类型代码 (值=“10”,类型=IndexedValue)1436、nodeID 值 1442、inFieldID 值 1448,以及两位删除位置代码 1454。nodeID 值 1418 指定 BIFS 命令中的某处定义的一组可更新的节点中的一个节点。用于对 nodeID 值 1418 进行编码的位数在 sdsM 流的解码器特定的信息 1072 中指定。

[0212] Route Deletion 命令 1466 包括两位删除代码 (值=“10”)1472,后面有两位参数类型代码 (值=“11”,类型=Route)1478 和 routeID 值 1484。routeID 值 1484 指定 BIFS 命令中的某处定义的一组可更新的路由中的一个路由。用于对 routeID 值 1484 进行编码的位数在 sdsM 流的解码器特定的信息 1072 中指定。

[0213] 如图 15 所示,有四种类型的替换命令,(a)Node Replacement 命令,(b)Field Replacement 命令,(c.Indexed Value Replacement 命令,以及 (d)Route Replacement 命令。替换命令的类型是由参数类型值 1250 确定的。Node Replacement 命令 1500 包括两位替换代码 (值=“10”)1504,后面有两位参数类型代码 (值=“01”,类型=Node)1508、nodeID 值 1510,以及 SFNode 数据结构 1514。nodeID 值 1510 指定 BIFS 命令中的某处定义的一组可更新的节点中的一个节点。用于对 nodeID 值 1510 进行编码的位数在 sdsM 流的解码器特定的信息 1072 中指定。下面将说明 SFNode 数据结构 1514 的结构。

[0214] Field Replacement 命令 1520 包括两位替换代码 (值=“10”)1524,后面有两位参数类型代码 (值=“01”,类型=Field)1528、nodeID 值 1530、inFieldID 值 1534,以及字段值数据结构 1538。nodeID 值 1530 指定 BIFS 命令中的某处定义的一组可更新的节点中的一个节点。用于对 nodeID 值 1530 进行编码的位数在 sdsM 流的解码器特定的信息 1072 中指定。inFieldID 值 1534 标识由 nodeID 1530 的值指定的 BIFS 节点的其中一个数据字段。用于对 inFieldID 值 1534 进行编码的位数取决于 MPEG-4 系统规范中所包含的表。

[0215] Indexed Value Replacement 命令 1540 包括两位替换代码 (值=“10”)1544,后面有两位参数类型代码 (值=“10”,类型=IndexedValue)1548、nodeID 值 1550、inFieldID 值 1554、两位替换位置代码 1558,以及字段值数据结构 1564。如果替换位置代码 1558 的值为零,则 8 位位置值 1560 跟随在替换位置代码 1558 之后。nodeID 值 1550 指定 BIFS 命令中的某处定义的一组可更新的节点中的一个节点。用于对 nodeID 值 1550 进行编码的位数在 sdsM 流的解码器特定的信息 1072 中指定。inFieldID 值 1554 标识由 nodeID 1550 的值指定的 BIFS 节点的其中一个数据字段。用于对 inFieldID 值 1554 进行编码的位数取决于 MPEG-4 系统规范中所包含的表。

[0216] Route Replacement 命令 1570 包括两位替换代码 (值=“10”)1574,后面有两位参数类型代码 (值=“11”,类型=Route)1578、routeID 值 1580、departureNodeID 值 1584、departureFieldID 值 1588、arrivalNodeID 值 1590,以及 arrivalFieldID 值 1594。routeID 值 1580 指定 BIFS 命令中的某处定义的一组可更新的路由中的一个路由。用于

对 routeID 值 1580 进行编码的位数在 sdsM 流的解码器特定的信息 1072 中指定。用于对 departureNodeID 值 1584 进行编码的位数和用于对 arrivalNodeID 值 1590 进行编码的位数在 sdsM 流的解码器特定的信息 1072 中指定。用于对 departureFieldID 值 1588 进行编码的位数和用于对 arrivalFieldID 值 1594 进行编码的位数取决于 MPEG-4 系统规范中包含的表。

[0217] 如图 12D 所示, ReplaceScene BIFS 命令 1270 包括两位场景替换代码 (值 = “11”) 1280, 后面有 BIFS 场景数据结构 1290。如图 16 所示, BIFS 场景数据结构 1600 包括 6 位预留字段 1610、两个一位标志 (USENAMES 1620 和 protoList 1630)、SFTopNode 数据结构 1640, 以及一位标志 (hasRoutes) 1650。如果 protoList 标志 1630 为真 (1), 那么, MPEG-4 系统规范中定义的其他数据跟随 protoList 标志。SFTopNode 数据结构 1640 是如图 17 所示的 SFNode 数据结构的特殊情况。如果 hasRoutes 标志 1650 为真 (1), 那么, Routes 数据结构 1660 跟随 hasRoutes 标志。Routes 数据结构的结构如图 18 所示。

[0218] 如图 17A、17B 和 17C 所示, SFNode 数据结构可以具有三种形式之一: (a) 重复使用、(b) 屏蔽节点, 以及 (c) 列表节点。所有三种形式都以一位标志 (isReused) 开始。在重复使用的 SFNode 1700 的情况下, isReused 标志的值为 “1” (真) 1704, SFNode 数据结构的其余部分包括 nodeIDref 值 1708。nodeIDref 1708 的值必须匹配 sdsM 数据中某处定义的可更新的 SFNode 的 nodeID 值。

[0219] 如果 isReusedFlag 为假 (0) 1712、1732, 则 SFNode 类型可以具有图 17B 和 17C 所示的两种形式中的一种, 具体取决于 maskAccess 标志位 1722、1742 的值。不论是哪一种情况, SFNode 的数据都包括本地节点类型值 (localNodeType) 1714、1734, 一位标志 (isUpdateable) 1716、1736, 以及第二个一位标志 (maskAccess) 1722、1742。用于对本地节点类型 1714、1734 进行编码的位数取决于 MPEG-4 系统规范中所指定的表。如果 isUpdateable 标志 1716、1736 为真 (1), 则 nodeID 值 1718、1738 跟随在 isUpdateable 标志之后。如果 isUpdateable 标志 1716、1736 为真 (1), 并且关联的 BIFSScene 数据结构 1600 中的 USENAMES 标志 1620 也为真 (1), 那么以零结束的字符串 (“name”) 1720、1740 跟随在 nodeID 值 1718、1738 之后。

[0220] 如果 maskAccess 位为真 (1) 1722, 则 SFNode 具有 “屏蔽节点” 结构 1710。在此情况下, 如图 17B 所示, maskAccess 位 1722 后面有屏蔽位 1726 的有序序列, BIFS 节点的 MPEG-4 规范中定义的每一个 nFields 属性字段一个, 节点类型由 localNodeType 1714 的值给出。在其中这些屏蔽位中的某一个屏蔽位为真 (1) 的每一种情况下, 屏蔽位后面有根据由 localNodeType 1734 确定的字段数据类型 (整数、布尔、字符串、节点等等) 编码的二进制字段值 1728, 字段编号 (屏蔽位的序列内的位置), 以及 MPEG-4 规范中定义的表。

[0221] 如果 maskAccess 位为假 (0) 1742, 则 SFNode 具有 “列表节点” 结构 1730。在此情况下, 如图 17C 所示, MaskAccess 位 1742 后面有一个或多个字段引用记录。每一个字段引用记录都以一位结束标志 1744、1750 开始。如果结束标志为假 (0) 1744, 则结束标志 1744 后面有为本节点类型 1734 定义的属性字段的字段引用索引编号 (fieldRef) 1746, fieldRef 值 1746 后面有根据由本地节点类型 1734 确定的字段数据类型 (整数、布尔、字符串、节点等等) 编码的二进制字段值 1748 和由 fieldRef 值 1746 指出的属性字段。用于对 fieldRef 值 1746 进行编码的位数由 MPEG-4 系统规范中定义的表确定。如果结束标志

为真 (1) 1750, 则字段值的列表结束。

[0222] SFNode 结构内包括的每一个属性字段值都可以包括单个数据值 (SFField 数据结构) 或多个数据值 (MFField 数据结构)。每一个 MFField 数据结构都包含零个或多个 SFField 组件。如图 17D 和 17E 所示, 基于 isList 位 1766、1786 的值, MFField 结构有两种形式, 列表形式 1760 和矢量形式 1780。两种形式都以一位预留位 1762、1782 开始, 后面有 isList 位 1766、1786。

[0223] 如果 isList 位具有值 (1) 1766, 则 MFField 数据结构具有列表形式 1760。在此情况下, isList 位 1766 后面有一位 endFlag 值 1770、1772 的序列。如果 endFlag 位的值是“0”1770, 则 endFlag 位后面有 SFField 数据结构 1774。如果 endFlag 位的值为“1”1772, 则 MFField 数据结构结束。

[0224] 如果 isList 位具有值 (0) 1786, 则 MFField 数据结构具有矢量形式 1780。在此情况下, isList 位 1786 后面有 5 位字段 (nBits) 1790, 该字段指定下面的字段计数值 (nFields) 1792 中的位数。后面有 nFields SFField 结构 1796 的序列。

[0225] 如 MPEG-4 系统规范中指定的表所示, 每一个 SFField 值的结构都取决于与对应的属性字段关联的特定字段数据类型。例如, 布尔字段包括单个位。MPEG-4 系统规范中定义和描述了包括整数、浮点、字符串、SFNode 的其他情况。

[0226] BIFS Scene 数据结构 1600 的最后一个组件是可选的 Routes 数据结构 1660。如图 18A 和 18B 所示, 有两种形式的 Routes 数据结构, 列表形式 1900 和矢量形式 1830。两种形式的 Routes 数据结构都以一位列表标志 1805、1835 开始。如果列表标志的值为真 (1) 1805, 则 Routes 数据结构具有列表形式 1800。在此情况下, 列表位 1805 后面有一个或多个 Route 数据结构 1810, 每一个 Route 数据结构 1810 后面有一位 moreRoutes 标志 1810、1820。如果 moreRoutes 标志的值为真 (1) 1810, 则另一个 Route 数据结构 1810 跟随其后。如果 moreRoutes 标志的值为假 (0) 1820, 则 Routes 数据结构 1800 结束。

[0227] 如果 Routes 数据结构中的列表标志的值为假 (0) 1835, 则 Routes 数据结构具有矢量形式 1830。在此情况下, 列表位 1835 后面有五位 nBits 字段 1840。NBits 字段中包含的无符号整数值指定了用于对下面的 numRoutes 值 1845 进行编码的位数。NumRoutes 值 1845 中编码的无符号整数指定了跟随在 numRoutes 值 1845 之后的 Route 数据结构 1850 的数量。

[0228] 如图 18C 所示, Route 数据结构 1860 包括一位标志 (isUpdateable) 1865、outNodeID 值 1880、outFieldRef 值 1885、inNodeID 值 1890 和 inFieldRef 值 1895。如果 isUpdateable 标志 1865 的值为真 (1), 那么 isUpdateable 标志 1865 后面有 routeID 值 1870。如果 isUpdateable 标志 1865 的值为真 (1), 则对应的 BIFS Scene 数据结构 1600 中的 USENAMES 标志 1620 的值也为真 (1), routeID 值 1870 后面有空值结束字符串 (routeName) 1875。用于对 outNodeID 值、inNodeID 值和 routeID 值进行编码的位数在 sds 流的解码器特定的信息 1072 中指定。用于对 outFieldRef 和 inFieldRef 进行编码的位数由 MPEG-4 系统规范中定义的表确定。

[0229] 4.0 对象描述符流 (odsm)

[0230] 类似于任何其他 MPEG-4 基本流, odsm (对象描述符流) 包含在一个或多个块 736 的序列中。如图 19 所示, 每一个 odsm 块 1900 都由 odsm 样本 1920 的序列组成, 而每一个

odsm 样本 1940 都由 odsm 命令 1960 的序列组成。每一个 odsm 块 1900 中的 odsm 样本 1920 的数量由对象描述符流的 trak 原子 790、900 中的样本到块表原子 (stsc. 960 的内容确定。每一个 odsm 块 1940 中的 odsm 命令 1960 的数量由对象描述符流的 trak 原子 790、900 中的样本大小表原子 (stsz) 970 确定。

[0231] 有两种可能的 odsm 命令: ObjectDescriptorUpdate 命令和 ObjectDescriptorRemove 命令。如图 20A 所示, ObjectDescriptorUpdate 命令 2000 包括一字节 ObjectDescriptorUpdateTag 2010, 命令其余部分中的字节数量的指示 (numBytes) 2020, 以及 ObjectDescriptors 2030 的序列。图 21 中概述了 ObjectDescriptor 的结构。如图 20B 所示, ObjectDescriptorRemove 命令 2040 包括一字节 ObjectDescriptorRemoveTag 2050, 命令其余部分中的字节数量的指示 (numBytes) 2060, objectDescriptorId 值 2070 的序列, 以及 2 到 6 个填充位 2080。

[0232] 每一个 numBytes 值 2020、2060 都指定了 odsm 命令的其余部分的字节数量。如果 numBytes 的值小于 128, 则此值以单字节进行编码。否则, numBytes 的值以大小字节的序列进行编码。每一个大小字节中的高阶位指出是否有另一个大小字节跟随其后。如果此高阶位是“1”, 那么, 会有另一个大小字节跟随其后。每一个大小字节中的其余七个位都指定了 numBytes 的产生的无符号整数值。

[0233] 每一个 objectDescriptorId 值 2070 都以 10 位进行编码, ObjectDescriptorRemove 命令 2040 中的 10 位 objectDescriptorId 值的序列被包装到字节的序列中。如果 objectDescriptorId 值的数量不是 4 的倍数, 则两个、四个或六个空值位 2080 跟随在最后一个 objectDescriptorId 值之后以填充此命令中的最后一个字节。

[0234] 如图 21A 所示, ObjectDescriptorUpdate 命令 2000 内的 ObjectDescriptor 2100 包括一字节 MP4_OD_Tag 2108, 后面有 numBytes 值 2116、十位 ObjectDescriptorID 值 2124、一位 URL_Flag 值 2132、五位预留字段 (0x1f) 2140, 以及 ES_Descr 数据结构或 EsIdRef 数据结构 2148 中的某一个。在此形式的 ObjectDescriptor 中, URL_Flag 2132 的值始终为假 (0)。NumBytes 值 2116 指定了包括对象描述符的其余部分的字节的数量, 以为 ObjectDescriptorUpdate 命令 2000 或 ObjectDescriptorRemove 命令 2040 中的 numBytes 值 2020、2060 同样的方式进行编码。

[0235] ES_Descr 数据结构 1000 的结构如图 10A 所示。如图 21B 所示, EsIdRef 数据结构 2160 包括一字节 ES_ID_RefTag 2170、numBytes 值 2180, 以及 16 位基本流 ID (ES_ID) 值 2190。在此情况下, numBytes 的值始终为“2”, 此值作为 8 位整数来指定。

[0236] 图 22 一般地显示了本发明的操作。过程 2200 基于 MPEG-4 中间文件 2210 的内容创建 XMT-A 文档 2220 和一组媒体数据文件 2230。输入 MPEG-4 中间文件 2210 也可以被称为“mp4 二进制文件”或“mp4 文件”。输出 XMT-A 可以包括基于 XMT-A 规范 ISO/IEC 14496-1 : 2000Amd. 2 的文本文件或一组代表这样的文件的数据结构。输出媒体数据文件 2230 代表 mp4 文件 2210 内包含的音频、视频和图像数据。每一个输出媒体数据文件都由输出 XMT-A 文档 2220 包含的 StreamSource 引用 670 标识。媒体数据文件 2230 的数量可以为零。

[0237] 由本发明 2200 执行的逻辑操作可以 (1) 作为在计算机系统上运行的计算机实现的步骤的序列和 / 或 (2) 作为计算系统内的互相连接的机器模块来实现。实现是依据应用本发明的系统的性能要求来进行选择的。相应地, 构成了这里所描述的本发明的实施例的

逻辑操作也被称为操作、步骤或模块。

[0238] 此外,本发明执行的操作也可以是作为计算机可读的介质来实现的计算机可读的程序。作为示例,而不作为限制,计算机可读的介质可以包括计算机存储介质和通信介质。计算机存储介质包括以存储诸如计算机可读的指令、数据结构、程序模块或其他数据之类的信息的任何方法或技术实现的易失性和非易失性、可移动和非可移动介质。计算机存储介质包括,但不仅限于,RAM、ROM、EEPROM、flash 存储器或其他存储技术、CD-ROM、数字通用光盘(DVD)或其他光存储器、盒式磁带、磁带、磁盘存储器或其他磁存储设备,或可以被用来存储所希望的信息并可以被计算机访问的任何其他介质。通信介质通常包含计算机可读的指令、数据结构、程序模块或诸如载波或其他传输机制之类的经过调制的数据信号中的其他数据,并包括任何信息传送介质。术语“经过调制的数据信号”是指其一个或多个特征以这样的方式设置或更改以便对信号中的信息进行编码的信号。作为示例,而不作为限制,计算机可读的介质可以包括计算机存储介质和通信介质。上述任何组合也应该包括在计算机可读的介质范围内。

[0239] 如图 22 所示,创建输出 XMT-A 文档 2220 和媒体数据文件 2230 的过程分两个步骤完成。在第一个步骤中,mp4 文件到中间文档转换器 2240 解释 mp4 文件 2210,并创建一组中间文档 2245 和一组媒体数据文件 2230。该组中间文档 2245 包括 mp4-file 文档 2250 和 mp4-bifs 文档 2260。在第二个步骤中,中间文档到 XMT-A 转换器 2270 基于中间文档 2245 生成输出 XMT-A 2230。

[0240] 将此过程 2200 分为这两个步骤的一个理由在于,尽管输入 mp4 文件 2210 代表了由输出 XMT-A 文档 2220 和媒体数据文件 2230 代表的相同信息,但是,输入 mp4 文件的组织和结构大大不同于输出 XMT-A 文档的组织和结构。例如,mp4 文件 2210 的结构与 Quicktime 媒体数据文件的结构密切相关,但是,XMT-A 文档 2220 的结构却没有 Quicktime 媒体数据文件的特征。XMT-A 文档 2220 包含场景描述流(sds)和对象描述符流(odsm)的描述,但是这些可以以任何顺序混合在一起。Mp4 文件 2210 还包含 sds 和 odsm 的描述,但是这些描述中的每一个描述都可以作为单独的在时间上排序的样本流。

[0241] 由于 mp4 文件 2210 的结构和组织与 XMT-A 文档 2220 的结构和组织显著不同,因此,将基于 mp4 文件 2210 创建 XMT-A 文档 2220 的过程分为两个步骤是有利的:(a) 解码,和(b) 重新组织。在第一个步骤中,mp4 文件内编码的二进制信息被解释并被用于创建一组表示 mp4 文件的结构和组织的结构化文档:sds 和 odsm。任何其他二进制数据只被复制到新媒体数据文件中。此步骤可以在不考虑 XMT-A 文档的定义的情况下执行。在第二个步骤中,使用在第一个步骤中创建的结构化文档来创建 XMT-A 文档。此步骤可以在不考虑此信息在 mp4 文件以二进制形式表示的方式的情况下执行。

[0242] 为了实现将基于 mp4 文件 2210 创建 XMT-A 文档 2220 的过程分为这两个步骤 2240、2270 的目标,必须定义新的结构化文档 2245,这些文档表示(a) mp4 文件的结构和组织,(b) 如在 mp4 文件中表示的那样的流描述流的结构和组织(sds),以及(c) 如在 mp4 文件中表示的那样的对象描述符流(odsm)的结构和组织。这可以通过定义三种新的结构化文档类型来完成,一个表示 mp4 文件,一个表示 sds,再有一个表示 odsm。其中,表示 mp4 文件和 sds 所要求的结构化文档相对比较复杂,而表示 odsm 所要求的结构化文档非常简单。因此,将 odsm 的描述包括到用于表示 mp4 文件的结构化文档中是方便的。因此,在本发明的

此实施例中,引入了两个新的结构化文档,一个用于 mp4 文件和 odsm,一个用于 sdsm。这两个结构化文档被标识为 mp4-file 文档 2250 和 mp4-bifs 文档 2260。这些文档统称为中间文档 2245。

[0243] 应该注意,为方便起见,已经选择了利用两种类型的结构化文档。通过定义三种类型的结构化文档(仅 mp4-文件、仅 odsm,以及仅 sdsm),可以实现相同的目标,或者,可以将所有三种类型的信息合并到单个复合结构化文档中。

[0244] 在本发明的一个实施例中,为表示重新组织的信息而创建的特定的结构化文档类型是基于“XML”(可扩展标记语言)技术进行的。这是有利的,因为:

[0245] (a) XMT-A 文档的定义基于 XML 技术,

[0246] (b) XML 技术提供了表示结构化文档的标准化手段,以及

[0247] (c) 有标准化的软件工具用于基于 XML 技术处理结构化文档。

[0248] 如此,在本发明的一个实施例中,重新组织 XMT-A 文件中包含的信息的过程被简化为 XML 到 XML 的转换。此外,由于存在标准化的软件用于处理 XML 文件,因此,可以管理输入 XMT-A 文档以及中间文档,而不必开发新的专门软件来执行相同的功能。

[0249] 虽然此实施例是基于使用 XML 技术来表示中间文档,但是,可以创建本发明的使用其他类型的结构化文档的其他实施例。

[0250] 下列材料描述了 (1)mp4-file 文档 2250 的结构,(2)mp4-bifs 文档 2260 的结构,XMT-A 到中间文档转换器 2240 的操作,以及 (4) 中间文档到 mp4 文件转换器 2270 的操作。

[0251] 5.0mp4-file 文档

[0252] 如图 23A 所示,mp4-file 文档 2300 的结构非常类似于 mp4 二进制文件 700 的结构。mp4file 文档 2300 包含一组一个或多个媒体数据 (mdat) 元素 2310 和单个 moov 元素 2320。moov 元素 2320 包括 mp4fiods(mp4 文件初始对象描述符)元素 2330 和一个或多个 trak 元素 2350。moov 元素 2320 还可以包括可选的用户数据 (udta) 元素 2340。udta 元素 2340 可以用来包括诸如版权声明之类的信息。mvhd 原子 772 的属性由 moov 元素 2320 的属性来表示。

[0253] 如图 23B 所示,mp4fiods 元素 2360 拥有 objectDescriptorID 属性 2370。Mp4fiods 元素 2360 还拥有多个其他图 23B 中没有显示的属性。这些其他属性包括布尔属性“includeinlineProfilesFlag”,和整数属性“sceneProfileLevelIndication”、“ODProfileLevelIndication”、“audioProfileLevelIndication”、“visualProfileLevelIndication”和“graphicsProfileLevelIndication”。Mp4fiods 元素 2360 还包括一个或多个 EsIdInc 元素 2380。每一个 EsIdInc 元素 2380 都拥有匹配相关的 trak 元素 2340 的 trackID 属性的 trackID 属性 2390。

[0254] 如图 24A 所示,每一个 mdat 元素 240 都可以包含一个或多个下列元素:sdsm 元素 2410、odsm 元素 2420,以及 mediaFile 元素 2430。这些元素中的每一个元素都拥有匹配相关的 trak 元素 2340 的 trackID 属性的唯一“trackID”属性。每一个 mediaFile 元素 2430 都具有“name”属性,该属性指定包含关联的媒体数据(音频数据、视频数据等等)的外部二进制文件的文件名。每一个 sdsm 元素 2410 都具有“xmlFile”属性,该属性指定表示关联的 mp4-bifs 文档 2260 的 XML 文件的名称。在一个实施例中,创建表示 mp4-file 文档和/或 mp4-bifs 文档的 XML 文件对诊断是有用的,但是这样的文件不是本发明的操作所必需

的。

[0255] 如图 24B 和 24D 所示, 每一个 sdsm 元素 2440 和每一个 mediaFile 元素 280 都包含一个或多个块元素 2450、2490。每一个块元素 2450、2490 都拥有“size”属性, 该属性指出关联的二进制数据块中的字节数量 (如果已知的话)。每一个块元素 2450、2490 还拥有“offset”属性, 该属性指出二进制 sdsm 数据或媒体数据文件的开始和二进制 sdsm 数据或媒体数据文件内的当前块的数据的开始之间的字节数量 (如果已知的话)。描述场景描述流 (sdsm) 的其他信息包含在 mp4-bifs 文档内。

[0256] 如图 24C 所示, 每一个 odsm 元素 2460 都包含一个或多个 odsmChunk 2470 元素。每一个 odsmChunk 元素 2470 都拥有“size”属性, 该属性指出对象描述符流的关联部分的字节数量 (如果已知的话)。每一个 odsmChunk 元素 2470 还拥有“offset”属性, 该属性指出关联的对象描述符流的二进制数据的开始和该流内的当前块的数据的开始之间的字节数量 (如果已知的话)。

[0257] 如图 25A 所示, 每一个 odsmChunk 元素 2500 都包含一个或多个 odsmSample 元素 2510。如图 25B 所示, 每一个 odsmSample 元素 2520 都包含一个或多个 odsm-command 元素 2530。如图 25C 所示, 每一个 odsm-command 元素都可以是 ObjectDescrUpdate 元素 2540 或 ObjectDescrRemove 元素 2570。每一个 ObjectDescrUpdate 元素 2540 都包含 ObjectDescriptor 元素 2550, ObjectDescrUpdate 元素 2540 内包含的 ObjectDescriptor 元素 2550 包含 EsIdRef 元素 2560。

[0258] 每一个 odsmSample 元素 2510、2520 都拥有“time”属性, 该属性指定当 odsmSample 元素 2510、2520 内包含的命令将执行时的时间 (以秒为单位)。每一个 ObjectDescriptor 元素 2550 和每一个 ObjectDescrRemove 元素 2570 都拥有“ODID”属性 2555、2575, 该属性指定数字对象描述符 ID。每一个 EsIdRef 元素 2560 都拥有“EsId”属性 2565, 该属性指定数字基本流 ID。

[0259] Trak 元素 2350、2600 的结构, 如图 26A 所示, 非常类似于 mp4 文件 700 内的 trak 原子 790、900 的结构。每一个 trak 元素 2600 都包含 mdia 元素 2604。Trak 元素 2600 还可以包含 tref (轨道引用) 元素 2636 和 / 或 edts (编辑列表) 元素 2644。没有类似于 mp4 文件中的 tkhd 原子 910 的 tkhd 元素。相反, tkhd 原子 910 内包含属性是作为 trak 元素 2600 的属性来表示的。

[0260] mdia 元素 2604 包含 hdlr 元素 2608 和 minf 元素 2612。mdhd 原子 915 的属性是作为 mdia 元素 2604 的属性来表示。Minf 元素 2612 包含 dinf 元素 2615、stbl 元素 2628, 以及媒体标头元素 2632。媒体标头元素 (“*mhd”) 2632 可以具有多种形式中的某一种形式, 具体取决于关联的数据流中的数据的数据的类型。与 sdsm 或 odsm 关联的 trak 元素内的媒体标头元素 2632 通过“nmhd”元素来表示。与音频流关联的 trak 元素内的媒体标头元素 2632 通过“smhd”元素来表示, 而与视频流关联的 trak 元素内的媒体标头元素 2632 通过“vmhd”元素来表示。

[0261] 如图 26B 所示, stbl (样本表) 元素 2628、2652 包含 stsc (样本到块表) 元素 2656、stts (时间到样本表) 元素 2660、stco (块偏移表) 元素 2664、stsz (样本大小表) 元素 2668, 以及 stsd (样本描述表) 元素 2676。stbl 元素 2664 还可以包括 stss (同步样本表) 元素 2672, 具体取决于流或媒体类型。stsd 元素 2676 可以包含被表示为图 26B 中的“mp4

元素”2680 的多种类型的从属元素中的某一个元素。在与 sdsm 或 odsm 流关联的 trak 元素 2600 内包含的 stsd 元素 2676 的情况下, stsd 元素 2680 包含“mp4s”元素。> 在与音频流关联的 trak 元素 2600 内包含的 stsd 元素 2676 的情况下, stsd 元素 2680 包含“mp4a”元素。· 在与视频流关联的 trak 元素 2600 内包含的 stsd 元素 2676 的情况下, stsd 元素 2680 包含“mp4v”元素。在每一种情况下, “mp4”元素 2680 包含 esds 元素 2684, esds 元素 2684 包含 ES_Descr 元素 2688。

[0262] 如图 27 所示, ES_Descr 元素 2700 包含 DecoderConfigDescriptor 元素 2710 和 SLConfigDescriptor 元素 2760。DecoderConfigDescriptor 元素 2710 可以包含多种类型的解码器特定的信息元素中的某一种, 包括 BIFS_DecoderConfig 元素 2720、JPEG DecoderConfig 2730、VisualConfig 2740 或 AudioConfig2750。各种类型的解码器特定信息元素中的每一种元素都代表二进制 DecoderConfigDescriptor 结构 1032 内包含的 DecoderSpecificInfo 数据结构 1072 的一种形式。二进制 ES_Descr 结构 1000、DecoderConfigDescriptor 结构 1032、SLConfigDescriptor 结构 1088, 以及 DecoderSpecificInfo 结构 1076 的属性通过 mp4-file 文档 2300 的对应元素 2700、2710、2760、2720、2730、2740、2750 的属性来表示。

[0263] 6.0mp4-bifs 文档

[0264] 如图 28A 所示, mp4-bifs 文档 2800 包含单个 bifsConflg 元素 2810, 后面有一个或多个 commandFrame 元素 2820 的序列。如图 28B 所示, 每一个 commandFrame 元素 2830 都包含多个 mp4bifs bifsCommand 元素 2840 中的某一个。每一个 commandFrame 元素 2820、2830 都拥有“time”属性, 该属性指定当 commandFrame 元素内包含的命令将执行时的时间(以秒为单位)。

[0265] 每一个 mp4bifs bifsCommand 元素 2840 都代表十一个可能的 MPEG-4BIFS 命令中的某一个: InsertNode、InsertIndexedValue、InsertRoute、DeleteNode、DeleteIndexedValue、DeleteRoute、ReplaceNode、ReplaceField、ReplaceIndexedValue、ReplaceRoute 和 ReplaceScene。如图 29A 所示, mp4bifsbifsCommand 元素 2910 可以包含一个或多个 mp4bifs Node 元素 2920。在十一种类型的 bifsCommand 元素中, InsertNode、InsertIndexedValue、ReplaceNode、ReplaceField、ReplaceIndexedValue, 以及 ReplaceScene 可以包括从属 mp4bifsNode 元素 2920。InsertIndexedValue、ReplaceIndexedValue 和 ReplaceField bifsCommand 元素可以具有从属 bifsCommand 元素 2910。

[0266] 如图 29B 所示, ReplaceScene bifsCommand 元素 2930 可以只包括单个从属 mp4bifsNode 元素, 这必须是“TopNode”元素 2940。TopNode 元素 2940 对应于 MPEG-4BIFS 节点的特定子集的成员。此子集在 MPEG-4 系统规范中进行了定义。此外, ReplaceScene bifsCommand 元素 2930 还可以包括从属“Routes”元素 2950, 而“Routes”元素 2950 可以包含一个或多个从属“Route”元素 2960。mp4bifs Route 元素 2960 具有属性“routeId”、“arrivalNodeId”、“arrivalField”、“departureNodeID”, 以及“departureField”。

[0267] 除了可能的从属 mp4bifs Node 元素外, 每一种类型的 mp4bifs bifsCommand 元素拥有下列属性值:

[0268] 1. InsertNode: “parentId”、“insertionPosition”, 以及“position”

[0269] 2. InsertIndexedValue : “nodeId”、“inFieldName”、“insertionPosition”、“position”, 以及 “value”

[0270] 3. InsertRoute : “RouteId”、“departureNode”、“departureField”、“arrivalNode”, 以及 “arrivalField”

[0271] 4. DeleteNode : “nodeId”

[0272] 5. DeleteIndexedValue : “nodeId”、“inFieldName”、“deletionPosition”, 以及 “position”

[0273] 6. DeleteRoute : “routeId”

[0274] 7. ReplaceNode : “parentID”

[0275] 8. ReplaceField : “nodeId”、“inFieldName”, 以及 “value”

[0276] 9. ReplaceIndexedValue : “nodeId”、“inFieldName”、“insertionPosition”、“position”, 以及 “value”

[0277] 10. ReplaceRoute : “routeId”、“departureNode”、“departureField”、“arrivalNode”, 以及 “arrivalField”

[0278] 11. ReplaceScene : “USENAMES” (布尔值)

[0279] 对于 bifsCommand 元素 InsertIndexedValue、ReplaceField 和 ReplaceIndexedValue, 如果 “inFieldName” 属性指定的属性字段具有 “node” 的节点数据类型 (根据 MPEG-4 规范), 那么, 此元素将包含一个或多个从属 mp4bifs 节点元素 2920, “value” 属性将包含与每一个从属节点元素关联的节点名称列表。

[0280] mp4bifsNode 元素 2920 代表许多类型的 MPEG-4BIFS Node 数据结构中的某一个。在 MPEG-4 系统规范中定义了 100 以上的不同类型的 BIFS 节点。每一种类型的 MPEG-4BIFS 节点具有特定的 NodeName 和一组属性字段。

[0281] mp4bifs Node 元素有两种基本类型: 原始 Node 元素和重复使用 Node 元素。如图 30A 所示, mp4bifs 原始 Node 元素 3000 通过对应于 MPEG-4 系统规范中定义的其中一个 BIFS 节点的 NodeName 属性的 “NodeName” 来标识。

[0282] mp4bifs 原始 Node 元素 3000 可以具有可选的 NodeId 属性 3010。如果为 NodeId 属性 3010 指定了值, Node 元素 3000 被分类为 “可重复使用的节点”。NodeId 属性 3010 的值, 如果指定的话, 是 1 到当前场景中定义的可重复使用的节点的总数范围内的整数。如果为 NodeId 属性 3010 指定了值, 并且关联的 ReplaceScene 命令的 “USENAMES” 属性的值为 “真”, 那么 Node 元素还具有 “name” 属性 3016。

[0283] 除了 NodeId 3010 和 name 3016 属性外, 每一个原始 Node 元素都具有许多属性字段属性 3020。每一个属性字段属性 3020 都对应于 MPEG-4 系统规范中为特定 Node 元素的 NodeName 标识的节点类型定义的其中一个属性字段。每一个属性字段都具有定义的字段数据类型, 如布尔、整数、浮点等等。可能的字段数据类型集包括 “SFNode” 和 “MFNode”。如果特定原始 Node 元素的 NodeName 对应于具有属性字段的 MPEG-4BIFS 节点或具有字段数据类型 “SFNode” 和 “MFNode” 的字段, 那么 Node 元素可以拥有一个或多个从属 Node 元素 3030。如果如此, 则对应的属性字段属性的值包括与属性字段关联的每一个从属 Node 元素的 NodeName 字符串。

[0284] 例如, 如果具有 NodeName “Group” 的特定 mp4bifs Node 元素拥有从属

mp4bifs Node 元素,这些元素具有与“children”属性关联的 NodeNames “Transform2D”、“Valuator”,以及“TimeSensor”,那么“children”属性的值将是“Transform2D Valuator TimeSensor”。

[0285] 在 Conditional BIFS 节点的特殊情况下,其中一个属性字段具有属性字段名称“buffer”,“buffer”属性字段的字段数据类型是“command buffer”,“buffer”属性字段的值包括一个或多个 BIFS 命令。在此情况下,对应的 mp4bifs Node 元素 3040 的 NodeName 为“Conditional”。Conditional Node 元素 3040 的 NodeId 属性 3050 和 name 属性 3056 的值可以为任何其他 mp4bifs 原始 Node 元素 3000 指定。代替从属 Node 元素 3030,Conditional Node 元素拥有一个或多个从属 bifsCommand 元素 3070,“buffer”属性的值包括从属 bifsCommand 元素 3070 的命令名称的有序列表。

[0286] 例如,如果特定 Conditional Node 元素拥有从属 InsertRoutebifsCommand 元素,后面有从属 DeleteNode bifsCommand 元素,那么,“buffer”属性的值将是“InsertRoute DeleteNode”。

[0287] 原始 Node 元素拥有从属 Node 元素或 bifsCommand 元素的能力可以递归地重复到 BIFS 命令和 Node 元素的层次集合。

[0288] 如图 30C 所示,重复使用的 Node 元素 3080 具有 NodeName “ReusedNode”。ReusedNode 元素 3080 没有从属元素。ReusedNode 元素 3080 具有名为“nodeRef”3090 的单个属性。nodeRef 属性 3090 的值必须匹配其中一个可重复使用的原始 Node 元素 3000、3040 的 NodeId 属性 3010、3050 的值。

[0289] 7.0mp4 到 xmta 转换器

[0290] 如图 22 所示,过程 2200 基于 MPEG-4 中间二进制文件 (“mp4 文件”)2210 创建 XMT-A 文档 2220 和一组零个或多个二进制媒体数据文件 2230。

[0291] 此过程包括两个主要步骤:

[0292] a. 基于 mp4 二进制文件 2210 创建 2240 一对中间 xml 文档 2240、2250 和一组媒体数据文件 2230。

[0293] b. 基于该对中间 xml 文档 2250、2260 创建 2270xmta xml 文档 2220。

[0294] 下面将描述这些步骤中的每一个步骤。

[0295] 基于 mp4 二进制文件创建一对中间 xml 文档和一组媒体数据文件

[0296] 基于 mp4 二进制文件 2210 创建一对中间 xml 文档 2240、2250 和一组媒体数据文件 2230 的过程 2240 包括两个部分:

[0297] (1) 基于 mp4 二进制文件 2210 的内容创建 QtInfo 数据结构 3100。

[0298] (2) 基于所产生的 QtInfo 数据结构 3100 创建一组媒体数据文件 2230 和一对中间 xml 文档 2250、2260。媒体数据文件 2230 集可以是空的。

[0299] 在本发明的另一个实施例中,这两个部分可以合并到单一的过程,其中,直接从 mp4 二进制文件 2210 创建中间 xml 文档 2250、2260 和媒体数据文件 2230,而不必创建 QtInfo 数据结构 3100。如图 7A 所示,mp4 二进制文件 700 由一个 moov 原子 712 和一个或多个 mdat 原子 706 组成。若没有 moov 原子 712 中指定的信息,则不能解释包含创建媒体数据文件 2230 所需要的 mdat 原子 706。因此,在创建中间 xml 文件 2250、2260 和媒体数据文件 2230 之前,创建诸如 QtInfo 数据结构 3100 之类的临时数据结构是有利的。

[0300] QtInfo 数据结构

[0301] QtInfo 数据结构代表 mp4 二进制文件 700 的内容。如图 31A 所示,QtInfo 数据结构 3100 包括下列组件:

[0302] 1. maxNumTracks 3103:指定 mdat 数组 3113 和 trak 数组 3120 中的条目的数量的整数。

[0303] 2. Mp4fInitObjectDescr 3106:表示由此 QtInfo 数据结构 3100 表示的 mp4 二进制文件 700 内包含的 moov 原子 712、754 内包含的 iods 原子 778 的属性的 Mp4fInitObjectDescr 数据结构 3130 的地址,

[0304] 3. mediaDataSize 3108:指定由此 QtInfo 数据结构 3100 表示的 mp4 二进制文件 700 中的 mdAt 原子 706、718 的全部的大小 724 的总和的整数,

[0305] 4. numMdatAtoms 3109:指定由此 QtInfo 数据结构 3100 表示的 mp4 二进制文件 700 中包含的 mdat 原子 706 的数量的整数,

[0306] 5. mdat 数组 3113: maxNumTracks 3103 mdat 结构 3160 的序列的地址。作为备选方案,这可以指定 mdat 结构 3160 的 maxNumTracks 3103 地址的序列的地址。此数组中的头几个 numMdatAtoms mdat 结构中的每一个都表示由此 QtInfo 数据结构 3100 表示的 mp4 二进制文件 700 中包含的 mdat 原子 706 中的某一个,

[0307] 6. MoovHeader 3116:表示由此 QtInfo 数据结构 3100 表示的 mp4 二进制文件 700 内包含的 moov 原子 712、754 内包含的 mvhd 原子 772 的属性的 MoovHeader 数据结构 3170 的地址,

[0308] 7. numTracks 3118:指定由此 QtInfo 数据结构 3100 表示的 mp4 二进制文件 700 中包含的 trak 原子 790 的数量的整数,

[0309] 8. trak 数组 3120: maxNumTracks 3103 trak 结构 3200 的序列的地址。作为备选方案,这可以指定 trak 结构 3200 的 maxNumTracks 3103 地址的序列的地址。此数组中的头几个 numTrakAtoms trak 结构中的每一个都表示由此 QtInfo 数据结构 3100 表示的 mp4 二进制文件 700 中包含的 moov 原子 712、754 内包含 trak 原子 790 中的某一个,

[0310] 9. (UserData):表示由此 QtInfo 数据结构 3100 表示的 mp4 二进制文件 700 中的 moov 原子 712 外部的可能的 udta 原子的可选数据结构的地址,

[0311] Mp4fInitObjectDescr 数据结构 3130 代表 iods 原子 778、800 的属性。如图 31B 所示, Mp4fInitObjectDescr 数据结构 3130 包括下列组件:

[0312] 1. size 3133:指出 iods 原子 778 中的字节的数量的整数,

[0313] 2. version 3136:指出 iods 原子 800 的版本 812 和标志 816 属性的值的整数,

[0314] 3. ObjectDescrID 3138:指出 iods 原子 800 内包含的 Mp4fInitObjectDescr 对象 824 的 ObjectDescriptorID 属性 836 的值的整数,

[0315] 4. urlFlag 3140:指出 iods 原子 800 内包含的 Mp4fInitObjectDescr 对象 824 的 URL_Flag 属性 840 的值的布尔值,

[0316] 5. urlString 3143:如果数量“urlFlag”的值为“假”,那么,数量“urlString”的值为零(空值)。否则,此数量的值指定表示字符串的空值结束的序列字节的地址。

[0317] 6. inlineProfiles 3146:表示 iods 原子 800 内包含的 Mp4fInitObjectDescr 对象 824 的 includeInlineProfilesFlag 属性 844 的值的布尔值,

[0318] 7. ProfileLevels 3148:五个整数的集合:ODProfileLevel、sceneProfileLevel、audioProfileLevel、visual ProfileLevel,以及 graphicsProfileLevel,它们指出 iods 原子 800 内包含的 Mp4fInitObjectDescr 对象 824 的 ODProfileLevelIndication 852、sceneProfileLevelIndication 856、audioProfileLevelIndication 860、visualProfileLevelIndication 864,以及 graphicsProfileLevelIndication 868,

[0319] 8. maxNumEsIdIncs 3153:指出 EsIdInc 数组 3156 中的条目的数量的整数(值为“2”就足够了),

[0320] 9. numEsIdIncs 3153:指出 iods 原子 800 内包含的 Mp4fInitObjectDescr 对象 824 中的 ES_ID_Inc 对象 872 的数量的整数,

[0321] 10. EsIdInc 数组 3156: maxNumEsIdIncs EsIdInc 结构的序列,或 maxNumEsIdIncs EsIdInc 结构的序列的地址。

[0322] 如果 URL_Flag 属性 840 的值为“0”,那么,将给组件“urlFlag”3140 指定值“假”,否则,将给数量“urlFlag”3140 指定值“真”。URL_Flag 属性 840 的值预计为“0”。

[0323] 如果 includeInlineProfilesFlag 属性 844 的值为“0”,那么,将给组件“inLineProfiles”3146 指定值“假”,否则,将给数量“inLineProfiles”3146 指定值“真”。

[0324] EsIdInc 结构包括两个整数,“size”和“trackID”。这些整数表示 Mp4fInitObjectDescr 对象 824 内的对应的 ES_ID_Inc 对象 872、876 中的数量“numBytes”884 和“ES_ID”883 的值。

[0325] 每一个 mdat 结构 3160 都表示 mdat 原子 706、718 的属性。如图 31C 所示,mdat 结构 3160 包括下列组件:

[0326] 1. size 3163:包括 mdat 原子 706 的字节的数量。这与 mdat 原子 718 的“原子大小”属性 724 相同。

[0327] 2. start 3166:指定 mp4 二进制文件 700 的开始和 mdat 原子 706 的第一个字节之间的字节的数量。

[0328] 3. MediaData 3168:这是(大小-8)字节的数组的地址,包含 mdat 原子 706 内包含的块 736 的集的副本。

[0329] MoovHeader 结构 3170 表示 mvhd 原子 772 的属性。如图 31D 所示,MoovHeader 结构 3170 包括下列值:

[0330] 1. size 3173:指出 mvhd 原子 772 中的字节的数量的整数,

[0331] 2. version&flags 3176:一对指出 mvhd 原子 772 的“version”和“flags”属性的值的整数,

[0332] 3. creationTime 3178:指出 mvhd 原子 772 的“creationTime”属性的值的整数,

[0333] 4. modifiedTime 3180:指出 mvhd 原子 772 的“modificationTime”属性的值的整数,

[0334] 5. timeScale 3183:指出 mvhd 原子 772 的“timeScale”属性的值的整数,

[0335] 6. duration 3186:指出 mvhd 原子 772 的“duration”属性的值的整数,

[0336] 7. nextTrackID 3188:指出 mvhd 原子 772 的“nextTrackID”属性的值的整数,

[0337] 每一个 trak 结构 3200 都表示 trak 原子 790、900 的属性。如图 32A 所示,每一个 trak 结构 3200 都可以包含下列组件:

- [0338] 1.size 3202:指出 trak 原子 900 中的字节的数量的整数,
- [0339] 2.TrackHeader 3204:TrackHeader 数据结构 3212 的地址,表示由此 trak 结构 3200 表示的 trak 原子 900 内包含的 tkhd 原子 910 的属性,
- [0340] 3.TrackMedia 3206:TrackMedia 数据结构 3226 的地址,表示由此 trak 结构 3200 表示的 trak 原子 900 内包含的 mdia 原子 912 的属性,
- [0341] 4.TrackEdit 3208:可选 TrackEdit 数据结构 3236 的地址,表示由此 trak 结构 3200 表示的 trak 原子 900 内包含的可选 edts 原子 945 的属性,
- [0342] 5.TrackReference 3210:可选 TrackReference 数据结构 3264 的地址,表示由 trak 结构 3200 表示的 trak 原子 900 内包含的可选 tref 原子 940 的属性,
- [0343] TrackHeader 结构 3212 表示 tkhd 原子 910 的属性。如图 32B 所示,TrackHeader 结构 3212 包括下列组件:
- [0344] 1.size 3214:指出 tkhd 原子 910 中的字节的数量的整数,
- [0345] 2.version&flags 3216:一对指出 tkhd 原子 910 的“version”和“flags”属性的值的整数,
- [0346] 3.creationTime 3218:指出 tkhd 原子 910 的“creationTime”属性的值的整数,
- [0347] 4.modifiedTime 3220:指出 tkhd 原子 910 的“modificationTime”属性的值的整数,
- [0348] 5.duration 3222:指出 tkhd 原子 910 的“duration”属性的值的整数,
- [0349] 6.trackID 3224:指出 tkhd 原子 910 的“trackID”属性的值的整数,
- [0350] TrackMedia 结构 3226 表示 mdia 原子 912 的属性。如图 32C 所示,TrackMedia 结构 3226 包括下列组件:
- [0351] 1.size 3228:指出 mdia 原子 912 中的字节的数量的整数,
- [0352] 2.MediaHeader 3230:MediaHeader 结构 3300 的地址,表示 mdia 原子 912 内包含的 mdhd 原子 915 的属性,
- [0353] 3.Handler 3232:Handler 结构 3320 的地址,表示 mdia 原子 912 内包含的 hdlr 原子 918 的属性,
- [0354] 4.MediaInfo 3234:MediaInfo 结构 3333 的地址,表示 mdia 原子 912 内包含的 minf 原子 920。
- [0355] TrackEdit 结构 3236 表示 edts 原子 945 的属性。如图 32D 所示,TrackEdit 结构 3236 包含下列组件:
- [0356] 1.size 3238:指出 edts 原子 945 中的字节的数量的整数,
- [0357] 2.version&flags 3240:一对指出 edts 原子 945 的“version”和“flags”属性的值的整数,
- [0358] 3.EditList 3242:表示 edts 原子 945 中包含的 elst 原子 948 的 EditList 结构 3244 的地址。
- [0359] EditList 结构 3244 表示 elst 原子 948 的属性。如图 32E 所示,EditList 结构 3244 包括下列组件:
- [0360] 1.size 3246:指出 elst 原子 948 中的字节的数量的整数,
- [0361] 2.version&flags 3248:一对指出 elst 原子 948 的“version”和“flags”属性的

值的整数，

[0362] 3.numSegments 3250 :指定 EditSegment 数组 3252 中的 EditSegment 结构 3256 的数量的整数，

[0363] 4.EditSegment array 3252 :这可以是 numSegmentsEditSegment 结构 3256 的序列或 EditSegment 结构 3256 的 numSegments 地址的序列。

[0364] 每一个 EditSegment 结构 3256 都表示由包含此 EditSegment 结构 3256 的 EditList 结构 3244 表示的 elst 原子 948 内包含的“edit segment”的属性。如图 32F 所示，每一个 EditSegment 结构 3256 都包括三个整数 :startTime 3258、duration 3260，以及 relativeRate 3262。这些整数中的每一个整数都表示对应的 editsegment 的名称类似的属性。

[0365] TrackReference 结构 3264 表示 tref 原子 940 的属性。如图 32G 所示，TrackReference 结构 3264 包括下列组件：

[0366] 1.size 3266 :指出 tref 原子 940 中的字节的数量的整数，

[0367] 2.Mpod 3268 :表示 tref 原子 940 内包含的 Mpod 原子 942 的 Mpod 结构 3270 的地址。

[0368] Mpod 结构 3270 表示 Mpod 原子 942 的属性。如图 32H 所示，Mpod 结构 3270 包括下列组件：

[0369] 1.size 3272 :指出 mpod 原子 942 中的字节的数量的整数，

[0370] 2.numEntries 3274 :指定 trackID 数组 3276 中的条目的数量的整数，

[0371] 3.trackID array 3276 :numEntries 整数的序列的地址。这些整数中的每一个整数都表示 mpod 原子 942 内指定的其中一个 trackID 值。

[0372] UserData 结构 3280 表示 udta 原子 784 的属性。如图 32I 所示，UserData 结构 3280 包括下列组件：

[0373] 1.size 3282 :指出 udta 原子 784 中的字节的数量的整数，

[0374] 2.numCprt 3284 :指定 cprt 数组 3286 的成员的数量的整数，

[0375] 3.cprt array 3286 :cprt 结构 3290 的序列，或 cprt 结构 3290 的地址的序列。

[0376] cprt 结构 3290 表示 cprt 原子的属性。如图 32J 所示，cprt 结构 3290 包括下列组件：

[0377] 1.size 3282 :指出 cprt 原子中的字节的数量的整数，

[0378] 2.version&flags 3294 :一对指出 cprt 原子的“version”和“flags”属性的值的整数，

[0379] 3.language 3296 :指出 cprt 原子的“language”属性的值的整数，以及

[0380] 4.notice 3298 :指出 cprt 原子中指定的消息的空值结束的字符串。

[0381] MediaHeader 结构 3300 表示 mdhd 原子 915 的属性。如图 33A 所示，MediaHeader 结构 3300 包括下列值：

[0382] 1.size 3303 :指出 mdhd 原子 915 中的字节的数量的整数，

[0383] 2.version&flag 3306 :一对指出 mdhd 原子 915 的“version”和“flags”属性的值的整数，

[0384] 3.creationTime 3308 :指出 mdhd 原子 915 的“creationTime”属性的值的整数，

[0385] 4.modifiedTime 3310:指出 mdhd 原子 915 的“modificationTime”属性的值的整数,

[0386] 5.timeScale 3313:指出 mdhd 原子 915 的“timeScale”属性的值的整数,

[0387] 6.duration 3316:指出 mdhd 原子 915 的“duration”属性的值的整数,

[0388] 7.language 3318:指出 mdhd 原子 915 的“language”属性的值的整数,

[0389] Handler 结构 3320 表示 hdlr 原子 918 的属性。如图 33B 所示,Handler 结构 3320 包括下列值:

[0390] 1.size 3323:指出 hdlr 原子 918 中的字节的数量的整数,

[0391] 2.version&flags 3326:一对指出 hdlr 原子 918 的“version”和“flags”属性的值的整数,

[0392] 3.handlerType 3328:指定 hdlr 原子 918 的处理程序类型属性的 4 字符标记,以及

[0393] 4.name 3330:为描述或诊断目的提供的空值结束的字符串的地址。

[0394] MediaInfo 结构 3333 表示 minf 原子 920 的属性。如图 33C 所示,MediaInfo 结构 3333 包括 MediaDataInfo 结构 3338、3350 的地址,SampleTables 结构 3340、3400 的地址,以及 MediaInfoHeader 结构 3343 的地址。MediaInfoHeader 结构 3343 表示下列媒体信息标头原子 936 之一:

[0395] 1.nmhd 原子(对于场景描述流,sdsm,或对象描述符流,odsm),

[0396] 2.vmhd 原子(对于视频流),或

[0397] 3.smhd 原子(对于音频流)。

[0398] MediaInfoHeader 结构 3343 包括下列值:

[0399] 1.大小:指出媒体信息标头原子 936 中的字节的数量的整数,

[0400] 2.version&flags:一对指出媒体信息标头原子 936 的“version”和“flags”属性的值的整数,

[0401] MediaDataInfo 结构 3350 表示 dinf 原子 924 的属性。如图 33D 所示,MediaDataInfo 结构 3350 包括下列组件:

[0402] 1.size 3353:指出 dinf 原子 924 中的字节的数量的整数,

[0403] 2.version&flags 3356:一对指出 dinf 原子 924 的“version”和“flags”属性的值的整数,

[0404] 3.DataRefTable 3358:DataRefTable 数据结构 3360 的地址。

[0405] DataRefTable 结构 3360 表示 dref 原子 927 的属性。如图 33E 所示,DataRefTable 结构 3360 包括下列组件:

[0406] 1.size 3363:指出 dref 原子 927 中的字节的数量的整数,

[0407] 2.version&flags 3366:一对指出 dref 原子 927 的“version”和“flags”属性的值的整数,

[0408] 3.numDataRefs 3368:指出 dataRef 数组 3370 中的条目的数量的整数,以及

[0409] 4.dataRef array 3370:DataRef 数据结构 3380 的序列的地址。

[0410] DataRef 结构 3380 表示数据引用表中的条目。如图 33F 所示,dataRef 结构 3380 包括下列组件:

[0411] 1. size 3383 :指出由此 dataRef 结构 3380 表示的数据引用结构中的字节的数量的整数,

[0412] 2. version&flags 3386 :一对指出数据引用结构的“version”和“flags”属性的值的整数,

[0413] 3. dataRefType 3388 :指出由此 dataRef 结构 3380 表示的数据引用结构的类型的四个字符的序列,以及

[0414] 4. dataRefData 3390 :表示数据引用结构的其余部分的大小 -12 字节的序列的地址。

[0415] SampleTables 结构 3400 表示 stbl 原子 933、950 的属性。如图 34A 所示, SampleTables 结构 3400 包括下列组件:

[0416] 1. size 3403 :指出 stbl 原子 950 中的字节 954 的数量的整数,

[0417] 2. SampleToChunkTable 3406 :表示 stsc 原子 960 的内容的数据结构的地址,

[0418] 3. ChunkOffsetTable 3408 :表示 stco 原子 966 的内容的数据结构的地址,

[0419] 4. TimeToSampleTable 3410 :表示 stts 原子 963 的内容的数据结构的地址,

[0420] 5. SampleSizeTable 3413 :表示 stsz 原子 970 的内容的数据结构的地址,

[0421] 6. SyncSampleTable 3416 :表示 stss 原子 974 的内容的可选数据结构的地址,

[0422] 7. SampleDescriptionTable 3418 :表示 stsd 原子 978 的内容的数据结构的地址,

[0423] 除 SampleSizeTable 3413 外, SampleTables 结构 3400 中包含的每一个样本表都具有下列成员,如图 34B 3420 所示:

[0424] 1. size 3423 :指出由此数据结构表示的样本表原子中的字节的数量的整数,

[0425] 2. version&flags 3426 :一对指出由此数据结构表示的样本表原子的“version”和“flags”属性的值的整数,

[0426] 3. numEntries 3428 :指出表数组 3430 中的条目的数量的整数,以及

[0427] 4. table array 3430 :整数或数据结构的数组的地址。

[0428] 如图 34C 所示, SampleSizeTable 结构 3440 类似于其他样本表,只是它还包括另一个成员,“sampleSize”3450,这是一个指定所有样本的固定样本大小的整数。如果成员“sampleSize”3450 的值为非零,成员“table array”3456 被省略。

[0429] 成员“table array”的结构取决于样本表的类型。在 ChunkOffsetTable 结构 3408、SampleSizeTable 结构 3413 以及 SyncSampleTable 结构 3416 的情况下,此成员表示 32 位整数的序列。

[0430] 在 SampleToChunkTable 结构 3406 的情况下,表数组表示 SampleToChunk 数据结构 3460 的数组。如图 34D 所示,每一个 SampleToChunk 数据结构 3460 都包括三个整数成员,“firstChunk”3463、“numSamples”3466,以及“description”3468。

[0431] 在 TimeToSampleTable 结构 3410 的情况下,表数组表示 TimeToSample 数据结构 3470 的数组。如图 34E 所示,每一个 TimeToSample 数据结构 3470 包括两个整数成员,“count”3473 和“duration”3476。

[0432] 在 SampleDescriptionTable 结构 3418 的情况下,表数组表示 SampleDescription 数据结构的数组。如图 34F 所示,每一个 SampleDescription 数据结构 3480 都包括下列六个成员:

[0433] 1. size 3483 :指出由此 SampleDescription 结构 3480 表示的样本描述表条目中的数量的整数,

[0434] 2. dataFormat 3486 :指定由此 SampleDescription 结构 3480 表示的 mp4 * 原子 982 的四字符标识符 (“mp4s”、“mp4a”或“mp4v”),

[0435] 3. dataReference 3488 :表示由此 SampleDescription 结构 3480 表示的 mp4 * 原子 982 的 dataReference 属性的整数,

[0436] 4. esdversion 3490 :指出由此 SampleDescription 结构 3480 表示的 mp4 * 原子 982 中包含的“esds”原子 986 的版本属性的整数,

[0437] 5. esdSize 3493 :指出由此 SampleDescription 结构 3480 表示的 mp4 * 原子 982 中包含的“esds”原子 986 中的字节的数量的整数,

[0438] 6. esdData 3496 :包含由此 SampleDescription 结构 3480 表示的 mp4 * 原子 982 中包含的“esds”原子 986 内包含的 ES_Descr 对象结构 990 的副本的字节的数组的地址。

[0439] 基于 mp4 二进制文件创建 QtInfo 结构

[0440] 基于 mp4 二进制文件创建 QtInfo 结构的过程从基于数量“maxNumTracks”的特定值创建新的空的 QtInfo 数据结构。数量“maxNumTracks”的值可以是诸如 128 之类的预先确定的常数,该常数足够大,以超过特定 mp4 二进制文件中的轨道的实际数量,或者也可以通过打开 mp4 二进制文件并统计该文件内包含的 trak 原子的数量来确定。在已经创建新的空的 QtInfo 结构之后,按如图 35 所示的那样对 mp4 文件进行解释。此过程包括下列步骤:

[0441] 1. 操作 3500,打开 mp4 二进制文件(读取)。

[0442] 2. 操作 3510,从 mp4 文件读取 32 位整数,并将此整数的值指定给数量“atomSize”。然后,从 mp4 文件读取第二个 32 位整数,并将此整数的值指定给数量“atomId”。

[0443] 3. 操作 3520,将数量“atomId”的值与值“mdat”进行比较。

[0444] 4. 操作 3525,如果数量“atomId”的值对应于“mdat”,则执行过程“处理 mdat 原子”。

[0445] 5. 操作 3530,将数量“atomId”的值与值“moov”进行比较。

[0446] 6. 操作 3535,如果数量“atomId”的值对应于“moov”,则执行过程“处理 moov 原子”。

[0447] 7. 操作 3540,将数量“atomId”的值与值“udta”进行比较。

[0448] 8. 操作 3545,如果数量“atomId”的值对应于“udta”,则执行过程“处理 udta 原子”,后面跟下面的操作 3570。

[0449] 9. 否则,执行操作 3560,跳过 mp4 文件中的 atomSize-8 字节。即,从 mp4 文件中读取此数量的字节,但是不对所产生的值执行任何操作。

[0450] 10. 操作 3570,测试 mp4 二进制文件 700 的当前文件位置。

[0451] 11. 如果 mp4 二进制文件 700 的当前文件位置对应于文件条件的末尾,则操作 3580 完成该过程(读取的字节的数量等于完整的 mp4 文件中的字节的数量)。否则,重复操作 3510 到操作 3570。

[0452] 如图 36 所示,过程“处理 mdat 原子”操作 3530 包括下列步骤:

- [0453] 1. 操作 3600, 创建新的 MediaData 结构 “newMediaData” 3160。
- [0454] 2. 操作 3610, 将此新的 MediaData 结构 3160 的地址指定到 QtInfo 结构 3100 中的 mdat 数组 3113 中的条目 numMdatAtoms。
- [0455] 3. 操作 3620, 将 numMdatAtoms 3110 的值增大 1。
- [0456] 4. 操作 3630, 将数量 “atomSize” 的值指定到新的 MediaData 结构 3160 中的成员 “size” 3163。
- [0457] 5. 操作 3640, 将当前文件指针的值 (-8) 指定到新的 MediaData 结构 3160 中的成员 “start” 3166。
- [0458] 6. 操作 3650, 创建具有 (atomSize-8) 字节的新字节数组, 并将此数组的地址指定到新的 MediaData 结构 3160 中的成员 “MediaData” 3168。
- [0459] 7. 操作 3660, 从 mp4 二进制文件读取 (atomSize-8) 字节, 并将结果复制到新的 MediaData 结构 3160 中的成员 “MediaData” 3168 指定的新的字节数组。
- [0460] 如图 37 所示, 过程 “处理 moov 原子” 操作 3550 包括下列步骤:
- [0461] 1. 操作 3700, 将零值指定到 QtInfo 结构 3100 中的成员 “numTracks” 3118。
- [0462] 2. 操作 3710, 从 mp4 文件读取 32 位整数, 并将此整数的值指定给数量 “atomSize”。然后, 从 mp4 文件读取第二个 32 位整数, 并将此整数的值指定给数量 “atomId”。
- [0463] 3. 操作 3720, 将数量 “atomId” 的值与值 “mvhd” 进行比较。
- [0464] 4. 操作 3725, 如果数量 “atomId” 的值对应于 “mvhd”, 则执行过程 “处理 mvhd 原子”。然后继续执行下面的操作 3770。
- [0465] 5. 操作 3730, 将数量 “atomId” 的值与值 “iods” 进行比较。
- [0466] 6. 操作 3735, 如果数量 “atomId” 的值对应于 “iods”, 则执行过程 “处理 iods 原子”。然后继续执行下面的操作 3770。
- [0467] 7. 操作 3740, 将数量 “atomId” 的值与值 “trak” 进行比较。
- [0468] 8. 操作 3745, 如果数量 “atomId” 的值对应于 “trak”, 则执行过程 “处理 trak 原子”。然后继续执行下面的操作 3770。
- [0469] 9. 否则, 执行操作 3760, 跳过 mp4 文件中的 atomSize-8 字节。即, 从 mp4 文件中读取此数量的字节, 但是不对所产生的值执行任何操作。
- [0470] 10. 操作 3770 检查是否到达 moov 原子 754 的末端。当自 moov 原子 754 的开始从 mp4 文件读取的数据字节的数量等于在操作 3710 中确定的数量 atomSize 的值时, 满足到达了 moov 原子 754 的末端的条件。
- [0471] 12. 如果已经到达了 moov 原子 (754) 的末端, 则操作 3780 完成了此过程。否则, 重复操作 3710 到操作 3770。
- [0472] 过程 “处理用户数据原子” 可以用来表示 UserData 结构 3280 中的可选 udta 原子 784 的内容。此信息不是最后的 XMT-A 文件所需的, 因此, 只需跳过任何 udta 原子 784 的其余部分 (atomSize 减去 8 字节) 就足够了。
- [0473] 或者, 下列步骤可以用来解释 udat 原子 784 的内容:
- [0474] 1. 创建新的 UserData 数据结构 3280。
- [0475] 2. 将此新的 UserData 数据结构 3280 的地址指定到 QtInfo 结构 3100 中的成员

UserData 3123。

[0476] 3. 将数量“atomSize”的值指定到此新的 UserData 数据结构 3280 中的成员“size”3282。

[0477] 4. 将值“0”指定到 UserData 结构 3280 的成员“numCprt”3284。

[0478] 5. 重复过程“读取 Copyright 原子”，直到到达 udta 原子 784 的末端。当自 udta 原子 784 的开始从 mp4 文件读取的数据字节的数量等于成员“size”3282 的值时，满足到达了 udta 原子 784 的末端的条件。

[0479] 过程“读取 Copyright 原子”包括下列步骤：

[0480] 1. 从 mp4 文件读取 32 位整数，并将此整数的值指定给数量“atomSize”。然后，从 mp4 文件读取第二个 32 位整数，并将此整数的值指定给数量“atomId”。

[0481] 2. 将数量“atomId”的值与值“cprt”进行比较。

[0482] 3. 如果数量“atomId”的值不对应于“cprt”，则跳过 mp4 文件中的数据的 atomSize-8 字节，并继续执行过程“处理用户数据原子”的步骤 5，否则执行下列步骤。

[0483] 4. 创建新的 cprt 结构“newCprt”3290。

[0484] 5. 将此新的 cprt 结构的地址指定给 UserData 结构 3280 的 cprt 数组成员 3286 中的条目 numCprt。

[0485] 6. 将数量“atomSize”的值指定到新的 cprt 结构 3290 中的成员“size” (3292)。

[0486] 7. 从 mp4 文件读取 32 位整数，并将此整数的值指定给新的 cprt 结构 3290 的成员“version&flags”3294。

[0487] 8. 从 mp4 文件读取 16 位整数，并将此整数的值指定给新的 cprt 结构 3290 的成员“language”3296。

[0488] 9. 从 mp4 文件读取字节的空值结束的序列，并将这些字节复制到新的字符串量“newNotice”。

[0489] 10. 将新的字符串量“newNotice”的地址指定到新的 cprt 结构 3290 中的成员“notice” (3298)。

[0490] 11. 将 UserData 结构 3280 的成员 numCprt 3284 的值增大 1。

[0491] 过程“处理 mvhd 原子”操作 3725 包括下列步骤：

[0492] 1. 创建新的 MoovHeader 数据结构 3170。

[0493] 2. 将此新的 MoovHeader 数据结构 3170 的地址指定到 QtInfo 结构 3100 中的成员 MoovHeader 3116。

[0494] 3. 将数量“atomSize”的值指定到此新的 MoovHeader 数据结构 3170 中的成员“size”3173。

[0495] 4. 从 mp4 文件读取 32 位整数，并将结果指定给 MoovHeader 结构 3170 中的成员“version&flags”3176。

[0496] 5. 从 mp4 文件读取 32 位整数，并将结果指定给 MoovHeader 结构 3170 中的成员“creationTime”3178。

[0497] 6. 从 mp4 文件读取 32 位整数，并将结果指定给 MoovHeader 结构 3170 中的成员“modificationTime”3180。

[0498] 7. 从 mp4 文件读取 32 位整数，并将结果指定给 MoovHeader 结构 3170 中的成员

“timeScale”3183。

[0499] 8. 从 mp4 文件读取 32 位整数,并将结果指定给 MoovHeader 结构 3170 中的成员 “duration”3186。

[0500] 9. 读取 mp4 文件中的 76 字节,并忽略所产生的值。

[0501] 10. 从 mp4 文件读取 32 位整数,并将结果指定给 MoovHeader 结构 3170 中的成员 “nextTrackID”3188。

[0502] 过程“处理 iods 原子”包括下列步骤:

[0503] 1. 创建新的 Mp4fInitObjectDescr 数据结构 3130。

[0504] 2. 将此新的 Mp4fInitObjectDescr 数据结构 3130 的地址指定给 QtInfo 结构 3100 中的成员 Mp4fInitObjectDescr 3106。

[0505] 3. 将“atomSize”的值指定到 Mp4fInitObjectDescr 结构 3130 中的成员 “size”3133。

[0506] 4. 从 mp4 文件读取 32 位整数 812,816,并将结果指定给 Mp4fInitObjectDescr 结构 3130 中的成员 “version”3136。

[0507] 5. 从 mp4 文件读取 8 位整数 828,并验证结果具有预期值“16”(MP4_IOD_Tag)。

[0508] 6. 从 mp4 文件读取 8 位整数 832,并将结果指定到数量“objSize”。

[0509] 7. 如果 objSize 的值大于 127,则重复步骤 6。

[0510] 8. 从 mp4 文件读取 16 位整数,并将结果指定给数量“odidandflags”。

[0511] 9. 将值“odidandflags”836 的高阶 10 位的值指定到 Mp4fInitObjectDescr 结构 3130 中的成员 “ObjectDescriptorID”3138。

[0512] 10. 如果逻辑操作的结果 (“odidandflags”的值)和 32) 为零,则将值“0”指定到 Mp4fInitObjectDescr 结构 3130 中的成员 “urlFlag”3140。否则,将值“1”指定到 Mp4fInitObjectDescr 结构 3130 中的成员 “urlFlag”3140。

[0513] 11. 如果逻辑操作的结果 (“odidandflags”的值)和 16) 为零,则将值“0”指定到 Mp4fInitObjectDescr 结构 3130 中的成员 “inlineProfiles”3146。否则,将值“1”指定到 Mp4fInitObjectDescr 结构 3130 中的成员 “inlineProfiles”3146。

[0514] 12. 将值“2”指定到 Mp4fInitObjectDescr 结构 3130 中的成员 “maxNumEsIdIncs”3150。

[0515] 13. 将零值指定到 Mp4fInitObjectDescr 结构 3130 中的成员 numEsIdIncs 3153。

[0516] 14. 创建 maxNumEsIdIncs 地址值的数组。

[0517] 15. 将此数组的地址指定到 Mp4fInitObjectDescr 结构 3130 中的成员 “EsIdInc 数组”3156。

[0518] 16. 如果成员 urlFlag 的值为“1”,则执行过程“处理 URL 字符串”。否则 (urlFlag 的值为“0”),则执行过程“处理配置文件级别的指示器”。

[0519] 过程“处理 URL 字符串”包括下列步骤:

[0520] 1. 从 mp4 文件读取字节的空值结束的序列,

[0521] 2. 将这些字节保存到新的字节数组中,

[0522] 3. 将此新的字节数组的地址指定到 Mp4fInitObjectDescr 结构 3130 中的成员 urlString 3143。

[0523] 过程“处理配置文件级别的指示器”包括下列步骤：

[0524] 1. 将零值指定到 Mp4fInitObjectDescr 结构 3130 中的数量 urlString 3143。

[0525] 2. 从 mp4 文件中读取五个配置文件级别的指示值 852 到 868, 并将每一个值指定到 Mp4fInitObjectDescr 结构 3130 的对应的成员。这些配置文件级别的值中的每一个值都包括单一的字节, 这 5 个值都由 Mp4fInitObjectDescr 结构 3130 的“ProfileLevels”3146 部分表示。

[0526] 3. 重复过程“处理 EsIdInc 对象”, 直到到达 iods 原子 800 的末端。当自 iods 原子 800 的开始从 mp4 文件读取的数据字节的数量等于 iods 原子 800 的大小 804、3133 时, 满足到达了 iods 原子 800 的末端的条件。

[0527] 过程“处理 EsIdInc 对象”包括下列步骤：

[0528] 1. 从 mp4 文件读取一字节对象标记 880。

[0529] 2. 验证对象标记的值是否具有预期值 (14, “EsIdIncTag”)。

[0530] 3. 创建新的 EsIdInc 结构。此结构包含两个成员值, “size”和“trackID”。

[0531] 4. 将此新的 EsIdInc 结构的地址指定到 Mp4fInitObjectDescr 结构 3130 中的 EsIdInc 数组 3156 中的条目 numEsIdIncs。

[0532] 5. 读取包括 EsIdInc 对象的其余部分的字节 884 的数量, 并将结果指定到新的 EsIdInc 结构的成员“size”。

[0533] 6. 从 mp4 文件 888 读取 32 位整数, 并将结果指定给新的 EsIdInc 结构的成员“trackID”。

[0534] 7. 将成员 numEsIdIncs 3153 的值增大 1。

[0535] 如图 38 所示, 过程“处理 trak 原子”操作 3745 包括下列步骤：

[0536] 1. 操作 3800, 创建新的 trak 数据结构 3200。将此新的 trak 结构的地址指定到当前 QtInfo 结构 3100 的 trak 数组成员 3120 中的条目 numTracks。将数量“AtomSize”的值指定到新的 trak 结构 3200 中的成员“size”3202。

[0537] 2. 操作 3810, 从 mp4 文件读取 32 位整数, 并将此整数的值指定给数量“atomSize”。然后, 从 mp4 文件读取第二个 32 位整数, 并将此整数的值指定给数量“atomId”。

[0538] 3. 操作 3820, 将数量“atomId”的值与值“tkhd”进行比较。

[0539] 4. 操作 3825, 如果数量“atomId”的值对应于“tkhd”, 则执行过程“处理 track header(tkhd) 原子”。然后继续执行操作 3870。

[0540] 5. 操作 3830, 将数量“atomId”的值与值“mdia”进行比较。

[0541] 6. 操作 3835, 如果数量“atomId”的值对应于“mdia”, 则执行过程“处理 track media(mdia) 原子”。然后继续执行操作 3870。

[0542] 7. 操作 3840, 将数量“atomId”的值与值“edts”进行比较。

[0543] 8. 操作 3845, 如果数量“atomId”的值对应于“edts”, 则执行过程“处理 track edit(edts) 原子”。然后继续执行操作 3870。

[0544] 9. 操作 3850, 将数量“atomId”的值与值“tref”进行比较。

[0545] 10. 操作 3855, 如果数量“atomId”的值对应于“tref”, 则执行过程“处理 track reference(tref) 原子”。然后继续执行操作 3870。

[0546] 11. 否则,执行操作 3860,跳过 mp4 文件中的 atomSize-8 字节。即,从 mp4 文件读取 atomSize-8 字节,并忽略所产生的值。

[0547] 12. 操作 3870,检查是否到达 trak 原子 900 的末端。当自 trak 原子 900 的开始从 mp4 文件读取的数据字节的数量等于当前 trak 结构 3200 的成员“size”3202 的值时,满足到达了 trak 原子 900 的末端的条件。

[0548] 13. 操作 3880,如果已经到达了 trak 原子 900 的末端,则将 numTracks 的值增大 1。否则,重复操作 3810 到操作 3870。

[0549] 过程“处 track header 原子”包括下列步骤:

[0550] 1. 创建新的 TrackHeader 数据结构 3212。

[0551] 2. 将此新的 TrackHeader 数据结构 3212 的地址指定到当前 trak 结构 3200 中的成员 TrackHeader 3204。

[0552] 3. 将数量“atomSize”的值指定到 TrackHeader 结构 3212 中的成员“size”3214。

[0553] 4. 从 mp4 文件读取 32 位整数,并将结果指定给 TrackHeader 结构 3212 中的成员“version&flags”3216。

[0554] 5. 从 mp4 文件读取 32 位整数,并将结果指定给 TrackHeader 结构 3212 中的成员“creationTime”3218。

[0555] 6. 从 mp4 文件读取 32 位整数,并将结果指定给 TrackHeader 结构 3212 中的成员“modifiedTime”3220。

[0556] 7. 从 mp4 文件读取 32 位整数,并将此值指定给数量“trackID”。将数量“trackID”的值指定到 TrackHeader 结构 3212 的成员“trackID”3224。

[0557] 8. 从 mp4 文件读取 4 个字节,并忽略所产生的值。

[0558] 9. 从 mp4 文件读取 32 位整数,并将结果指定给 TrackHeader 结构 3212 的成员“duration”3222。

[0559] 10. 从 mp4 文件读取 60 个字节,并忽略所产生的值。

[0560] 如图 39 所示,过程“处理 track media 原子”包括下列步骤:

[0561] 1. 操作 3900,创建新的 TrackMedia 数据结构 3226。将此新的 TrackMedia 结构 3226 的地址指定到当前 trak 结构 3200 中的成员 TrackMedia 3206。将“atomSize”的值指定到新的 TrackMedia 结构 3226 中的成员“size”3228。

[0562] 2. 操作 3910,从 mp4 文件读取 32 位整数,并将此整数的值指定给数量“atomSize”。然后,从 mp4 文件读取第二个 32 位整数,并将此整数的值指定给数量“atomId”。

[0563] 3. 操作 3920,将数量“atomId”的值与值“mdhd”进行比较。

[0564] 4. 操作 3925,如果数量“atomId”的值对应于“mdhd”,则执行过程“处理媒体标头(mdhd)原子”。然后继续执行下面的操作 3970。

[0565] 5. 操作 3930,将数量“atomId”的值与值“hdlr”进行比较。

[0566] 6. 操作 3935,如果数量“atomId”的值对应于“hdlr”,则执行过程“处理 handler(hdlr)原子”。然后继续执行下面的操作 3970。

[0567] 7. 操作 3940,将数量“atomId”的值与值“minf”进行比较。

[0568] 8. 操作 3945,如果数量“atomId”的值对应于“minf”,则执行过程“处理 media

info(mdhd) 原子”。然后继续执行下面的操作 3970。

[0569] 9. 否则,执行操作 3960,跳过 mp4 文件中的 atomSize-8 字节。即,从 mp4 文件中读取此数量的字节,但是不对所产生的值执行任何操作。

[0570] 10. 操作 3970,检查是否到达 mdia 原子 912 的末端。当自 mdia 原子 912 的开始从 mp4 文件读取的数据字节的数量等于当前 TrackMedia 结构 3226 的成员“size”3228 的值时,满足到达了 mdia 原子 912 的末端的条件。

[0571] 11. 操作 3980,如果已经到达了 mdia 原子 912 的末端,则此过程完成,并继续执行操作 3870。否则,重复操作 3910 到操作 3970。

[0572] 过程“处理 track edit 原子”包括重复下列步骤,直到到达 edts 原子 945 的数据的末端。当自 edts 原子 945 的开始从 mp4 文件读取的数据字节的数量等于 edts 原子的大小时,满足到达了 edts 原子 945 的末端的条件。预计可以在 edts 原子 945 内找到单一的 elst 原子 948。

[0573] 1. 创建新的 TrackEdit 结构 3236。

[0574] 2. 将此新的 TrackEdit 结构 3236 的地址指定到当前 trak 结构 3200 的成员 TrackEdit 3208。

[0575] 3. 将“atomSize”的值指定到新的 TrackEdit 结构 3236 中的成员“size”3238。

[0576] 4. 从 mp4 文件读取 32 位整数,并将结果指定给 TrackEdit 结构 3236 的成员“version&flags”3240。

[0577] 5. 从 mp4 文件读取 32 位整数,并将此整数的值指定到数量“atomSize”。然后,从 mp4 文件读取第二个 32 位整数,并将此整数的值指定给数量“atomId”。

[0578] 6. 将数量“atomId”的值与值“elst”进行比较。

[0579] 7. 如果数量“atomId”的值对应于“elst”,则执行过程“处理 editlist 原子”。否则,跳过 mp4 文件中的 atomSize-8 字节。即,从 mp4 文件读取 atomSize-8 字节,并忽略所产生的值。

[0580] 8. 检查是否到达了 edts(track edit) 原子 945 的末端。当自 edts 原子 945 的开始从 mp4 文件读取的数据字节的数量等于 TrackEdit 结构 3236 的成员“size”3238 的值时,满足到达了 edts 原子 945 的末端的条件。

[0581] 9. 如果已经到达了 edts 原子 945 的末端,则此过程完成。否则,重复步骤 5 到 8。

[0582] 过程“处理 edit list 原子”包括下列步骤:

[0583] 1. 创建新的 EditList 结构 3244。

[0584] 2. 将此新的 EditList 结构 3244 的地址指定到当前 TrackEdit 结构 3236 的成员“EditList”3242。

[0585] 3. 将“atomSize”的值指定到新的 EditList 结构 3244 中的成员“size”3246。

[0586] 4. 从 mp4 文件读取 32 位整数,并将结果指定到 EditList 结构 3244 中的成员“version&flags”。

[0587] 5. 从 mp4 文件读取 32 位整数,并将此值指定到数量“numSegments”。

[0588] 6. 将数量“numSegments”的值指定到 EditList 结构 3244 的成员“numSegments”3250。

[0589] 7. 创建 numSegments EditSegment 结构 3256 的数组,并将此数组的地址指定到

EditList 结构 3244 的成员“EditSegmentarray”。

[0590] 8. 将零值指定到数量“segment”。

[0591] 9. 重复下列步骤 (10 到 13) numSegments 次 (直到数量“segment”的值等于数量“numSegments”的值)：

[0592] 10. 从 mp4 文件读取 32 位整数, 并将结果指定给 EditSegment 结构 3252 的数组中的成员“startTime”3258。

[0593] 11. 从 mp4 文件读取 32 位整数, 并将结果指定给 EditSegment 结构 3252 的数组中的条目“segment”的成员“duration”3260。

[0594] 12. 从 mp4 文件读取 32 位整数, 并将结果指定给 EditSegment 结构 3252 的数组中的条目“segment”的成员“relativeRate”3262。

[0595] 13. 将数量“segment”的值增大 1。

[0596] 过程“处理 track reference 原子”包括下列步骤。预计可以在 tref 原子 940 内找到单一的 mpod 原子 942。

[0597] 1. 创建新的 TrackReference 结构 3264。

[0598] 2. 将此新的 TrackReference 结构 3264 的地址指定到当前 trak 结构 3200 的成员“TrackReference”。

[0599] 3. 将 atomSize 的值指定到此新的 TrackReference 结构 3264 的成员“size”3266。

[0600] 4. 重复下列步骤, 直到到达 tref 原子 940 的数据的末端。当自 tref 原子 940 的开始从 mp4 文件读取的数据字节的数量等于 tref 原子 940 的大小时, 满足到达了 tref 原子 940 的末端的条件。

[0601] 5. 从 mp4 文件读取 32 位整数, 并将此整数的值指定给数量“atomSize”。然后, 从 mp4 文件读取第二个 32 位整数, 并将此整数的值指定给数量“atomId”。

[0602] 6. 将数量“atomId”的值与值“mpod”进行比较。

[0603] 7. 如果数量“atomId”的值对应于“mpod”, 则执行过程“处理 mpod 原子”。否则, 跳过 mp4 文件中的 atomSize-8 字节。即, 从 mp4 文件读取 atomSize-8 字节, 并忽略所产生的值。

[0604] 8. 检查是否到达了 tref(track reference) 原子 940 的末端。当自 tref 原子 940 的开始从 mp4 文件读取的数据字节的数量等于 TrackReference 结构 3264 的成员“size”3266 的值时, 满足到达了 tref 原子 940 的末端的条件。

[0605] 9. 如果已经到达了 tref 原子 940 的末端, 则此过程完成。否则, 重复步骤 5 到 8。

[0606] 过程“处理 mpod 原子”包括下列步骤：

[0607] 1. 创建新的 Mpod 数据结构 3270。

[0608] 2. 将此新的 Mpod 结构 3270 的地址指定到当前 TrackReference 结构 3264 的成员“Mpod”3268。

[0609] 3. 将 atomSize 的值指定到此新的 Mpod 结构的成员“size”3272。

[0610] 4. 将 Mpod 成员“size”的值减去 8, 并将结果除以 4, 然后, 将结果指定到数量“numEntries”。

[0611] 5. 将“numEntries”的值指定到新的 Mpod 数据结构 3270 的成员“numEntries”3274。

[0612] 6. 创建具有“numEntries”条目的整数的数组,并将此数组的地址指定到新的 Mpod 数据结构 3270 的成员“trackID array”3276。

[0613] 6. 将零值指定到数量“entry”。

[0614] 7. 重复下列步骤 (7 到 9) numEntries 次 (直到数量“entry”的值等于数量“numEntries”的值) :

[0615] 8. 从 mp4 文件读取 32 位整数,并将结果指定给当前 Mpod 数据结构 3270 中的数组“trackID array”3276 中的条目“entry”。

[0616] 9. 将数量“entry”的值增大 1。

[0617] 过程“处理 media header 原子”包括下列步骤 :

[0618] 1. 创建新的 MediaHeader 结构 3300。

[0619] 2. 将此新的 MediaHeader 结构 3300 的地址指定到当前 TrackMedia 结构 3226 中的成员“MediaHeader”3230。

[0620] 3. 将数量“atomSize”的值指定到 MediaHeader 结构 3300 中的成员“size”3303。

[0621] 4. 从 mp4 文件读取 32 位整数,并将结果指定给 MediaHeader 结构 3300 的成员“version&flags”3306。

[0622] 5. 从 mp4 文件读取 32 位整数,并将结果指定给 MediaHeader 结构 3300 的成员“creationTime”3308。

[0623] 6. 从 mp4 文件读取 32 位整数,并将结果指定给 MediaHeader 结构 3300 的成员“modifiedTime”3310。

[0624] 7. 从 mp4 文件读取 32 位整数,并将结果指定给 MediaHeader 结构 3300 的成员“timeScale”3313。

[0625] 8. 从 mp4 文件读取 32 位整数,并将结果指定给 MediaHeader 结构 3300 的成员“duration”3316。

[0626] 9. 从 mp4 文件读取 16 位整数,并将结果指定给 MediaHeader 结构 3300 的成员“language”3318。

[0627] 10. 读取 mp4 文件中的 2 个字节并忽略所产生的值。

[0628] 过程“处理 handler 原子”包括下列步骤 :

[0629] 1. 创建新的 Handler 结构 3320。

[0630] 2. 将此新的 Handler 结构 3320 的地址指定到当前 TrackMedia 结构 3226 中的成员“Handler”3232。

[0631] 3. 将数量“atomSize”的值指定到 Handler 结构 3320 中的成员“size”3323。

[0632] 4. 从 mp4 文件读取 32 位整数,并将结果指定给 Handler 结构 3300 的成员“version&flags”3326。

[0633] 5. 从 mp4 文件读取 32 位整数,并将结果指定给 Handler 结构 3300 的成员“handlerType”3328。

[0634] 8. 读取 mp4 文件中的 12 个字节并忽略所产生的值。

[0635] 9. 从 mp4 文件读取空值结束的字符串,将所产生的字符的序列存储在字符的数组中,并将此字符的数组的地址指定到 Handler 结构 3320 中的成员“name”3330。

[0636] 如图 40 所示,过程“处理 media info 原子”包括下列步骤 :

[0637] 1. 操作 4000, 创建新的 MediaInfo 数据结构 3333。将此新的 MediaInfo 数据结构 3333 的地址指定到当前 TrackMedia 数据结构 3226 中的成员 MediaInfo 3234。将数量“atomSize”的值指定到 MediaInfo 数据结构 3333 中的成员“size”3336。

[0638] 2. 操作 4010, 从 mp4 文件读取 32 位整数, 并将此整数的值指定到数量“atomSize”。然后, 从 mp4 文件读取第二个 32 位整数, 并将此整数的值指定给数量“atomId”。

[0639] 3. 操作 4020, 将数量“atomId”的值与值“dinf”进行比较。

[0640] 4. 操作 4025, 如果数量“atomId”的值对应于“dinf”, 则执行过程“Process media data info(dinf) 原子”。然后继续执行操作 4080。

[0641] 5. 操作 4030, 将数量“atomId”的值与值“stbl”进行比较。

[0642] 6. 操作 4035, 如果数量“atomId”的值对应于“stbl”, 则执行过程“处理样本表原子”。然后继续执行操作 4080。

[0643] 7. 操作 4040, 将数量“atomId”的值与值“nmhd”进行比较。

[0644] 8. 操作 4035, 如果数量“atomId”的值对应于“nmhd”, 则执行过程“处理 mpeg4 媒体标头原子”。然后继续执行操作 4080。

[0645] 9. 操作 4050, 将数量“atomId”的值与值“smhd”进行比较。

[0646] 10. 操作 4055, 如果数量“atomId”的值对应于“smhd”, 则执行过程“处理声音媒体标头原子”。然后继续执行操作 4080。

[0647] 11. 操作 4060, 将数量“atomId”的值与值“vmhd”进行比较。

[0648] 12. 操作 4065, 如果数量“atomId”的值对应于“vmhd”, 则执行过程“处理视频媒体标头原子”。然后继续执行操作 4080。

[0649] 13. 否则, 执行操作 4070, 跳过 mp4 文件中的 atomSize-8 字节。即, 从 mp4 文件读取 atomSize-8 字节, 并忽略所产生的值。

[0650] 14. 操作 4080, 检查是否到达了 minf 原子 920 的末端。当自 minf 原子 920 的开始从 mp4 文件读取的数据字节的数量等于 MediaInfo 数据结构 3333 中的成员“size”3336 的值时, 满足到达了 minf 原子 920 的末端的条件。

[0651] 15. 操作 4090, 如果已经到达 minf 原子 920 的末端, 则完成该过程。否则, 重复操作 4010 到操作 4080。

[0652] 过程“处理媒体数据信息 (dinf) 原子”包括下列步骤:

[0653] 1. 创建新的 MediaDataInfo 结构 3350。

[0654] 2 将此新的 MediaDataInfo 结构 3350 的地址指定到当前 MediaInfo 结构 3333 中的成员“MediaDataInfo”3338。

[0655] 3. 将数量“atomSize”的值指定到 MediaDataInfo 结构 3350 中的成员“size”3353。

[0656] 4. 从 mp4 文件读取 32 位整数, 并将结果指定给 MediaDataInfo 结构 3350 的成员“version&flags”3356。

[0657] 5. 重复下列步骤, 直到到达 dinf 原子 924 的数据的末端。当自 dinf 原子 924 的开始从 mp4 文件读取的数据字节的数量等于 MediaDataInfo 结构 3350 的成员“size”的值时, 满足到达了 dinf 原子 924 的末端的条件。

- [0658] 6. 从 mp4 文件读取 32 位整数,并将此整数的值指定给数量“atomSize”。
- [0659] 7. 从 mp4 文件读取 32 位整数,并将此整数的值指定给数量“atomId”。
- [0660] 8. 如果数量“atomId”的值对应于“dref”,则执行过程“处理数据引用表原子”。否则,跳过 mp4 文件中的 atomSize-8 字节。即,读取 mp4 文件中的 atomSize-8 字节,并忽略所产生的值。
- [0661] 9. 检查是否到达了 dinf 原子 924 的末端。如果已经到达 dinf 原子 924 的末端,则完成该过程。否则,重复步骤 2 到 9。
- [0662] 过程“处理数据引用表原子”包括下列步骤:
- [0663] 1. 创建新的 DataRefTable 数据结构 3360。
- [0664] 2. 将此新的 DataRefTable 数据结构 3360 的地址指定到当前 MediaDataInfo 数据结构 3350 中的成员 DataRefTable 3358。
- [0665] 3. 将数量“atomSize”的值指定到 DataRefTable 数据结构 3360 中的成员“size”3363。
- [0666] 4. 从 mp4 文件读取 32 位整数,并将结果指定给 DataRefTable 结构 3360 的成员“version&Flags”3366。
- [0667] 6. 从 mp4 文件读取 32 位整数,并将此值指定给数量“numDataRefs”。
- [0668] 7. 将数量“numDataRefs”的值指定到 DataRefTable 结构 3360 中的成员“numDataRefs”3368。
- [0669] 8. 创建具有“numDataRefs”条目的地址的数组,并将此数组的地址指定到新的 DataRefTable 数据结构 3360 中的成员“dataRefarray”3370。
- [0670] 9. 将零值指定到数量“entry”。
- [0671] 10. 重复下列步骤 numDataRefs 次(直到数量“entry”的值等于数量“numDataRefs”的值)。
- [0672] 11. 创建新的 dataRef 数据结构 3380。
- [0673] 12. 将此新的 dataRef 数据结构 3380 的地址指定到 DataRefTable 数据结构 3360 的 dataRef 数组成员 3370 中的条目“entry”。
- [0674] 13. 从 mp4 文件读取 32 位整数,并将值指定到数量“drefSize”。
- [0675] 14. 将数量“drefSize”的值指定到此新的 dataRef 数据结构 3380 的成员“size”3383。
- [0676] 15. 从 mp4 文件读取 32 位整数,并将结果指定给当前 dataRef 数据结构 3380 中的成员“version&Flags”3386。
- [0677] 16. 从 mp4 文件读取 32 位整数,并将结果指定给此新的 dataRef 数据结构 3380 的成员“dataRefType”3388。数量“type”的值可以解释为四个文本字符的序列。
- [0678] 17. 创建(drefSize-12)字节的数组,并将此数组的地址指定到当前 DataRef 数据结构 3380 中的成员“dataRefData”3390。
- [0679] 18. 从 mp4 文件读取(drefSize-12)字节的序列,并这些复制到数组“dataRefData”3390 中的连续的条目。
- [0680] 19. 将数量“entry”的值增大 1。
- [0681] 过程“处理 mpeg4 媒体标头原子”包括下列步骤:

[0682] 1. 创建新的 Mpeg4MediaHeader 数据结构。Mpeg4MediaHeader 数据结构包含成员“size”和“version&flags”。

[0683] 2. 将此新的 Mpeg4MediaHeader 数据结构的地址指定到当前 MediaInfo 数据结构 3333 的成员 MediaInfoHeader 3343。

[0684] 3. 将数量“atomSize”的值指定到此新的 Mpeg4MediaHeader 数据结构的成员“size”。

[0685] 4. 从 mp4 文件读取 32 位整数,并将结果指定给此新的 Mpeg4MediaHeader 数据结构的成员“verFlags”。

[0686] 5. 跳过 mpeg4 媒体标头 (nmhd) 原子 936 内包含的任何其余数据 (如果 atomSize 大于 12)。

[0687] 过程“处理声音媒体标头原子”包括下列步骤:

[0688] 1. 创建新的 SoundMediaHeader 数据结构。SoundMediaHeader 数据结构包含成员“size”、“version&flags”和“balance”。

[0689] 2. 将此新的 SoundMediaHeader 数据结构的地址指定到当前 MediaInfo 数据结构 3333 的成员 MediaInfoHeader 3343。

[0690] 3. 将数量“atomSize”的值指定到此新的 SoundMediaHeader 数据结构的成员“size”。

[0691] 4. 从 mp4 文件读取 32 位整数,并将结果指定给新的 SoundMediaHeader 结构中的成员“version&Flags”。

[0692] 5. 从 mp4 文件读取 32 位整数,并将结果指定给当前 SoundMediaHeader 结构中的成员“balance”。

[0693] 6. 跳过声音媒体标头 (smhd) 原子 936 内包含的任何其余数据 (如果 atomSize 大于 16)。

[0694] 过程“处理视频媒体标头原子”包括下列步骤:

[0695] 1. 创建新的 VideoMediaHeader 数据结构。VideoMediaHeader 数据结构包含成员“size”、“version&flags”、“mode”、“opRed”、“opGreen”,以及“opBlue”。

[0696] 2. 将此新的 VideoMediaHeader 数据结构的地址指定到当前 MediaInfo 数据结构 3333 的成员 MediaInfoHeader 3343。

[0697] 3. 将 atomSize 的值指定到此新的 VideoMediaHeader 结构的成员“size”。

[0698] 4. 从 mp4 文件读取 32 位整数,并将结果指定给新的 VideoMediaHeader 结构中的成员“version&flags”。

[0699] 6. 从 mp4 文件读取 16 位整数,并将结果指定给新的 VideoMediaHeader 结构中的成员“mode”。

[0700] 7. 从 mp4 文件读取 16 位整数,并将结果指定给当前 VideoMediaHeader 结构中的成员“opRed”。

[0701] 8. 从 mp4 文件读取 16 位整数,并将结果指定给当前 VideoMediaHeader 结构中的成员“opGreen”。

[0702] 9. 从 mp4 文件读取 16 位整数,并将结果指定给当前 VideoMediaHeader 结构中的成员“opBlue”。

[0703] 10. 跳过视频媒体标头 (vmhd) 原子 936 内包含的任何其余数据 (如果 atomSize 大于 20)。

[0704] 如图 41 所示,过程“处理样本表原子”包括下列步骤:

[0705] 1. 操作 4100,创建新的 SampleTables 数据结构 3400。将此新的 SampleTables 数据结构 3400 的地址指定到当前 MediaInfo 数据结构 3333 的成员 SampleTables 3340。将“atomSize”的值指定到新的 SampleTables 数据结构 3400 中的成员“size”3403。

[0706] 2. 操作 4110,从 mp4 文件读取 32 位整数,并将此整数的值指定给数量“atomSize”。然后,从 mp4 文件读取第二个 32 位整数,并将此整数的值指定给数量“atomId”。

[0707] 3. 操作 4120,将数量“atomId”的值与值“stsc”进行比较。

[0708] 4. 操作 4125,如果数量“atomId”的值对应于“stsc”,则执行过程“处理样本到块表 (stsc. 原子)”。然后继续执行操作 4190。

[0709] 5. 操作 4130,将数量“atomId”的值与值“stco”进行比较。

[0710] 6. 操作 4135,如果数量“atomId”的值对应于“stco”,则执行过程“处理块偏移表 (stco) 原子”。然后继续执行操作 4190。

[0711] 7. 操作 4140,将数量“atomId”的值与值“stts”进行比较。

[0712] 8. 操作 4145,如果数量“atomId”的值对应于“stts”,则执行过程“处理时间到样本表 (stts) 原子”。然后继续执行操作 4190。

[0713] 9. 操作 4150,将数量“atomId”的值与值“stsz”进行比较。

[0714] 10. 操作 4155,如果数量“atomId”的值对应于“stsz”,则执行过程“处理样本大小表 (stsz) 原子”。然后继续执行操作 4190。

[0715] 11. 操作 4160,将数量“atomId”的值与值“stss”进行比较。

[0716] 12. 操作 4165,如果数量“atomId”的值对应于“stss”,则执行过程“处理同步样本表 (stss) 原子”。然后继续执行操作 4190。

[0717] 13. 操作 4170,将数量“atomId”的值与值“std”进行比较。

[0718] 14. 操作 4175,如果数量“atomId”的值对应于“std”,则执行过程“处理样本描述表 (std) 原子”。然后继续执行操作 4190。

[0719] 15. 否则,执行操作 4180,跳过 mp4 文件中的 atomSize-8 字节。即,从 mp4 文件读取 atomSize-8 字节,并忽略所产生的值。

[0720] 16. 操作 4190,检查是否到达了 stbl 原子 950 的末端。当自 stbl 原子 950 的开始从从 mp4 文件读取的数据字节的数量等于 SampleTables 数据结构 3400 中的成员“size”3403 的值时,满足到达了 stbl 原子 950 的末端的条件。

[0721] 17. 操作 4195,如果已经到达 stbl 原子 950 的末端,则完成该过程。否则,重复操作 4110 到操作 4190。

[0722] 过程“处理样本到块表 (stsc. 原子)”包括下列步骤:

[0723] 1. 创建新的样本表数据结构 3420。

[0724] 2. 将此新的样本表数据结构 3420 的地址指定到当前 SampleTables 数据结构 3400 中的成员 SampleToChunkTable3406。

[0725] 3. 将数量“atomSize”的值指定到新的样本表数据结构 3420 中的成员

“size” 3423。

[0726] 4. 从 mp4 文件读取 32 位整数, 并将结果指定给新的样本表数据结构 3420 中的成员 “version&Flags” 3426。

[0727] 5. 从 mp4 文件读取 32 位整数, 并将结果指定给新的样本表数据结构 3420 中的成员 “numEntries” 3428。

[0728] 6. 创建具有 numEntries 条目的地址的数组, 并将此数组的地址指定到新的样本表数据结构 3420 中的成员 “table array” 3430。

[0729] 7. 将零值指定到数量 “entry”。

[0730] 8. 重复下列步骤 numEntries 次 (直到数量 “entry” 的值等于成员 “numEntries” 3428 的值)。

[0731] 9. 创建新的 SampleToChunk 数据结构 3460。

[0732] 10. 将此新的 SampleToChunk 数据结构 3460 的地址指定到新的样本表数据结构 3420 的成员 “table array” 3430 中的条目 “entry”。

[0733] 13. 从 mp4 文件读取 32 位整数, 并将结果指定给新的 SampleToChunk 数据结构 3460 中的成员 “firstChunk” 3463。

[0734] 14. 从 mp4 文件读取 32 位整数, 并将结果指定给 SampleToChunk 数据结构 3460 中的成员 “numSamples” 3466。

[0735] 15. 从 mp4 文件读取 32 位整数, 并将结果指定给 SampleToChunk 数据结构 3460 的成员 “description” 3468。

[0736] 16. 将数量 “entry” 的值增大 1。

[0737] 过程 “处理块偏移表 (stco) 原子” 包括下列步骤:

[0738] 1. 创建新的样本表数据结构 3420。

[0739] 2. 将此新的样本表数据结构 3420 的地址指定到当前 SampleTables 数据结构 3400 中的成员 ChunkOffsetTable 3408。

[0740] 3. 将数量 “atomSize” 的值指定到新的样本表数据结构 3420 中的成员 “size” 3423。

[0741] 4. 从 mp4 文件读取 32 位整数, 并将结果指定给新的样本表数据结构 3420 中的成员 “version&Flags” 3426。

[0742] 5. 从 mp4 文件读取 32 位整数, 并将结果指定给新的样本表数据结构 3420 中的成员 “numEntries” 3428。

[0743] 6. 创建具有 numEntries 条目的整数的数组, 并将此数组的地址指定到新的样本表数据结构 3420 中的成员 “table array” 3430。

[0744] 7. 将零值指定到数量 “entry”。

[0745] 8. 重复下列步骤 numEntries 次 (直到数量 “entry” 的值等于成员 “numEntries” 3428 的值)。

[0746] 9. 从 mp4 文件读取 32 位整数, 并将结果指定给新的样本表数据结构 3420 中的数组 “table array” 3430 中的条目 “entry”。

[0747] 10. 将数量 “entry” 的值增大 1。

[0748] 过程 “处理时间到样本表 (stts) 原子” 包括下列步骤:

- [0749] 1. 创建新的样本表数据结构 3420。
- [0750] 2. 将此新的样本表数据结构 3420 的地址指定到当前 SampleTables 数据结构 3400 中的成员 TimeToSampleTable 3410。
- [0751] 3. 将数量“atomSize”的值指定到新的样本表数据结构 3420 中的成员“size”3423。
- [0752] 4. 从 mp4 文件读取 32 位整数,并将结果指定给新的样本表数据结构 3420 中的成员“version&Flags”3426。
- [0753] 5. 从 mp4 文件读取 32 位整数,并将结果指定给新的样本表数据结构 3420 中的成员“numEntries”3428。
- [0754] 6. 创建具有 numEntries 条目的地址的数组,并将此数组的地址指定到新的样本表数据结构 3420 中的成员“table array”3430。
- [0755] 7. 将零值指定到数量“entry”。
- [0756] 8. > 重复下列步骤 numEntries 次(直到数量“entry”的值等于成员“numEntries”3428 的值)。
- [0757] 9. 创建新的 TimeToSample 数据结构 3470。
- [0758] 10. 将此新的 TimeToSample 数据结构 3470 的地址指定到样本表数据结构 3420 的成员“table array”3430 中的条目“entry”。
- [0759] 11. 创建新的 TimeToSample 数据结构 3470。
- [0760] 12. 将此新的 TimeToSample 数据结构 3470 的地址指定到样本表数据结构 3420 中的数组““table array”3430 中的条目“entry”。
- [0761] 13. 从 mp4 文件读取 32 位整数,并将结果指定给新的 TimeToSample 数据结构 3470 中的成员“count”3473。
- [0762] 14. 从 mp4 文件读取 32 位整数,并将结果指定给新的 TimeToSample 数据结构 3470 中的成员“duration”3476。
- [0763] 15. 将数量“entry”的值增大 1。
- [0764] 过程“处理样本大小表(stsz)原子”包括下列步骤:
- [0765] 1. 创建新的 SampleSizeTable 数据结构 3440。
- [0766] 2. 将此新的 SampleSizeTable 结构 3440 的地址指定到当前 SampleTables 数据结构 3400 中的成员 SampleSizeTable 3413。
- [0767] 3. 将数量“atomSize”的值指定到新的 SampleSizeTable 数据结构 3440 中的成员“size”3443。
- [0768] 4. 从 mp4 文件读取 32 位整数,并将结果指定给新的 SampleSizeTable 数据结构 3440 的成员“version&Flags”3446。
- [0769] 5. 从 mp4 文件读取 32 位整数,并将结果指定给新的 SampleSizeTable 数据结构 3440 中的成员“sampleSize”3450。如果新的 SampleSizeTable 数据结构 3440 中的成员 sampleSize 3450 的值为零,则执行下列步骤,否则,将零值指定给新的 SampleSizeTable 数据结构 3440 中的成员“numEntries”3453 和“table array”3456。
- [0770] 6. 从 mp4 文件读取 32 位整数,并将结果指定给新的 SampleSizeTable 数据结构 3440 中的成员“numEntries”3453。

[0771] 7. 创建具有 numEntries 条目的整数的数组, 并将此数组的地址指定到新的 SampleSizeTable 数据结构 3440 中的成员“tablearray”3456。

[0772] 8. 将零值指定到数量“entry”。

[0773] 9. 重复下列步骤 numEntries 次(直到数量“entry”的值等于成员“numEntries”3453 的值)。

[0774] 10. 从 mp4 文件读取 32 位整数, 并将结果指定给 SampleSizeTable 数据结构 3440 中的成员“table array”3456 中的条目“entry”。

[0775] 11. 将数量“entry”的值增大 1。

[0776] 过程“处理同步样本表(stss)原子”包括下列步骤:

[0777] 1. 创建新的样本表数据结构 3420。

[0778] 2. 将此新的样本表数据结构 3420 的地址指定到当前 SampleTables 数据结构 3400 中的成员 SyncSampleTable 3416。

[0779] 3. 将数量“atomSize”的值指定到新的样本表数据结构 3420 中的成员“size”3423。

[0780] 4. 从 mp4 文件读取 32 位整数, 并将结果指定给新的样本表数据结构 3420 中的成员“version&Flags”3426。

[0781] 5. 从 mp4 文件读取 32 位整数, 并将结果指定给新的样本表数据结构 3420 中的成员“numEntries”3428。

[0782] 6. 创建具有 numEntries 条目的整数的数组, 并将此数组的地址指定到新的样本表数据结构 3420 中的成员“table array”3430。

[0783] 7. 将零值指定到数量“entry”。

[0784] 8. 重复下列步骤 numEntries 次(直到数量“entry”的值等于成员“numEntries”3428 的值)。

[0785] 9. 从 mp4 文件读取 32 位整数, 并将结果指定给新的样本表数据结构 3420 中的数组“table array”3430 中的条目“entry”。

[0786] 10. 将数量“entry”的值增大 1。

[0787] 过程“处理样本描述表(stsd)原子”包括下列步骤:

[0788] 1. 创建新的样本表数据结构 3420。

[0789] 2. 将此新的样本表数据结构 3420 的地址指定到当前 SampleTables 数据结构 3400 中的成员 SampleDescriptionTable3418。

[0790] 3. 将数量“atomSize”的值指定到新的样本表数据结构 3420 中的成员“size”3423。

[0791] 4. 从 mp4 文件读取 32 位整数, 并将结果指定给新的样本表数据结构 3420 中的成员“version&Flags”3426。

[0792] 5. 从 mp4 文件读取 32 位整数, 并将结果指定给新的样本表数据结构 3420 中的成员“numEntries”3428。

[0793] 6. 创建具有 numEntries 条目的地址的数组, 并将此数组的地址指定到新的样本表数据结构 3420 中的成员“table array”3430。

[0794] 7. 将零值指定到数量“entry”。

[0795] 8. 重复下列步骤 numEntries 次（直到数量“entry”的值等于成员“numEntries”3428 的值）。

[0796] 9. 创建新的 SampleDescription 数据结构 3480。

[0797] 10. 将此新的 SampleDescription 数据结构 3480 的地址指定到新的样本表数据结构 3420 的成员“table array”3430 中的条目“entry”。

[0798] 11. 从 mp4 文件读取 32 位整数，并将结果指定给 SampleDescription 数据结构 3480 中的成员“size”3483。

[0799] 12. 从 mp4 文件读取 32 位整数，并将结果指定给 SampleDescription 数据结构 3480 中的成员“dataFormat”3486。成员“dataFormat”的值可以表示下列 4 字符序列之一：“mp4a”、“mp4v”、“mp4s”。

[0800] 13. 从 mp4 文件读取 6 个字节，并忽略它们的值。

[0801] 14. 从 mp4 文件读取 16 位整数，并将结果指定给 SampleDescription 数据结构 3480 中的成员“dataReference”3488。

[0802] 15. 如果数量“dataFormat”的值对应于字符串“mp4a”，则从 mp4 文件读取 20 个字节，并忽略它们的值。

[0803] 16. 如果数量“dataFormat”的值对应于字符串“mp4v”，则从 mp4 文件读取 70 个字节，并忽略它们的值。

[0804] 17. 从 mp4 文件读取 32 位整数，并将结果指定给数量“atomSize”。

[0805] 18. 从 mp4 文件读取 32 位整数，并将此值指定到数量“atomId”。

[0806] 19. 如果数量“atomId”的值对应于“esds”，则执行过程“处理 esds 原子”。否则，从 mp4 文件读取 (atomSize-8) 个字节，并忽略它们的值。

[0807] 过程“处理 esds 原子”包括下列步骤：

[0808] 1. 从 mp4 文件读取 32 位整数，并将结果指定给 SampleDescription 数据结构 3480 中的成员“esdVersion”3490”。

[0809] 2. 将数量“atomSize”的值减去 12，并将结果指定到当前 SampleDescription 数据结构 3480 中的成员“esdSize”3493。

[0810] 4. 创建具有 esdSize 3493 条目的字节的数组，并将此数组的地址指定到新当前 SampleDescription 数据结构 3480 中的成员“esdData”3496。

[0811] 5. 从 mp4 文件读取 esdSize 3496 字节的序列，并将这些值复制到数组“esdData”3496 中的连续的条目。

[0812] 基于 QtInfo 数据结构创建中间 xml 文档和媒体数据文件。

[0813] 在基于 mp4 二进制文件 700、2210 创建 QtInfo 数据结构 3100 之后，使用 QtInfo 数据结构 3100 创建一组媒体数据文件 2230 和一对中间 xml 文档 2250、2260。这些 xml 文档包括“mp4file”文档 2250、2300 和“mp4bifs”文档 2260、2800。mp4file 文档 2250、2300 的结构在图 23 到 27 中表示。Mp4bifs 文档 2260、2800 的结构在图 28 到 30 中表示。

[0814] 此过程被分成两个步骤：

[0815] 1. 构成以 QtInfo 数据结构 3100 表示 3113、3160 的媒体数据 (mdat) 原子 706 的 xml 表示 2310，并将关联的媒体数据 3168 存储在 xml 文档或二进制媒体数据文件 2230 中。

[0816] 2. 构成以 QtInfo 数据结构 3100 表示的 moov 原子 712 的 xml 表示 2320。

[0817] 下面将描述这些步骤中的每一个步骤。

[0818] 步骤 1. 构成以 QtInfo 数据结构表示的媒体数据原子的 xml 表示, 并将关联的媒体数据存储在 xml 文档或二进制媒体数据文件中。

[0819] 在此步骤中, 以 QtInfo 数据结构 3100 表示的每一个 mdat 原子 706 都通过 mp4file 文档 2300 中的 mdat 元素 2310 来表示。将与音频或视频流 (如果有的话) 关联的媒体数据 3168 传输到新的二进制媒体数据文件 2230。与场景描述流 (sdsm 或 BIFS) 关联的媒体数据 3168 以 mp4bifs 文档 2260、2800 的形式来表示。与对象描述符流 (如果有的话) 关联的媒体数据 3168 以 mp4file 文档 2300 内的 xml 形式来表示。

[0820] 如图 42 所示, 此步骤用下列过程来完成:

[0821] 1. 在操作 4200 中, 给数量“iMdat”、“numVisual”, 以及“numAudio”指定零值。

[0822] 2. 操作 4210, 将数量“iMdat”的值与 QtInfo 结构 3100 的成员“numMdatAtoms”3110 的值进行比较。

[0823] 3. 在操作 4220 中, 如果数量“iMdat”的值等于成员“numMdatAtoms”的值, 则此步骤完成。否则, 继续执行下列过程。

[0824] 4. 操作 4230, 将 QtInfo 结构 3100 的成员“mdat 数组”3113 中的条目“iMdat”的值指定到表示 mdat 数据结构 3160 的地址的数量“mdatAtom”。

[0825] 5. 操作 4240, 将 mdat 数据结构“mdatAtom”3160 的成员“start”3166 的值指定到数量“mdatStart”。

[0826] 6. 操作 4250, 将 mdatAtom 成员“size”3163 和数量“mdatStart”的值的总和指定到数量“nextMdat”。

[0827] 7. 操作 4260, 创建新的“mdat”元素 2310, 并将它附加到 mp4file 文档 2300 的最高一级的元素。它将数量“iMdat”的值指定到新的 mdat 元素的属性“mdatID”, 将数量“mdatSize”的值指定到新的 mdat 元素的属性“size”, 并将 mdat 数据结构“mdatAtom”的成员“MediaData”的值指定到数量“mediaData”。

[0828] 8. 操作 4270, 执行过程“处理媒体数据”。

[0829] 9. 操作 4280, 将数量“iMdat”的值增大 1, 然后重复操作 4210。

[0830] 过程“处理媒体数据”

[0831] 如图 43 所示, 过程“处理媒体数据”4270 包括下列步骤:

[0832] 1. 操作 4300, 将零值指定到数量“iTrack”。

[0833] 2. 操作 4310, 将数量“iTrack”的值与 QtInfo 结构 3100 的成员“numTracks”3118 的值进行比较。

[0834] 3. 操作 4315, 如果数量“iTrack”的值等于 QtInfo 结构 3100 的成员“numTracks”的值, 则此过程完成。

[0835] 4. 操作 4320, 将 QtInfo 结构 3100 的成员“trak 数组”3120 中的条目“iTrack”的值指定到 trak 数据结构 3200 “trakAtom”的地址。将数据结构“trakAtom”3200 中的成员 TrackMedia 3206 中的成员 MediaInfo 3234 中的成员 SampleTables 3340 中的成员 ChunkOffsetTable 3408 中的成员“table array”3430 指定的整数的数组中的第一条目 (条目 0) 的值指定到数量“chunkOffset”。

[0836] 5. 操作 4315, 如果数量“chunkOffset”的值大于零, 大于或等于数量“mdatStart”

的值,小于数量“nextMdat”的值,则执行下列步骤。否则,将数量“iTrack”的值增大 1(操作 4390)。

[0837] 6. 操作 4340,执行过程“查找流类型”。

[0838] 7. 操作 4350,将数量“streamType”的值与值“1”进行比较。

[0839] 8. 操作 4355,如果数量“streamType”的值为“1”,则执行过程“OdsMDataToXml”。

[0840] 9. 操作 4360,将数量“streamType”的值与值“3”进行比较。

[0841] 10. 操作 4365,如果数量“streamType”的值为“3”,则执行过程“SdsMDataToXml”。

[0842] 11. 操作 4370,将数量“streamType”的值与值“4”进行比较。

[0843] 12. 操作 4360,如果数量“streamType”的值为“4”,则执行过程“VisualDataToXml”,然后将“numVisual”的值增大 1。

[0844] 13. 操作 4380,将数量“streamType”的值与值“5”进行比较。

[0845] 14. 操作 4385,如果数量“streamType”的值为“5”,则执行过程“AudioDataToXml”,然后将“numAudio”的值增大 1。

[0846] 15. 操作 4390,将数量“iTrack”的值增大 1。

[0847] 过程“查找流类型”

[0848] 如图 44 所示,过程“查找流类型”包括下列步骤:

[0849] 1. 操作 4400,将数据结构“trakAtom”中的成员 TrackMedia3206 中的成员 MediaInfo 3234 中的成员 SampleTables 3340 中的成员 SampleDescrTable 3418 的值指定到表示样本表结构 3420 的地址的数量“stsdAtom”。

[0850] 2. 操作 4410,将由数量“stsdAtom”的值指定的样本表数据结构 3420 的成员“table array”3430 中的第一条目(条目 0)指定的 SampleDescription 数据结构 3480 中的成员“esdData”3496 的值指定到表示字节的数组的数量“esdData”。

[0851] 3. 操作 4420,将由数量“stsdAtom”的值指定的样本表数据结构 3420 的成员“table array”中的第一条目(条目 0)指定的 SampleDescription 数据结构 3480 中的成员“esdSize”的值指定到数量“esdSize”。

[0852] 4. 操作 4430,将零值指定到数量“esdPos”。

[0853] 5. 操作 4440,执行过程“查找解码器配置描述符”。

[0854] 6. 操作 4450,执行过程“获取标记大小”。

[0855] 7. 操作 4460,将数组“esdData”中的条目“esdPos”的值指定到数量“objectType”。

[0856] 9. 操作 4470,将数量“esdPos”的值增大 1。

[0857] 10. 操作 4480,获取数组“esdData”中的条目“esdPos”的值,将结果向右移动两个位(除以 4),并将结果指定到数量“streamType”。

[0858] 如果数量“esdPos”的值在步骤 5 到 10 的过程中等于或超过数量“esdSize”的值,则将零值指定到数量“StreamType”和“ObjectType”,并终止此过程。

[0859] 过程“查找解码器配置描述符”

[0860] 过程“查找解码器配置描述符”,如图 45 所示,包括下列步骤:

[0861] 1. 操作 4500,将数组“esdData”中的条目“esdPos”的值指定到数量“esdTag”。

[0862] 2. 操作 4510,将数量“esdTag”的值与值“3”进行比较。

[0863] 3. 操作 4520, 如果数量“esdTag”的值不是“3”, 则将数量“esdSize”的值指定到数量“esdPos”, 并结束此过程 4525。否则, 执行下列步骤。

[0864] 4. 操作 4530, 将数量“esdPos”的值增大 1。

[0865] 5. 操作 4535, 执行过程“获取标记大小”。

[0866] 6. 操作 4540, 将数量“esdPos”的值增大 4。

[0867] 7. 操作 4550, 将数组“esdData”中的条目“esdPos”的值指定到数量“esdTag”。

[0868] 8. 操作 4560, 将数量“esdTag”的值与值“4”进行比较。

[0869] 9. 操作 4565, 如果数量“esdTag”的值不是“4”, 则将数量“esdPos”的值增大 1, 执行过程“获取标记大小” 4570, 将数量“tagSize”的值与数量“esdPos” 4575 的值相加, 然后重复操作 4550。

[0870] 10. 否则, 执行操作 4580, 将数量“esdPos”的值增大 1。

[0871] 11. 操作 4585, 完成此过程, 并继续执行过程“查找流类型”。

[0872] 过程“获取标记大小”

[0873] 过程“获取标记大小”包括下列步骤:

[0874] 1. 操作 4600, 将零值指定到数量“tagSize”。

[0875] 2. 操作 4610, 将数组“esdData”中的条目“esdPos”的值指定到数量“sizeByte”。

[0876] 3. 操作 4620, 将数量“sizeByte”的值与值 128 进行比较。如果数量“sizeByte”的值大于或等于 128, 则执行下列操作:

[0877] 5. 操作 4630, 将数量“sizeByte”的值减去 128 (或将数量“sizeByte”的值替换为此值的下 7 位),

[0878] 6. 操作 4640, 将数量“tagSize”的值乘以 128 (或将此值向左移 7 位),

[0879] 7. 操作 4650, 将数量“sizeByte”的值与数量“tagSize”的值相加,

[0880] 8. 操作 4660, 将数量“esdPos”的值增大值“1”。

[0881] 9. 重复操作 4610。如果数量“sizeByte”的值小于 128, 则执行下列操作:

[0882] 10. 操作 4670, 将数量“tagSize”的值乘以 128 (向左移 7 位)。

[0883] 11. 操作 4680, 将数量“sizeByte”的值与数量“tagSize”的值进行比较。

[0884] 12. 操作 4690, 将数量“esdPos”的值增大值“1”。

[0885] 13. 操作 4695, 完成此过程, 并延续执行调用此过程的过程。

[0886] 过程“OdsmdDataToXml”

[0887] 过程“OdsmdDataToXml” 4355 创建 mdat 结构“mdatAtom” 4230 内包含的 odsmd 数据流 (对象描述符流) 的 xml 表示。如图 47 所示, 使用下列操作来创建此 odsmd 数据流的 xml 表示:

[0888] 1. 操作 4700, 将 mdat 结构“mdatAtom” 4230 的成员“MediaData” 3168 的值指定到表示字节的数组的地址的数量“OdsmdData”。将 mdat 结构“mdatAtom” 4230 的成员“size” 3163 的值指定到整型量“OdsmdSize”。

[0889] 2. 操作 4705, 创建新的“odsmd”元素 2460, 并附加到当前 mdat 元素 2400、4260。将 trak 结构 3200 “trakAtom” 4320 内包含的 TrackHeader 结构 3212、3204 的成员 trackID 3224 的值指定到新的“odsmd”元素的属性“trackID”的值。

[0890] 3. 操作 4710, 将样本表结构 3420ChunkOffsetTable 3408 的成员 numEntries

3428 的值指定到数量“numChunks”。将样本表结构 3420SampleToChunkTable 3406 的成员 numEntries 3428 的值指定到数量“numStscEntries”。样本表结构 ChunkOffsetTable、SampleToChunkTable 等等是 trak 结构 3200trakAtom 4320 的成员“TrackMedia”3206 表示的 TrackMedia 数据结构 3226 的成员“MediaInfo”3234 表示的 MediaInfo 数据结构 3333 的成员“Sample Tables”3340 表示的 SampleTables 结构 3400 的成员。

[0891] 4. 操作 4715, 将零值指定到数量“sttsIndex”。将样本表结构 3420TimeToSampleTable 3410 内包含的表数组 3430 中的第一条目的值指定到表示 TimeToSample 结构 3470 的地址的数量 sttsEntry。将 TimeToSample 结构 sttsEntry 的成员“count”的值指定到数量“count”。将 TimeToSample 结构 sttsEntry 的成员“duration”的值指定到数量“duration”。

[0892] 5. 操作 4720, 将零值指定到数量“stscIndex”。将样本表结构 3420SampleToChunkTable 3406 内包含的表数组 3430 中的第一条目的值指定到表示 SampleToChunk 结构 3460 的地址的数量 stscEntry。将 SampleToChunk 结构 stscEntry 的成员“firstChunk”的值(-1),再指定到整型量“firstChunk”。

[0893] 6. 操作 4725, 将零值指定到整型量“time”、“sample”, 以及“chunk”。

[0894] 7. 操作 4730, 将数量“chunk”的值与数量“numChunks”的值进行比较。

[0895] 8. 在操作 4735 中, 如果数量“chunk”的值等于数量“numChunks”的值, 则过程“OdsmdDataToXml”完成。

[0896] 9. 在操作 4740 中, 如果数量“chunk”的值不等于数量“numChunks”的值, 则将值“8”加到 mdat 结构 3160mdatAtom 4230 的成员“start”的值, 再将样本表结构 3420ChunkOffsetTable 3408 中包含的表数组 3430 中的条目“chunk”的值减去该结果, 然后, 再将所得的结果指定到整型量“offset”。

[0897] 10. 操作 4745, 将数量“chunk”的值与数量“firstChunk”的值进行比较。

[0898] 11. 在操作 4750 中, 如果数量“chunk”的值等于数量“firstChunk”的值, 则将 SampleToChunk 结构 3460stscEntry 中的成员“numSamples”3466 的值指定到数量“numSamples”。将数量“stscIndex”的值增大 1。如果数量“stscIndex”的值小于数量“numStscEntries”的值, 则将样本表结构 3420SampleToChunkTable3406 内包含的表数组 3430 中的条目“stscIndex”的值指定到 SampleToChunk 结构 3460stscEntry, 并将 SampleToChunk 结构 stscEntry 的成员“firstChunk”的值(-1)指定到整型量“firstChunk”。否则, 将数量“numChunks”的值指定到数量“firstChunk”。

[0899] 12. 操作 4755 通过对 SampleSizeTable 结构 3440SampleSizeTable 3413 中包含的整数的数组 3456 中的条目“sample”到“sample+numSamples-1”进行求和来确定数量“chunkSize”的值。

[0900] 13. 操作 4760, 创建新的“odsmChunk”元素, 并附加到在操作 4705 中创建的新“odsm”元素。将数量“offset”的值指定到新的“odsmChunk”元素的属性“offset”。将数量“chunkSize”的值指定到新的“odsmChunk”元素的属性“size”。

[0901] 14. 操作 4770, 执行过程“解码 odsm 块”, 如图 48 所示。

[0902] 15. 操作 4780, 将数量“chunk”的值增大 1, 并重复操作 4730。

[0903] 过程“解码 odsm 块”

- [0904] 如图 48 所示,过程“解码 odsm 块”包括下列步骤:
- [0905] 1. 操作 4800,将数量“chunkSize”的值与零进行比较。
- [0906] 2. 操作 4805,如果数量“chunkSize”的值不大于零,则完成过程“解码 odsm 块”,并在操作 4780 中恢复过程“OdsmDataToXml”。
- [0907] 3. 操作 4810,如果数量“chunkSize”的值大于零,则将 SampleSizeTable 结构 3440SampleSizeTable 3413 中的成员表 3456 中的条目“sample”的值指定到整型量“sampleSize”。
- [0908] 4. 操作 4820,创建新的 odsmSample 元素,并附加到在上面的 4760 创建的 odsmChunk 元素。将数量“offset”的值指定到新的 odsmSample 元素的属性“offset”。将数量“sampleSize”的值指定到新的 odsmSample 元素的属性“size”。将数量“time”的值指定到新的 odsmSample 元素的属性“time”。
- [0909] 5. 操作 4825,将数量“time”的值增大数量“duration”的值。将数量“count”的值减去值“1”。
- [0910] 6. 操作 4830,将数量“count”的值与零进行比较。
- [0911] 7. 操作 4835,如果数量“count”的值不大于零,则将数量“sttsIndex”的值增大值“1”。将样本表结构 3420TimeToSampleTable 3410 内包含的表数组 3430 中的条目“sttsIndex”的值指定到 TimeToSample 结构 3470sttsEntry。将 TimeToSample 结构 sttsEntry 的成员“count”的值指定到数量“count”。将 TimeToSample 结构 sttsEntry 的成员“duration”的值指定到数量“duration”。
- [0912] 8. 操作 4840,将数量“chunkSize”的值减少数量“sampleSize”的值。
- [0913] 9. 操作 4845,将数量“sampleSize”的值与零进行比较。
- [0914] 10. 操作 4850,如果数量“sampleSize”的值不大于零,则将数量“sample”的值增大值“1”,并继续执行操作 4800。
- [0915] 11. 操作 4860,如果数量“sampleSize”的值大于零,则将字节的数组“OdsmData”中的条目“offset”的值指定到数量“tag”。将数量“offset”的值增大值“1”,将数量“sampleSize”的值减少值“1”。
- [0916] 12. 操作 4865,执行过程“获取标记大小”。这相当于上文所描述的过程“获取标记大小”,只是
- [0917] (a) 使用字节的数组代替数组“esdData”,
- [0918] (b) 使用数量“offset”代替数量“esdPos”
- [0919] (c) 每次数量“offset”的值(esdPos)增大 1 时,数量“sampleSize”的值都减少值“1”(操作 4660,4690)。
- [0920] 13. 操作 4870,将数量“tag”的值与 MPEG-4 系统规范中定义的数量“ObjectDescrUpdateTag”的值进行比较。
- [0921] 14. 操作 4875,如果数量“tag”的值等于数量“ObjectDescrUpdateTag”的值,则执行过程“解码 OdUpdate”。
- [0922] 15. 否则,执行操作 4880,将数量“tag”的值与 MPEG-4 系统规范中定义的数量“ObjectDescrRemoveTag”的值进行比较。
- [0923] 16. 操作 4885,如果数量“tag”的值等于数量“ObjectDescrRemoveTag”的值,则

执行过程“解码 OdRemove”。

[0924] 17. 否则,执行操作 4890,将数量“offset”的值增大数量“tagSize”的值

[0925] 18. 操作 4895,将数量“sampleSize”的值减少数量“tagSize”的值(如在操作 4865 中确定的)。

[0926] 18. 继续执行操作 4845。

[0927] 过程“解码 OdUpdate”

[0928] 如果在操作 4860 中 odsm 命令的命令标记的值等于值“ObjectDescrUpdateTag”(操作 4870、2000),则执行下列步骤 4875,如图 49 所示:

[0929] 1. 操作 4900,创建新的 ObjectDescrUpdate 元素 2540,并将它附加到当前 odsmSample 元素 2520。

[0930] 2. 操作 4910,使用过程“获取标记大小”,以获得数量“tagSize”的值(numBytes 2020)。这相当于上文所描述的过程“获取标记大小”,只是(a)使用字节的数组代替数组“esdData”,以及(b)使用数量“offset”代替数量“esdPos”。将数量“tagSize”的值指定到数量“residualSize”。

[0931] 3. 操作 4920,将数量“residualSize”的值与值“1”进行比较。

[0932] 4. 在操作 4925 中,如果数量“ResidualSize”的值小于 1,则此过程完成。

[0933] 5. 在操作 4930 中,如果“residualSize”的值不小于 1,则字节的数组“OdsmData”中的条目“offset”的值指定到数量“odTag”。将数量“offset”的值增大“1”,并将数量“residualSize”的值减少“1”。

[0934] 6. 操作 4940,使用过程“获取标记大小”,以获得数量“tagSize”的值(numBytes 2116)。这相当于上文所描述的过程“获取标记大小”,只是(a)使用字节的数组代替数组“esdData”,以及(b)使用数量“offset”代替数量“esdPos”。将数量“tagSize”的值指定到数量“odSize”。

[0935] 7. 操作 4745,将数量“residualSize”的值减少数量“odSize”的值。

[0936] 8. 操作 4960,将数量“odTag”的值与 MPEG-4 系统规范中定义的值“MP4_OD_Tag”进行比较。

[0937] 9. 在操作 4965 中,如果数量“odTag”的值不等于“MP4_OD_Tag”,则将数量“offset”的值增大数量“tagSize”的值,然后重复操作 4920。

[0938] 10. 在操作 4970 中,如果数量“odTag”的值等于值“MP4_OD_Tag”2108,则创建新的“ObjectDescriptor”元素 2550,并将它附加到当前 ObjectDescriptorUpdate 元素 2540。

[0939] 11. 操作 4980,确定数量“ObjectDescriptorID”2124 的值,并将结果指定到新的 ObjectDescriptor 元素 2550 的属性“ODID”。

[0940] 数量“ObjectDescriptorID”的值可以通过下列操作来确定:

[0941] (a) 将字节的数组“OdsmData”中的 entry offset 的值指定到整型量“temp1”

[0942] (b) 将数量“offset”的值增大 1,并将数量“odSize”的值减少 1,

[0943] (c) 将字节的数组“OdsmData”中的 entry offset 的值指定到整型量“temp2”

[0944] (d) 将数量“offset”的值增大 1,并将数量“odSize”的值减少 1,

[0945] (e) 将数量“temp1”的值乘以 4(向左移 2 位),

[0946] (f) 将数量“temp2”的值除以 64(向右移 6 位),

- [0947] (g) 将值“temp1”和“temp2”的总和指定到整型量“ObjectDescriptorID”，
- [0948] 12. 操作 4990, 执行如下所述的“decodeEsIdRef”。
- [0949] 过程“decodeEsIdRef”
- [0950] 如图 50 所示, 过程“decodeEsIdRef”4990 包括下列步骤：
- [0951] 1. 操作 5000, 将数量“odSize”的值与值“1”进行比较。
- [0952] 2. 在操作 5010 中, 如果数量“odSize”的值小于 1, 则此过程完成。
- [0953] 3. 操作 5020, 将字节的数组“OdsMData”的条目“offset”的值指定到数量“esdTag”, 将数量“offset”的值增大“1”并将数量“odSize”的值减少“1”。
- [0954] 4. 操作 5030, 使用过程“获取标记大小”, 以获得数量“tagSize”的值 (numBytes 2180)。这相当于上文所描述的过程“获取标记大小”, 只是 (a) 使用字节的数组代替数组“esdData”, 以及 (b) 使用数量“offset”代替数量“esdPos”。将数量“tagSize”的值指定到数量“esdSize”。
- [0955] 5. 操作 5040, 将数量“odSize”的值减少数量“esdSize”的值。
- [0956] 6. 操作 5050, 将数量“esdTag”的值与 MPEG-4 系统规范中定义的值“ES_ID_RefTag”进行比较。
- [0957] 7. 在操作 5060 中, 如果数量“esdTag”的值不等于值“ES_ID_RefTag”, 则将数量“offset”的值增大数量“esdSize”的值, 然后重复操作 5000。
- [0958] 8. 在操作 5070 中, 如果数量“esdTag”的值等于值“ES_ID_RefTag”2170, 则创建新的“EsIdRef”元素 2560, 并将它附加到当前 ObjectDescriptor 元素 2550。
- [0959] 9. 操作 5080, 确定数量“ES_ID”2190 的值, 并将结果指定到新的 EsIdRef 元素 2560 的属性“EsId”。
- [0960] 数量“ES_ID”的值可以通过下列操作来确定：
- [0961] (a) 将字节的数组“OdsMData”中的 entry offset 的值指定到整型量“temp1”
- [0962] (b) 将数量“offset”的值增大 1,
- [0963] (c) 将字节的数组“OdsMData”中的 entry offset 的值指定到整型量“temp2”
- [0964] (d) 将数量“offset”的值增大 1,
- [0965] (e) 将数量“temp1”的值乘以 256 (向左移 8 位),
- [0966] (f) 将值“temp1”和“temp2”的总和指定到整型量“ES_ID”。
- [0967] 过程“解码 OdRemove”
- [0968] 如果 odsm 命令 1960 的命令标记 4860 的值等于值“ObjectDescrRemoveTag”4880、2050, 则执行 4885 下列步骤：
- [0969] 1. 创建新的 ObjectDescrRemove 元素 2570, 并将它附加到当前 odsmSample 元素 2520。
- [0970] 2. 使用过程“获取标记大小”, 以获得数量“tagSize”的值 (numBytes 2060)。这相当于上文所描述的过程“获取标记大小”, 只是 (a) 使用字节的数组代替数组“esdData”, 以及 (b) 使用数量“offset”代替数量“esdPos”。
- [0971] 3. 将数量“tagSize”的值乘以 8, 并将结果除以 10, 然后, 将结果指定到数量“numOdIds”。
- [0972] 4. 创建新的空字符串数量“OdIdList”。

[0973] 5. 将字节的数组“OdsData”当作以条目“offset”开始的位的序列。从此位的序列中提取 numOdId 连续的 10 位整数值 (ObjectDescriptorId 2070)。

[0974] 6. 创建这些 objectDescriptorId 值 2070 中的每一个值的字符串表示。将这些字符串值中的每一个字符串值连接到字符串量“OdIdList”，将连续值用单个空格字符分隔。

[0975] 7. 将字符串量“OdIdList”的所产生的值指定到新的 ObjectDescrRemove 元素 2570 的属性“ODID”。

[0976] 过程“VisualDataToXml”

[0977] 过程“VisualDataToXml”4375 创建块的 xml 表示，包括 mdat 结构“mdatAtom”4230 内包含的视频媒体数据流。这些块内包含的媒体数据也被复制到外部媒体数据文件。如图 51 所示，此过程包括下列操作：

[0978] 1. 在操作 5100 中，将 mdat 结构“mdatAtom”4230 的成员“MediaData”3168 的值指定到表示字节的数组的地址的数量“MediaData”。将 mdat 结构“mdatAtom”4230 的成员“size”3163 的值指定到整型量“MediaSize”。

[0979] 2. 在操作 5105 中，创建新的“mediaFile”元素 2480，并附加到当前 mdat 元素 2400、4260。每一个这样的“mediaFile”元素 2480 都具有两个属性，“trackID”和“name”。属性“name”的值指定将包含媒体数据块（包括此流）的媒体数据文件 2230 的名称。

[0980] 将 trak 结构 3200 “trakAtom”4320 内包含的 TrackHeader 结构 3212、3204 的成员 trackID 3224 的值指定到新的“mediaFile”元素 2480 的属性“trackID”的值。

[0981] “name”属性的值通过连接三个字符串（文件名存根、序列号和文件类型下标）来确定。文件名存根可以通过输入 mp4 文件的名称的副本来确定，排除“.”（点或句号）字符和所有后续字符的最后一次出现。序列号可以由视频对象序列号的值的字符串表示“numVisual”构成。

[0982] 文件类型下标可以取决于数量 objectType 的值和 / 或媒体数据流内包含的文件格式指示。例如，如果数量 objectType 的值为“32”，则文件类型下标可以被指定为“.m4v”（MPEG-4video）。

[0983] 3. 在操作 5110 中，则打开新媒体数据文件进行写入操作。此新媒体数据文件的名称通过在操作 5105 中建立的 mediaFile 元素 2480 的“name”属性的值来确定。

[0984] 在“MPEG-4video”（objectType 1044 为“32”）的情况下，数据的初始块必须放在新媒体数据文件的开始。此数据块包含在 trak 结构 3200（具有 trackID 值 3224，该值具有数量“iTrack”指定的值）SampleDescription 数据结构 3480 中的成员“esdData”3496 指定的字节的数组内包含的“解码器特定信息”块 1072、1076 中。SampleDescription 数据结构 3480 通过此 trak 结构的 SampleDescriptions 数据结构 3400 中的 SampleDescriptionTable 数据结构 3418 的成员表 3430 中的第一条目（条目 0）。“解码器特定信息”块 1072、1076 包含在此 esdData 数据块 3496 内包含的“解码器配置描述符”数据块 1024、1032 内。

[0985] 解码器配置描述符数据 1032 可以使用上文所描述的过程“发现解码器配置描述符”来进行定位。解码器特定信息数据块 1072、1076 位于在解码器配置描述符 1032 内，跟在 decoderConfigDescriptorTag 1036、解码器配置描述符大小值 1040，以及 14 个字节的解码器配置描述符数据 1044 到 1068。解码器特定信息数据块 1076 包括一字

节 decoderSpecificInfoTag 1080(值“5”)、解码器特定信息数据块大小值(numBytes 1084),以及 numBytes 字节的解码器特定信息数据。此 numBytes 1084 字节的序列被复制到输出 MPEG-4 视频文件 2230。

[0986] 4. 在操作 5115 中,将样本表结构 3420ChunkOffsetTable3408 的成员 numEntries 3428 的值指定到数量“numChunks”。将样本表结构 3420SampleToChunkTable 3406 的成员 numEntries3428 的值指定到数量“numStscEntries”。样本表结构 ChunkOffsetTable、SampleToChunkTable 等等是 trak 结构 3200trakAtom 4230 的成员“TrackMedia”3206 表示的 TrackMedia 数据结构 3226 的成员“MediaInfo”3234 表示的 MediaInfo 数据结构 3333 的成员“Sample Tables”3340 表示的 SampleTables 结构 3400 的成员。

[0987] 5. 在操作 5120 中,将 TimeToSample 结构 sttsEntry 的成员“count”的值指定到数量“count”。

[0988] 6. 在操作 5125 中,将零值指定到数量“stscIndex”。将样本表结构 3420SampleToChunkTable 3406 内包含的表数组 3430 中的第一条目的值指定到表示 SampleToChunk 结构 3460 的地址的数量 stscEntry。将 SampleToChunk 结构 stscEntry 的成员“firstChunk”的值(-1),再指定到整型量“firstChunk”。

[0989] 7. 在操作 5130 中,将零值指定到整型量“chunk”。

[0990] 8. 在操作 5140 中,将数量“chunk”的值与数量“numChunks”的值进行比较。

[0991] 9. 在操作 5145 中,如果数量“chunk”的值等于数量“numChunks”的值,则过程“VisualDataToXml”完成。

[0992] 10. 在操作 5150 中,如果数量“chunk”的值不等于数量“numChunks”的值,则将值“8”加到 mdat 结构 3160mdatAtom 4230 的成员“start”的值,再将样本表结构 3420ChunkOffsetTable 3408 中包含的表数组 3430 中的条目“chunk”的值减去该结果,然后,再将所得的结果指定到整型量“offset”。

[0993] 11. 在操作 5160 中,将数量“chunk”的值与数量“firstChunk”的值进行比较。

[0994] 12. 在操作 5165 中,如果数量“chunk”的值等于数量“firstChunk”的值,则将 SampleToChunk 结构 3460stscEntry 中的成员“numSamples”3466 的值指定到数量“numSamples”。将数量“stscIndex”的值增大 1。如果数量“stscIndex”的值小于数量“numStscEntries”的值,则将样本表结构 3420SampleToChunkTable3406 内包含的表数组 3430 中的条目“stscIndex”的值指定到 SampleToChunk 结构 3460stscEntry,并将 SampleToChunk 结构 stscEntry 的成员“firstChunk”的值(-1)指定到整型量“firstChunk”。否则,将数量“numChunks”的值指定到数量“firstChunk”。

[0995] 13. 在操作 5170 中,通过对 SampleSizeTable 结构 3440SampleSizeTable 3413 中包含的整数的数组 3456 中的条目“sample”到“sample+numSamples-1”进行求和来确定数量“chunkSize”的值。

[0996] 14. 在操作 5180 中,创建新的“chunk”元素,并将其附加到在操作 5105 中创建的新的“mediaFile”元素。将数量“offset”的值指定到新的“chunk”元素的属性“offset”。将数量“chunkSize”的值指定到新的“chunk”元素的属性“size”。

[0997] 15. 在操作 5185 中,从字节数组 MediaData 的开始在偏移字节的开始的媒体数据的 chunkSize 字节复制到在操作 5110 中打开的输出媒体数据文件。

[0998] 16. 在操作 5190 中,将数量“chunk”的值增大 1,并重复操作 5140。

[0999] 过程“AudioDataToXml”

[1000] 过程“AudioDataToXml”4385 创建块的 xml 表示,包括 mdat 结构“mdatAtom”4230 内包含的音频媒体数据流。这些块内包含的媒体数据也被复制到外部媒体数据文件。如图 51 所示,此过程相当于上文所描述的过程“VisualDataToXml”,下列差异除外:

[1001] 用于构成“name”属性的值的序列号可以由音频对象序列号的字符串表示“numAudio”构成。

[1002] 文件类型下标可以取决于数量 ObjectType 的值和 / 或媒体数据流内包含的文件格式指示。例如,如果数量 objectType 的值为“107”,则文件类型下标可以被指定为“.mp3”(MPEG-1 层 3 音频)。

[1003] 在操作 5145 中,如果数量“chunk”的值等于数量“numChunks”的值,则过程“AudioDataToXml”完成。

[1004] 某些类型的音频数据,如 MPEG-2AAC(高级音频编码)可能要求这里所描述的内容以外的其他处理。执行这样的处理的装置可以由精通本技术的人基于对应的媒体特定的文件结构规范来进行设计。这里没有介绍用于处理每一种类型的音频数据的特定装置。

[1005] 过程“SdsmDataToXml”

[1006] 过程“SdsmDataToXml”4365 创建当前 mdat 结构“mdatAtom”4230 内包含的 sdsm(BIFS) 数据流(场景描述流)的 xml 表示。此过程创建的 sdsm 的 xml 表示包括两个部分:

[1007] (a) 包含媒体数据块 2450(包括 sdsm)的摘要的 sdsm 元素 2440。此 sdsm 元素 2440 包含在当前 mdat 元素 4260(构成了 mp4file 文档 2300 的一部分)内。

[1008] (b) 在 mp4bifs 文档 2260、2800 中创建 sdsm 媒体数据样本 1100、1110 的 xml 表示。

[1009] 如图 52 所示,此过程包括下列操作:

[1010] 1. 在操作 5200 中,将 mdat 结构“mdatAtom”4230 的成员“MediaData”3168 的值指定到表示字节的数组的地址的数量“SdsmData”。将 mdat 结构“mdatAtom”4230 的成员“size”3163 的值指定到整型量“SdsmSize”。

[1011] 2. 在操作 5205 中,创建新的“sdsm”元素 2410、2440 并附加到当前 mdat 元素 2400、4260。每一个这样的“sdsm”元素 2440 都具有两个属性,“trackID”和“xmlFile”。属性“XmlFile”指定可以用来表示所产生的 mp4bifs 文档 2260、2800 的可选 xml 输出文件的名称。此可选的 xml 输出文件对诊断是有用的,但是这样的文件不是本发明的操作所必需的(即对于 mp4 二进制文件创建 XMT-A 文件)。

[1012] 将 trak 结构 3200 “trakAtom”4320 内包含的 TrackHeader 结构 3212、3204 的成员 trackID 3224 的值指定到新的“sdsm”元素 2440 的属性“trackID”的值。

[1013] 属性“xmlFile”的值通过连接两个字符串(文件名存根和文件类型下标)来确定。文件名存根可以通过输入 mp4 文件的名称的副本来确定,排除“.”(点或句号)字符和所有后续字符的最后一次出现。文件类型下标是“Bifs.xml”。

[1014] 3. 在操作 5210 中,创建新的“bifsConfig”元素 2810,并附加到 mp4bifs 文档 2260、2800 中的最顶端的元素。然后使用如下所述的过程“设置 bifsConfig 属性”来设置

“bifsConfig”元素 2810 的属性。

[1015] 4. 在操作 5215 中,将样本表结构 3420ChunkOffsetTable3408 的成员 numEntries 3428 的值指定到数量“numChunks”。将样本表结构 3420SampleToChunkTable 3406 的成员 numEntries3428 的值指定到数量“numStscEntries”。样本表结构 ChunkOffsetTable、SampleToChunkTable 等等是 trak 结构 3200trakAtom 4320 的成员“TrackMedia”3206 表示的 TrackMedia 数据结构 3226 的成员“MediaInfo”3234 表示的 MediaInfo 数据结构 3333 的成员“Sample Tables”3340 表示的 SampleTables 结构 3400 的成员。

[1016] 5. 在操作 5220 中,将零值指定到数量“sttsIndex”。将样本表结构 3420TimeToSampleTable 3410 内包含的表数组 3430 中的第一条目的值指定到表示 TimeToSample 结构 3470 的地址的数量 sttsEntry。将 TimeToSample 结构 sttsEntry 的成员“count”的值指定到数量“count”。将 TimeToSample 结构 sttsEntry 的成员“duration”的值指定到数量“duration”。

[1017] 6. 在操作 5225 中,将零值指定到数量“stscIndex”。将样本表结构 3420SampleToChunkTable 3406 内包含的表数组 3430 中的第一条目的值指定到表示 SampleToChunk 结构 3460 的地址的数量 stscEntry。将 SampleToChunk 结构 stscEntry 的成员“firstChunk”的值(-1),再指定到整型量“firstChunk”。

[1018] 7. 在操作 5230 中,将零值指定到整型量“chunk”。

[1019] 8. 在操作 5240 中,将数量“chunk”的值与数量“numChunks”的值进行比较。

[1020] 9. 在操作 5245 中,如果数量“chunk”的值等于数量“numChunks”的值,则过程“SdsmDataToXml”完成。

[1021] 10. 在操作 5250 中,如果数量“chunk”的值不等于数量“numChunks”的值,则将值“8”加到 mdat 结构 3160mdatAtom 4230 的成员“start”的值,再将样本表结构 3420ChunkOffsetTable 3408 中包含的表数组 3430 中的条目“chunk”的值减去该结果,然后将,再将所得的结果指定到整型量“offset”。

[1022] 11. 在操作 5260 中,将数量“chunk”的值与数量“firstChunk”的值进行比较。

[1023] 12. 在操作 5265 中,如果数量“chunk”的值等于数量“firstChunk”的值,则将 SampleToChunk 结构 3460stscEntry 中的成员“numSamples”3466 的值指定到数量“numSamples”。将数量“stscIndex”的值增大 1。如果数量“stscIndex”的值小于数量“numStscEntries”的值,则将样本表结构 3420SampleToChunkTable3406 内包含的表数组 3430 中的条目“stscIndex”的值指定到 SampleToChunk 结构 3460stscEntry,并将 SampleToChunk 结构 stscEntry 的成员“firstChunk”的值(-1)指定到整型量“firstChunk”。否则,将数量“numChunks”的值指定到数量“firstChunk”。

[1024] 13. 在操作 5270 中,通过对 SampleSizeTable 结构 3440SampleSizeTable 3413 中包含的整数的数组 3456 中的条目“sample”到“sample+numSamples-1”进行求和来确定数量“chunkSize”的值。

[1025] 14. 在操作 5275 中,基于关联的 trak 结构 3200 中包含的 TimeToSampleTable 数据结构 3410、TrackEdit 数据结构 3208、3236,以及 MediaHeader 数据结构 3230 中的表数组成员 3430 中的条目的内容,创建具有 numSamples 条目的时间戳的表。此表包含当前块中的每一个样本的一个条目,此表中的每一个条目都表示此流中的所有前面的样本的持续时

间值 3476,加在可选的 TrackEdit 数据结构 3208、3236 中指定的任何起始延迟（如果有的话）的总和。任何精通本技术的人员都可以创建用于从可用的信息生成这样的表的装置。

[1026] 15. 在操作 5280 中,创建新的“chunk”元素,并将其附加到在步骤 2 中创建的新的“sdsM”元素。将数量“offset”的值指定到新的“chunk”元素的属性“offset”。将数量“chunkSize”的值指定到新的“chunk”元素的属性“size”。

[1027] 16. 在操作 5285 中,如图 53 所示,执行过程“解码 sdsM 块”。

[1028] 17. 在操作 5290 中,将数量“chunk”的值增大 1,并重复操作 5240。

[1029] 过程“设置 bifsConfig 属性”

[1030] mp4bifs bifsConfig 元素 2810 具有七个属性:“nodeIdBits”、“routeIdBits”、“commandFrame”、“useBIFSv2Config”、“protoIdbits”、“use3DMeshCoding”,以及“usePredictiveMFfield”。这些属性的值通过当前 trak 数据结构“trakAtom”4320 的解码器特定信息数据 1072 的内容来确定。此数据块包含在当前 trackAtom 数据结构 3400、4320 内包含的 SampleTables 数据结构 3400 的成员 SampleDescriptionTable 3418 的成员“tablearray”3430 中的条目 0 指定的 SampleDescription 数据结构 3480 的 esdData 成员 1000、3496 内的解码器配置描述符 1024、1032 内。

[1031] 如果数量“objectType”4460 的值为“2”,那么,解码器特定信息数据表示“版本 2”BIFS 配置。在此情况下,解码器特定信息块的开头三个字节具有下列内容:

[1032] [uraaaaab bbbpppp pcxxxxxx]

[1033] 其中,

[1034] u = “use3DMeshCoding”的 1 位值,

[1035] r = “usePredictiveMFfield”的 1 位值,

[1036] aaaaa = “nodeIdBits”的 5 位无符号整数

[1037] bbbbb = “routeIdBits”的 5 位无符号整数

[1038] ppppp = “protoIdbits”的 5 位无符号整数

[1039] c = “commandFrame”的 1 位值。

[1040] 由 aaaaa、bbbbb,以及 ppppp 指定的值被分配到指定的属性。

[1041] 由“aaaaa”指定的无符号整数值被分配给 mp4bifs bifsConfig 元素 2810 的“nodeIdBits”。由“bbbbb”指定的无符号整数值被分配给 mp4bifs bifsConfig 元素 2810 的“routeIdBits”。由“ppppp”指定的无符号整数值被分配给 mp4bifs bifsConfig 元素 2810 的“protoIdbits”。

[1042] 如果“u”、“r”或“c”的值为“0”,那么,将值“假”分配给 mp4bifsbifsConfig 元素 2810 的对应的属性。如果“u”、“r”或“c”的值为“1”,那么,将值“真”分配给 mp4bifs bifsConfig 元素 2810 的对应的属性。

[1043] 如果数量“objectType”4460 的值不是“2”,那么,解码器特定信息数据表示“版本 1”BIFS 配置。在此情况下,解码器特定信息块的开头两个字节具有下列内容:

[1044] [aaaaabbb bbcxxxxx]

[1045] 其中,

[1046] aaaaa = “nodeIdBits”的 5 位无符号整数

[1047] bbbbb = “routeIdBits”的 5 位无符号整数

[1048] c = “commandFrame” 的 1 位值。

[1049] 在此情况下,不需要“protoIdbits”、“use3DMeshCoding”,以及“usePredictiveMFField”的值。

[1050] 由“aaaaa”指定的无符号整数值被分配给 mp4bifs bifsConfig 元素 2810 的“nodeIdBits”。由“bbbbbb”指定的无符号整数值被分配给 mp4bifs bifsConfig 元素 2810 的“routeIdBits”。

[1051] 如果“c”的值为“0”,那么,将值“假”分配给 mp4bifs bifsConfig 元素 2810 的“commandFrame”属性。否则,(如果“c”的值为“1”),则将值“真”分配给 mp4bifs bifsConfig 元素 2810 的“commandFrame”属性。

[1052] 过程“解码 sdsm 块”

[1053] 如图 53 所示,过程“解码 sdsm 块”包括下列步骤:

[1054] 1. 在操作 5300 中,将数量“offset”5250 的值指定到整型量“sdsmPos”。将数量“chunkSize”5270 的值加到数量“offset”的值,并将结果指定到整型量“sdsmEnd”。数量“sdsmPos”的值指定字节在字节的数组“SdsmData”5200 内的位置。

[1055] 关联的整型量“sdsmBit”指定位在由数量 sdsmPos 的值指定的字节内的位置。最初,将零值指定到关联的数量“sdsmBit”。对于从 sdsm 位流提取的每一个位,EdsmBit 的值都增大 1。每当 sdsmBit 的值超过 7,“sdsmBit”的值都减少 8,“sdsmPos”的值都增大 1。

[1056] 2. 在操作 5305 中,将零值指定到整型量“sample”,并将值“假”指定到布尔量“FrameOpen”。

[1057] 3. 在操作 5310 中,将数量“sdsmPos”的值与数量“sdsmEnd”的值进行比较。

[1058] 4 在操作 5315 中,如果数量“sdsmPos”的值不小于数量“sdsmEnd”的值,则此过程完成。

[1059] 5. 在操作 5320 中,如果数量“sdsmPos”的值小于数量“sdsmEnd”的值,则将数量“frameOpen”的值与值“假”进行比较。

[1060] 6. 在操作 5325 中,如果数量“frameOpen”的值等于值“假”,则创建新的“commandFrame”元素 2820,并附加到 mp4bifs 文档 2800。将时间戳的表 5275 中的条目“sample”的值指定到新的 commandFrame 元素的“time”属性。将值“真”指定到布尔量“frameOpen”。

[1061] 7. 在操作 5330 中,将“sdsmPos”和“sdsmBit”的值指定的点开始的 sdsm 位流中的下两个位指定到数量“cmdCode”。将数量“sdsmBit”的值增大 2,并将数量“cmdCode”的值与 0 进行比较。

[1062] 8. 在操作 5335 中,如果数量“cmdCode”的值等于“0”1206,则执行过程“解码插入命令”。

[1063] 9. 在操作 5340 中,将数量“cmdCode”的值与“1”进行比较。

[1064] 10. 在操作 5345 中,如果数量“cmdCode”的值等于“1”1226,则执行过程“解码删除命令”。

[1065] 11. 在操作 5350 中,将数量“cmdCode”的值与“2”进行比较。

[1066] 1. 2. 在操作 5355 中,如果数量“cmdCode”的值等于“2”1244,则执行过程“解码替换命令”。

[1067] 13. 在操作 5360 中,如果“cmdCode”的值不是“0”、“1”或“2”,则它必须是“3”,因为此值由两个位的数据确定。因此,此步骤的结果始终是“是”。

[1068] 14. 在操作 5365 中,如果数量“cmdCode”的值等于“3”1280,则执行过程“解码 ReplaceScene 命令”。

[1069] 过程“解码 ReplaceScene cmd”包括创建新的“ReplaceScene”bifsCommand 元素 2930,并将它附加到当前“commandFrame”元素 2830(操作 5310)。然后,解释跟在场景替换 cmdCode 1280 后面的 BIFS 场景数据结构 1290、1600,并根据需要提高 sdsmBit 和 sdsmByte 的值。

[1070] 通过增大 sdsmBit 的值,跳过“预留”组件 1610 指定的值。确定 USENAMES 1620 指定的值,以及 BIFS 场景结构 1600 的 protoList 1630 组件,并指定到新的“ReplaceScene”元素的对应的属性。

[1071] 然后使用过程“解码 SFNode”来解释 BIFS 场景结构 1600 的 SFTopNode 组件 1640。新的“ReplaceScene”bifsCommand 元素 2930 充当所产生的 TopNode BIFS 节点元素 2940 的父元素。

[1072] 将一位值“hasRoutes”1650 指定到布尔值“hasRoutes”。如果“hasRoutes”的值为“1”(“真”),那么,解释下列 Routes 组组件 1660、1800、1830。创建新的“Routes”元素 2950,并附加到新的“ReplaceScene”元素 2930。创建新的“Route”元素 2950,并附加到 Routes 组件 1660 内包含的每一个 Route 结构 1810 或 1850 的新的“Routes”元素 2950。确定每一个 Route 结构 1860 的 routeID1870、routeName 1875、outNodeID 1880、outFieldRef 1885、inNodeID1890,以及 inFieldRef 1895 组件指定的值,并指定到每一个新的“Route”元素 2960 的对应的属性。

[1073] 15. 在操作 5370 中,在完成过程“解码插入命令”、“解码删除命令”、“解码替换命令”或“解码 ReplaceScene 命令”之后,将 sdsm 位流中的下一位指定到布尔量“continue”1130 或 1140。如果数量“continue”的所产生的值为“1”(“真”)1130,则此过程继续执行操作 5310。

[1074] 16. 操作 5375,如果数量“continue”的值为“0”(“假”)1140,则 sdsm 位流转到下一字节的开始。即,如果数量 sdsmBit 的值为零,则不执行任何操作。否则,将“sdsmBit”的值设置为零,并将 sdsmByte 的值增大 1。此步骤跳过 sdsm 命令帧 1110 结尾处的“填充位”1150。

[1075] 17. 在操作 5380 中,将布尔量“frameOpen”的值设置为“假”,将数量“sample”的值增大 1,此过程继续执行操作 5310。

[1076] 过程“解码插入命令”

[1077] 如图 54 所示,过程“解码插入命令”包括下列步骤:

[1078] 1. 在操作 5400 中,将 sdsm 位流 1210 中的下两位指定到数量“paramType”。

[1079] 2. 在操作 5410 中,将数量“paramType”的值与“0”进行比较。

[1080] 3. 在操作 5420 中,如果数量“paramType”的值为“0”1300,则执行过程“解码 InsertNode cmd”。

[1081] 过程“解码 InsertNode cmd”包括创建新的“InsertNode”bifsCommand 元素 2840,并将它附加到当前父元素 2830(操作 5310)。确定节点插入命令 1300 的 nodeID

1312、insertionPosition1316, 以及 position 1320 组件指定的值, 并指定到新的“InsertNode”bifsCommand 元素的对应的属性。

[1082] 然后使用过程“解码 SFNode”来解释节点插入命令 1300 的 SFNode 组件 1324。

[1083] 4. 在操作 5430 中, 将数量“paramType”的值与“1”进行比较。

[1084] 5. 在操作 5440 中, 如果数量“paramType”的值为 1, 则遇到错误。在有效 sdsm 数据流内不允许此值。解码 sdsm 位流的过程必须终止。

[1085] 6. 在操作 5450 中, 将数量“paramType”的值与“2”进行比较。

[1086] 7. 在操作 5460 中, 如果数量“paramType”的值为“2”1336, 则执行过程“解码 InsertIdxValue cmd”。

[1087] 过程“解码 InsertIdxValue cmd”包括创建新的“InsertIndexedValue”bifsCommand 元素 2840, 并将它附加到当前父元素 2830(操作 5310)。确定 Indexed Value Insertion 命令 1328 的 nodeID 1340、inFieldID 1344、insertionPosition 1348, 以及 position 1320 组件指定的值, 并指定到新的“InsertIndexedValue”bifsCommand 元素的对应的属性。

[1088] 由组件“field value”1356 表示的数据的类型(布尔、整数、字符串等等)由数量“nodeID”、“inFieldID”的值, 以及 MPEG-4 系统规范中定义的表来确定。如果字段值的数据类型为“SFNode”, 那么, 使用过程“解码 SFNode”来解释 Indexed Value Insertion 命令 1328 的“field value”组件 1356, 并将“field value”组件中包含的 BIFS 节点的名称指定到新的“InsertIndexedValue”元素的“value”属性。否则, 将由“field value”组件 1356 表示的数值或字符串值指定到新的“InsertIndexedValue”bifsCommand 元素的“value”属性。

[1089] 8. 在操作 5470 中, 如果“paramType”的值不是“0”、“1”或“2”, 则它必须是“3”1368, 因为此值由两个位的数据确定。因此, 此步骤的结果始终是“是”。

[1090] 9. 在操作 5480 中, 执行过程“解码 InsertRoute cmd”。

[1091] 过程“解码 InsertRoute cmd”包括创建新的“InsertRoute”bifsCommand 元素 2840, 并将它附加到当前父元素 2830(操作 5310)。确定 Route Insertion 命令 1360 的 isUpdateable 1372、routeID 1376、departureNodeID 1380、departureFieldID 1384、arrivalNodeID 1388, 以及 arrivalFieldID 1392 组件指定的值, 并指定到新的“InsertRoute”bifsCommand 元素的对应的属性。

[1092] 10. 在操作 5490 中, 在完成过程“解码 InsertNode cmd”、“解码 InsertIdxValue cmd”或“解码 InsertRoute cmd”之后, 对 sdsm 位流的处理在操作 5370 中用数量“moreCommands”的值继续。

[1093] 过程“解码删除命令”

[1094] 如图 55 所示, 过程“解码删除命令”包括下列步骤:

[1095] 1. 在操作 5500 中, 将 sdsm 位流中的下两位指定到数量“paramType”。

[1096] 2. 在操作 5510 中, 将数量“paramType”的值与“0”进行比较。

[1097] 3. 在操作 5520 中, 如果数量“paramType”的值为“0”1412, 则执行过程“解码 DeleteNode cmd”。

[1098] 过程“解码 DeleteNode cmd”包括创建新的“DeleteNode”bifsCommand 元素 2840,

并将它附加到当前父元素 2830(操作 5310)。确定 Node Deletion 命令 1400 的 nodeID 1418 组件指定的值,并指定到新的“DeleteNode”bifsCommand 元素的“nodeID”属性。

[1099] 4. 在操作 5530 中,将数量“paramType”的值与“1”进行比较。

[1100] 5. 在操作 5540 中,如果数量“paramType”的值为 1,则遇到错误。在有效 sdsm 数据流内不允许此值。解码 sdsm 位流的过程必须终止。

[1101] 6. 在操作 5550 中,将数量“paramType”的值与“2”进行比较。

[1102] 7. 在操作 5560 中,如果数量“paramType”的值为“2”1436,则执行过程“解码 DeleteIdxValue cmd”。

[1103] 过程“解码 DeleteIdxValue cmd”包括创建新的“DeleteIndexedValue”bifsCommand 元素 2840,并将它附加到当前父元素 2830(操作 5310)。确定 Node Deletion 命令 1424 的 nodeID 1442、inFieldID 1448、deletionPosition 1454,以及 position1460 组件指定的值,并指定到新的“DeleteIndexedValue”bifsCommand 元素的对应的属性。

[1104] 8. 在操作 5570 中,如果“paramType”的值不是“0”、“1”或“2”,则它必须是“3”1478,因为此值由两个位的数据确定。因此,此步骤的结果始终是“是”。

[1105] 9. 在操作 5580 中,执行过程“解码 DeleteRoute cmd”。

[1106] 过程“解码 DeleteRoute cmd”包括创建新的“DeleteRoute”bifsCommand 元素 2840,并将它附加到当前父元素 2830(操作 5310)。确定 Route Deletion 命令 1466 的 routeID 1484 组件指定的值,并指定到新的“DeleteRoute”bifsCommand 元素的“routeID”属性。

[1107] 10. 在操作 5590 中,在完成过程“解码 InsertNode cmd”、“解码 InsertIdxValue cmd”或“解码 InsertRoute cmd”之后,对 sdsm 位流的处理在操作 5370 中用数量“moreCommands”的值继续。

[1108] 过程“解码替换命令”

[1109] 如图 56 所示,过程“解码替换命令”包括下列步骤:

[1110] 1. 在操作 5600 中,将 sdsm 位流中的下两位指定到数量“paramType”。

[1111] 2. 在操作 5610 中,将数量“paramType”的值与“0”进行比较。

[1112] 3. 在操作 5620 中,如果数量“paramType”的值为“0”1508,则执行过程“解码 ReplaceNode cmd”。

[1113] 过程“解码 ReplaceNode cmd”包括创建新的“ReplaceNode”bifsCommand 元素 2840,并将它附加到当前父元素 2830(操作 5310)。确定 Node Replacement 命令 1500 的 nodeID 1510 组件指定的值,并指定到新的“ReplaceNode”bifsCommand 元素的“nodeID”属性。

[1114] 然后使用过程“解码 SFNode”来解释 Node Replacement 命令 1500 的 SFNode 组件 1514。

[1115] 4. 在操作 5630 中,将数量“paramType”的值与“1”进行比较。

[1116] 5. 在操作 5640 中,如果数量“paramType”的值为“1”1538,则执行过程“解码 ReplaceField cmd”。

[1117] 过程“解码 ReplaceField cmd”包括创建新的“ReplaceField”bifsCommand

元素 2840, 并将它附加到当前父元素 2830(操作 5310)。确定 Field Replacement 命令 1520 的 nodeID 1530, 以及 inFieldID 1534 组件指定的值, 并指定到新的“ReplaceField”bifsCommand 元素的对应的属性。

[1118] 由组件“field value”1538 表示的数据的类型(布尔、整数、字符串等等)由数量“nodeID”、“inFieldID”的值, 以及 MPEG-4 系统规范中定义的表来确定。如果字段值的数据类型为“SFNode”, 那么, 使用过程“解码 SFNode”来解释 Field Replacement 命令 1520 的“field value”组件 1538, 并将“field value”组件中包含的 BIFS 节点的名称指定到新的“ReplaceField”元素的“value”属性。否则, 将由“field value”组件 1538 表示的数值或字符串值指定到新的“ReplaceField”bifsCommand 元素的“value”属性。

[1119] 6. 在操作 5650 中, 将数量“paramType”的值与“2”进行比较。

[1120] 7. 在操作 5660 中, 如果数量“paramType”的值为“2”1548, 则执行过程“解码 ReplaceIdxValue cmd”。

[1121] 过程“解码 ReplaceIdxValue cmd”包括创建新的“ReplaceIndexedValue”bifsCommand 元素 2840, 并将它附加到当前父元素 2830(操作 5310)。确定 Indexed Value Replacement 命令 1540 的 nodeID 1550、inFieldID 1554、replacementPosition 1558, 以及 position 1560 组件指定的值, 并指定到新的“ReplaceIndexedValue”bifsCommand 元素的对应的属性。

[1122] 由组件“field value”1564 表示的数据的类型(布尔、整数、字符串等等)由数量“nodeID”、“inFieldID”的值, 以及 MPEG-4 系统规范中定义的表来确定。如果字段值的数据类型为“SFNode”, 那么, 使用过程“解码 SFNode”来解释 Indexed Value Replacement 命令 1540 的“field value”组件 1564, 并将“field value”组件中包含的 BIFS 节点的名称指定到新的“ReplaceIndexedValue”元素的“value”属性。否则, 将由“field value”组件 1564 表示的数值或字符串值指定到新的“replaceIndexedValue”bifsCommand 元素的“value”属性。

[1123] 8. 在操作 5670 中, 如果“paramType”的值不是“0”、“1”或“2”, 则它必须是“3”1578, 因为此值由两个位的数据确定。因此, 此步骤的结果始终是“是”。

[1124] 9. 在操作 5680 中, 执行过程“解码 ReplaceRoute cmd”。

[1125] 过程“解码 ReplaceRoute cmd”包括创建新的“ReplaceRoute”bifsCommand 元素 2840, 并将它附加到当前“commandFrame”元素 2830(操作 5310)。确定 Route Replacement 命令 1570 的 routeID1580、departureNodeID 1584、departureFieldID 1588、arrivalNodeID1590, 以及 arrivalFieldID 1594 组件指定的值, 并指定到新的“ReplaceRoute”bifsCommand 元素的对应的属性。

[1126] 10. 在操作 5490 中, 在完成过程“解码 InsertNode cmd”、“解码 InsertIdxValue cmd”或“解码 InsertRoute cmd”之后, 对 sdsm 位流的处理在操作 5370 中用数量“moreCommands”的值继续。

[1127] 过程“解码 SFNode”

[1128] 图 17 概述了 SFNode 数据结构 1700、1710、1730 的结构。每一个 SFNode 数据结构都包含在父数据结构内。父数据结构可以是 BIFS 命令 1120 或另一个 SFNode 结构。不论是哪一种情况, 在 mp4bifs 文档中将会有对应的父元素。在 BIFS 命令父结构的情况下,

mp4bifs 父元素将是 bifsCommand 元素 2910。在 SFNode 父结构的情况下,mp4bifs 父元素将是 BIFS Node 元素。

[1129] 如图 57 所示,过程“解码 SFNode”包括下列步骤:

[1130] 1. 在操作 5700 中,将 sdsm 位流中的下一位指定到布尔量“isReused”1704、1712、1732,并将所产生的值与值“真”(1) 进行比较。

[1131] 2. 在操作 5710 中,如果“isReused”的值为“真”(1)1704,则创建新的“ReusedNode”元素,并附加到当前 mp4bifs 父元素。然后,解释“nodeIDRef”组件 1708 的值,并将所产生的整数值指定到新的“ReusedNode”元素的“nodeRef”属性。用于指定值“nodeIDRef”的位数由 mp4bifs 文档 2800 的“bifsconfig”元素 2810 的“nodeIdBits”属性来确定。这就完成了 SFNode 结构的处理。

[1132] 3. 在操作 5720,如果“isReused”的值为“0”(“假”)1712、1732,则从 sdsm 位流提取“localNodeType”1714、1734 的值。表示此值的位数由父结构的身份和 MPEG-4 系统规范中定义的表来确定。

[1133] 父结构的身份、“localNodeType”的值,以及 MPEG-4 系统规范中定义的表用于确定由此 SFNode 结构表示的 BIFS 节点的特定类型的“nodeNumber”属性的整数值。整型量“nodeNumber”的值和 MPEG-4 系统规范中定义的表用于确定字符串量“nodeName”的值,该值描述了由此 SFNode 结构表示的 BIFS 节点的类型。创建具有数量“nodeName”的值指定的元素名称的新 BIFS 节点元素,并附加到 mp4bifs 父元素。

[1134] 4. 在操作 5730 中,将 sdsm 位流中的下一位指定到布尔量“isupdateable”1716、1736,并将所产生的值与值“真”(1) 进行比较。

[1135] 5. 在操作 5740 中,如果“isUpdateable”的值为“1”,那么,从 sdsm 位流中提取数量“nodeID”的整数值。用于指定值“nodeID”的位数由 mp4bifs 文档 2800 的“bifsConfig”元素 2810 的“nodeIdBits”属性来确定。并将所产生的值指定到在操作 5720 中创建的新的 BIFS 节点元素的“nodeID”属性。

[1136] 6. 在操作 5750 中,将关联的“ReplaceScene”元素的“USENAMES”属性的布尔值与值“真”(1) 进行比较。

[1137] 7. 在操作 5760 中,如果“USENAMES”的值为“真”,那么,从 SFNode 结构中提取字母数字字符的空值结束的字符串 1720、1740,并将所产生的字符串值指定到在操作 5720 中创建的新的 BIFS 节点元素的“name”属性。

[1138] 8. 在操作 5770 中,将 sdsm 位流中的下一位指定到布尔量“maskAccess”1722、1742,并将所产生的值与值“真”(1) 进行比较。

[1139] 9. 在操作 5780 中,如果“maskAccess”的值为 1(“真”),则 SFNode 结构具有图 17B 所示的形式。在此情况下,maskAccess 位 1722 后面有屏蔽位 1726 的序列。屏蔽位的数量由数量“nFields”1728 的值表示。数量“nFields”的值由操作 5720 中的数量“nodeNumber”的值和 MPEG-4 系统规范中定义的表来确定。在解释“maskAccess”1722 的值之后,数量“fieldIndex”被设置为 0。在解释每一个屏蔽位 1726 之后,将数量“fieldIndex”的值增大 1。具有值“1”的每一个屏蔽位 1726 后面都有数量“nodeNumber”的值和数量“fieldIndex”的值指定的属性字段的值。

[1140] 每一个属性字段的值都由过程“解码字段值”来确定。

[1141] 10. 在操作 5790 中,如果“maskAccess”的值为 0 (“假”),则 SFNode 结构具有图 17C 所示的形式。在此情况下,maskAccess 位 1722 后面有 endFlag 位 1744。如果 endFlag 位的值为“1”1750,则已经达到 SFNode 结构的结尾。如果 endFlag 位的值为“0”1744,则 endFlag 位后面有 fieldRef 值 1746,以及数量“nodeNumber”的值和数量“fieldIndex”的值指定属性字段的值 1748。

[1142] 每一个属性字段的值都由过程“解码字段值”来确定。

[1143] 过程“解码字段值”

[1144] 每一个 SFNode 结构的每一个属性字段都具有特定的字段数据类型属性和特定的 fieldName 字符串。字段数据类型属性可以是下列类型之一:布尔、整数、浮点、vector2d、vector3d、旋转、颜色、时间、字符串、URL、节点或缓冲。每一个属性字段都被定义为 SFField(单值字段)或 MFField(多值字段)。

[1145] 每一个属性字段的字段数据类型属性、fieldName 字符串,以及 MFField/SFField 属性由 SFNode 结构的“nodeNumber”数量 5720 的值、“fieldIndex”或“fieldRef”数量的值,以及 MPEG-4 系统规范中定义的表来确定。

[1146] 在 SFField 属性字段的情况下,用于表示字段值的位的序列由字段数据类型属性和 MPEG-4 系统规范中提供的信息来确定。

[1147] 在 MFField 属性字段的情况下,字段值由零或多个 SFField 值的集合来确定。包括 MFField 属性字段的 SFField 值的集合具有图 17D 或图 17E 所描述的形式。不论是哪一种情况,第一位 1762、1782 都必须为零。下一位指定布尔量“isList”1766、1786 的值。

[1148] 如果数量“isList”的值为“1”,则 MFField 属性字段具有图 17D 所示的列表形式。在此情况下,isList 位 1766 后面有 endFlag 位 1770。如果 endFlag 位的值是“0”1770,则 endFlag 位后面有根据当前字段数据类型格式化的 SFField 值 1774。每一个这样的 SFField 值 1774 后面都有另一个 endFlag 位 1770。具有值“1”1772 的 endFlag 位结束列表形式 MFField。

[1149] 如果数量“isList”的值为“1”,则 MFField 属性字段具有图 17D 所示的列表形式。在此情况下,isList 位 1766 后面有 5 位整数值“numBits”1790。“numBits”的值后面有包括 numBits 位的整数值“nFields”1792。“nFields”的值后面有 nFields SFField 值,每一个值都根据当前字段数据类型格式化。

[1150] 在每一种情况下(SFField、列表形式 MFField 或矢量形式 MFField),每一个 SFField 值都可以转换为字符串。每一个这样的字符串都附加到属性字符串中,连续地附加的值由空格分隔。在字段数据类型为整数、浮点、vector2d、vector3d、旋转、颜色、时间的情况下,使用标准装置来将每一个数值都表示为字符串。在字段数据类型为布尔的情况下,值“0”和“1”由字符串“false”和“true”来表示。在字段数据类型为字符串和 URL 的情况下,每一个 SFField 值都用引号括起来。

[1151] 在字段数据类型为“node”的情况下,每一个 SFField 值都表示 SFNode 结构 1700、1710 或 1720。每一个这样的 SFNode 结构都叫做“子节点”。每一个这样的子节点都通过过程“解码 SFNode”的递归使用来解释。为每一个这样的子节点确定的字符串量“nodeName”的值被附加到当前属性字符串。此外,创建具有由字符串量“nodeName”的值指定的元素名称的新的 xml 元素,并附加到过程“解码 SFNode”的步骤 3 中创建的 BIFS 节点元素。

[1152] 在字段数据类型为“缓冲”的情况下,每一个 SFField 值都表示 sdsm BIFS 命令 1120。每一个这样的 BIFS 命令都使用过程“解码 sdsm 块”的操作 5330 到操作来解释,以当前节点元素充当父元素。将与每一个 BIFS 命令关联的 commandName 字符串(例如,“InsertNode”)附加到当前属性字符串。此外,创建具有由 commandName 字符串指定的元素名称的新的 xml 元素,并附加到过程“解码 SFNode”的步骤 3 中创建的 BIFS 节点元素。

[1153] 在完成每一个字段数据值的解释之后,将所产生的属性字符串指定到对应的 BIFS 节点元素的 fieldName 属性。即,将所产生的属性字符串指定到具有由指定的 BIFS 节点的指定的属性字段的 fieldName 字符串给出的属性名称的属性的属性值。

[1154] 构成以 QtInfo 数据结构表示的 moov 原子的 xml 表示。

[1155] 基于 QtInfo 数据结构 3100 创建 mp4file 文档 2300 中的第二个步骤是创建 mp4 二进制文件 700 中包含的 moov 原子 712、754 的 xml 表示。如图 58 所示,此步骤包括下列操作:

[1156] 1. 在操作 5800 中,创建新的“moov”元素 2320,并将它附加到 mp4file 文档 2300 的最高一级的元素。

[1157] 2. 在操作 5805 中,处理 QtInfo 数据结构 3100 的 MoovHeader 成员 3116。如图 31D 所示,MoovHeader 数据结构 3170 包括数量 size 3173、version&flags 3176、creationTime 3178、modifiedTime 3180、timeScale 3183、duration 3186,以及 nextTrackID 3188 的值。将这些值中的每一个值都指定到新的“moov”元素 2320 的名称类似的属性。

[1158] 3. 在操作 5810 中,执行过程“处理 Mp4fInitObjectDescr”。

[1159] 4. 在操作 5820 中,如果成员 UserData 3123 的值不为零,则执行过程“处理 UserData 数据结构”。

[1160] 5. 在操作 5825 中,将零值指定到整型量“iTrack”。

[1161] 6. 在操作 5830 中,将数量“iTrack”的值与 QtInfo 成员“numTracks”3118 的值进行比较。

[1162] 7. 在操作 5835 中,如果数量“iTrack”的值等于 QtInfo 成员“numTracks”的值,则此步骤完成。

[1163] 此时,创建中间 xml 文件 2250 和 2260 的过程完成。创建最后的 XMT-A 文档的过程从“基于中间 xml 文档创建 XMT-xml 文档”继续。

[1164] 8. 在操作 5840 中,如果数量“iTrack”的值不等于 QtInfo 成员“numTracks”的值,则创建新的 trak 元素 2350,并将其附加到在操作 5800 中创建的新 moov 元素 2320。这是“当前 trak 元素”。

[1165] 将 QtInfo 成员“trak array”3120 中的条目“iTrack”指定到表示 trak 结构 3200 的地址的数量“trakAtom”。由值“trakAtom”表示的 trak 结构是“当前 trak 结构”。

[1166] 9. 在操作 5850 中,执行过程“处理 TrakHeader 数据结构”。

[1167] 10. 在操作 5860 中,执行过程“处理 TrakMedia 数据结构”。

[1168] 11. 在操作 5870 中,如果当前 trak 结构的 TrackEdit 成员 3208 的值不为零,则执行过程“处理 TrackEdit 数据结构”。

[1169] 12. 在操作 5880 中,如果当前 trak 结构的 TrackReference 成员 3210 的值不为零,则执行过程“处理 TrackReference 数据结构”。

[1170] 13. 在操作 5890 中,将数量“iTrack”的值增大 1,然后重复将数量“iTrack”的值与数量“numTracks”进行比较的过程 5830。

[1171] 过程“处理 Mp4InitObjectDescr”

[1172] 过程“处理 Mp4InitObjectDescr”包括下列步骤:

[1173] 1. 创建新的“mp4fiods”元素 2330,并附加到“moov”元素 2320。

[1174] 2. 将 Mp4fInitObjectDescr 成员“ObjectDescrID”3138 的整数值指定到新的“mp4fiods”元素 2360 的“objectDescriptorID”属性 2370。

[1175] 3. 将成员“inlineProfiles”3146 的布尔值指定到新的 mp4fiods 元素 2360 的“includeInlineProfilesFlag”属性。

[1176] 4. 如果成员“urlFlag”3140 的布尔值为“真”,则将由成员“urlString”3143 表示的字符串指定到“mp4fiods”元素 2360 的“url”属性,从而完成 Mp4fInitObjectDescr 数据结构 3130 的处理。如果成员“urlFlag”3140 的布尔值为“假”,则将空字符串指定到 mp4fiods 元素 2360 的“url”属性。

[1177] 5. 基于 Mp4fInitObjectDescr 数据结构 3130 的“ProfileLevels”成员 3148 的对应的成员(ODProfileLevel、sceneProfileLevel、audioProfileLevel、visualProfileLevel,以及 graphicsProfileLevel),将整数值指定到 mp4fiods 元素 2360 的“ODProfileLevelIndication”、“sceneProfileLevelIndication”、“audioProfileLevelIndication”、“visualProfileLevelIndication”,以及“graphicsProfileLevelIndication”属性。

[1178] 6. 为 Mp4fInitObjectDescr 数据结构 3130 中的 EsIdInc 数组 3156 中的每一个 numEsIdIncs 3153 条目创建新的 EsIdInc 元素 2380。将每一个新的 EsIdInc 元素 2380 附加到 mp4fiods 元素 2360。EsIdInc 数组 3156 中的每一个条目都包括整数成员“trackID”。将成员“trackID”的值指定到对应的新的 EsIdInc 元素 2380 的“trackID”属性 2390。

[1179] 过程“处理 UserData 数据结构”

[1180] 过程“处理 UserData 数据结构”包括下列步骤:

[1181] 1. 创建新的“udta”元素 2340,并将它附加到新的“moov”元素 2320。

[1182] 如果 UserData 数据结构 3280 中的成员 numCprt 3284 的值大于零,则对于成员 cprt 数组 3286 中的每一个条目重复下列步骤:

[1183] 2. 创建新的“cprt”元素,并将它附加到新的“udta”元素 2340。

[1184] 3. 将 cprt 数组 3286 中的选定的条目表示的 cprt 结构 3290 的 version&flags 成员 3294 的值指定到新的“cprt”元素的“version”和“flags”属性。

[1185] 4. 将“cprt”结构 3290 中的成员“language”3296 的值指定新的“cprt”元素的“language”属性。

[1186] 5. 基于 cprt 数据结构 3290 中的成员“notice”3298,创建新的 xml 文本节点,并将此新文本节点附加到“Cprt”元素。

[1187] 过程“处理 TrskHeader 数据结构”

[1188] 过程“处理 TrakHeader 数据结构”包括下列步骤:

[1189] 1. 将当前 trak 结构的 TrackHeader 成员的成员“version”3216 的值指定到当前“trak”元素 2600 的“version”属性。

[1190] 2. 将当前 trak 结构的 TrackHeader 成员的成员“flags”3216 的值指定到当前“trak”元素 2600 的“flags”属性。

[1191] 3. 将当前 trak 结构的 TrackHeader 成员的成员“creationTime”3218 的值指定到当前“trak”元素 2600 的“creationTime”属性。

[1192] 4. 将当前 trak 结构的 TrackHeader 成员的成员“modifiedTime”3220 的值指定到当前“trak”元素 2600 的“modifiedTime”属性。

[1193] 5. 将当前 trak 结构的 TrackHeader 成员的成员“duration”3222 的值指定到当前“trak”元素 2600 的“duration”属性。

[1194] 6. 将当前 trak 结构的 TrackHeader 成员的成员“trackID”3224 的值指定到当前“trak”元素 2600 的“trackID”属性。

[1195] 过程“处理 TrakMedia 数据结构”

[1196] 过程“处理 TrakMedia 数据结构”包括下列步骤：

[1197] 1. 创建新的“mdia”元素 2604, 并将它附加到当前的“trak”元素 2600。

[1198] 2. 执行过程“处理 MediaHeader 数据结构”。

[1199] 3. 执行过程“处理 Handler 数据结构”。

[1200] 4. 执行过程“处理 MediaInfo 数据结构”。

[1201] 过程“处理 TrackEdit 数据结构”

[1202] 过程“处理 TrackEdit 数据结构”包括下列步骤：

[1203] 1. 创建新的 edts 元素 2644, 并将它附加到当前 trak 元素。

[1204] 如果当前 TrackEdit 数据结构 3208、3236 的成员 EditList3242 不为零, 则执行下列补充步骤：

[1205] 2. 创建新的 elst 元素 2648, 并将它附加到新的 edts 元素 2644。

[1206] 3. 将当前 TrackEdit 数据结构 3208、3236 的 EditList 成员 3242 表示的 EditList 数据结构 3244 的成员 version&flags 3248 的值指定到新的“elst”元素 2648 的“version”和“flags”属性。

[1207] 4. 将零值指定到数量“segment”, 并将 EditList 数据结构 3244 中的成员“numSegments”3250 的值指定到数量“numSegments”。数量“numSegments”的值预计为“2”。

[1208] 5. 对于 EditList 数据结构 3244 的 EditSegment 数组成员 3252 中的每一个 numSegments 条目, 重复下列步骤。这些条目中的每一个条目都包括 EditSegment 结构 3256。

[1209] 6. 创建新的“segment”元素, 并附加到新的“elst”元素 2648。

[1210] 7. 将 EditSegment 结构 3256 的成员“startTime”3258 的值指定到新的“segment”元素的“startTime”属性。

[1211] 8. 将 EditSegment 结构 3256 的成员“duration”3260 的值指定到新的“segment”元素的“duration”属性。

[1212] 9. 将 EditSegment 结构 3256 的成员“relativeRate”3262 的值指定到新的“segment”元素的“relativeRate”属性。

[1213] 过程“处理 TrackReference 数据结构”

[1214] 过程“处理 TrackReference 数据结构”包括下列步骤：

[1215] 1. 创建新的“tref”元素 2636, 并将它附加到当前的 trak 元素 2600。

[1216] 如果当前 TrackReference 数据结构 3210、3264 的成员 Mpod 3268 不为零, 则执行下列补充步骤：

[1217] 2. 创建新的“mpod”元素 2640, 并将它附加到新的“tref”元素 2636。

[1218] 3. 将零值指定到数量“entry”, 并 Mpod 结构 3268、3270 中的成员“numEntries”3274 的值指定到数量“numEntries”, 并创建名为“trackIdList”的新字符串(作为空字符串)。

[1219] 4. 如果数量“numEntries”的值大于零, 则对于 Mpod 结构 3270 中的成员 trackID 数组 3276 中的每一个 numEntries 条目, 重复下列三个步骤：

[1220] a. 如果数量“entry”的值大于零, 则将空格附加到字符串“trackIdList”。

[1221] b. 创建 trackID 数组 3276 中的条目“entry”表示的整数值的字符串表示, 并将结果附加到字符串“trackIdList”。

[1222] c. 将数量“entry”的值增大 1。

[1223] 5. 将字符串“trackIdList”的值指定到 mpod 元素 2640 的 trackID 属性。

[1224] 过程“处理 MediaHeader 数据结构”

[1225] 过程“处理 MediaHeader 数据结构”包括下列步骤：

[1226] 1. 将 MediaHeader 结构 3230、3300 的成员“version”3306 的值指定到当前“mdia”元素 2604 的“version”属性。

[1227] 2. 将 MediaHeader 结构 3230、3300 的成员“flags”3306 的值指定到当前“mdia”元素 2604 的“flags”属性。

[1228] 3. 将 MediaHeader 结构 3230、3300 的成员“creationTime”3308 的值指定到当前“mdia”元素 2604 的“creationTime”属性。

[1229] 4. 将 MediaHeader 结构 3230、3300 的成员“modifiedTime”3310 的值指定到当前“mdia”元素 2604 的“modifiedTime”属性。

[1230] 5. 将 MediaHeader 结构 3230、3300 的成员“timeScale”3313 的值指定到当前“mdia”元素 2604 的“timeScale”属性。

[1231] 6. 将 MediaHeader 结构 3230、3300 的成员“duration”3316 的值指定到当前“mdia”元素 2604 的“duration”属性。

[1232] 7. 将 MediaHeader 结构 3230、3300 的成员“language”3318 的值指定到当前“mdia”元素 2604 的“language”属性。

[1233] 过程“处理 Handler 数据结构”

[1234] 过程“处理 Handler 数据结构”包括下列步骤：

[1235] 1. 创建新的“hdlr”元素 2608, 并将它附加到当前“mdia”元素 2604。

[1236] 2. 将 Handler 结构 3232、3320 的成员“version”3326 的值指定到新的“hdlr”元素 2608 的“version”属性。

[1237] 3. 将 Handler 结构 3232、3320 的成员“flags”3326 的值指定到新的“hdlr”元素 2608 的“flags”属性。

[1238] 4. 将 Handler 结构 3232、3320 的成员“handlerType”3328 的值指定到新的“hdlr”

元素 2608 的“handlerType”属性。

[1239] 5. 将 Handler 数据结构 3232、3320 的成员“name”3330 的值指定到新的“hdlr”元素 2608 的“name”属性。

[1240] 过程“处理 MediaInfo 数据结构”

[1241] 过程“处理 MediaInfo 数据结构”包括下列步骤：1. 创建新的“minf”元素 2612，并将它附加到当前“mdia”元素 2604。

[1242] 2. 执行过程“处理 MediaDataInfo 数据结构”。

[1243] 3. 执行过程“处理 MediaInfoHeader 数据结构”。

[1244] 4. 执行过程“处理 SampleTables 数据结构”。

[1245] 过程“处理 MediaDataInfo 数据结构”

[1246] 过程“处理 MediaDataInfo 数据结构”包括下列步骤：

[1247] 1. 创建新的“dinf”元素 2616，并将它附加到当前“minf”元素 2612。

[1248] 2. 如果 MediaDataInfo 结构 3338 的成员 DataRefTable3358 不是零，则执行下列步骤：

[1249] 3. 创建新的“dref”元素 2620，并将它附加到新“dinf”元素 2616。

[1250] 4. 将由 MediaDataInfo 结构 3338、3350 中的成员 DataRefTable 3358 表示的成员“version”3366 的值指定到新的“dref”元素 2620 的“version”属性。

[1251] 5. 将 DataRefTable 结构 3360 的成员“flags”3366 的值指定到新的“dref”元素 2620 的“flags”属性。

[1252] 6. 对于 DataRefTable 结构 3360 中的成员 dataRef 数组 3370 中的每一个条目，重复下列步骤：.

[1253] 7. 创建新的“urlData”元素 2624，并将其附加到新“dref”元素 2620。

[1254] 8. 将由 DataRef 数组 3370 中的选定条目表示的 dataRef 结构 3380 中的成员“version”3386 的值指定到新的“urlData”元素 2624 的“version”属性。

[1255] 9. 将 dataRef 结构 3380 的成员“flags”3386 的值指定到新的“urlData”元素 2624 的“flags”属性。

[1256] 10. 如果 dataRef 结构 3380 的成员“flags”3386 的值不是“1”，成员“dataRefType”3388 的值是“url”（即，u-r-l-space），那么，将成员“dataRefData”3390 解释为字符串，并将结果指定到新的“urlData”元素 2624 的“location”属性。

[1257] 过程“处理 MediaInfoHeader 数据结构”

[1258] 给定 trak 结构内包含的 MediaInfoHeader 数据结构可以是三种类型中的某一种类型，具体取决于与 trak 结构关联的媒体数据的流类型。

[1259] 如果当前“trak”结构与场景描述流（sdsm）或对象描述符流（odsm）关联，那么，MediaInfoHeader 结构 3343 必须是“Mpeg4MediaHeader”数据结构。此结构具有成员“size”、“version”和“flags”。在此情况下，过程“处理 MediaInfoHeader 数据结构”包括下列步骤：

[1260] 1. 创建新的“nmhd”元素 2632，并将它附加到当前“minf”元素 2616。

[1261] 2. 将 MediaInfoHeader 结构 3343 的成员“version”的值指定到新的“nmhd”元素 2632 的“version”属性。

[1262] 3. 将 MediaInfoHeader 结构 3343 的成员“flags”的值指定到新的“nmhd”元素 2632 的“flags”属性。

[1263] 如果当前“trak”结构与音频流关联,那么,MediaInfoHeader 结构 3343 必须是“SoundMediaHeader”数据结构。此数据结构具有成员“size”、“version”、“flags”,和“balance”。在此情况下,过程“处理 MediaInfoHeader 数据结构”包括下列步骤:

[1264] 1. 创建新的“smhd”元素 2632,并将它附加到当前“minf”元素。

[1265] 2. 将 MediaInfoHeader 结构 3343 的成员“version”的值指定到新的“smhd”元素 2632 的“version”属性。

[1266] 3. 将 MediaInfoHeader 结构 3343 的成员“flags”的值指定到新的“smhd”元素 2632 的“flags”属性。

[1267] 4. 将 MediaInfoHeader 结构 3343 的成员“balance”的值指定到新的“smhd”元素 2632 的“balance”属性。

[1268] 如果当前“trak”结构与视频流关联,那么,MediaInfoHeaderstructure 3343 必须是“VideoMediaHeader”数据结构。此数据结构具有成员“size”、“version”、“flags”、“graphMode”、“opRed”、“opGreen”和“opBlue”。在此情况下,过程“处理 MediaInfoHeader 数据结构”包括下列步骤:

[1269] 1. 创建新的“vmhd”元素 2632,并将它附加到当前“minf”元素。

[1270] 2. 将 MediaInfoHeader 结构 3343 的成员“version”的值指定到新的“vmhd”元素 2632 的“version”属性。

[1271] 3. 将 MediaInfoHeader 结构 3343 的成员“flags”的值指定到新的“vmhd”元素 2632 的“flags”属性。

[1272] 4. 将 MediaInfoHeader 结构 3343 的成员“graphMode”的值指定到新的“vmhd”元素 2632 的“transferMode”属性。

[1273] 5. 将 MediaInfoHeader 结构 3343 的成员“opRed”的值指定到新的“vmhd”元素 2632 的“opColorRed”属性。

[1274] 6. 将 MediaInfoHeader 结构 3343 的成员“opGreen”的值指定到新的“vmhd”元素 2632 的“opColorGreen”属性。

[1275] 7. 将 MediaInfoHeader 结构 3343 的成员“opBlue”的值指定到新的“vmhd”元素 2632 的“opColorBlue”属性。

[1276] 过程“处理 SampleTables 数据结构”

[1277] 过程“处理 SampleTables 数据结构”包括下列步骤:

[1278] 1. 创建新的“stbl”元素 2628,并将它附加到当前“minf”元素 2612。

[1279] 2. 执行过程“处理 SampleToChunkTable 数据结构”。

[1280] 3. 执行过程“处理 ChunkOffsetTable 数据结构”。

[1281] 4. 执行过程“处理 TimeToSampleTable 数据结构”。

[1282] 5. 执行过程“处理 SampleSizeTable 数据结构”。

[1283] 6. 如果 SampleTables 3400 成员 SyncSampleTable 3416 的值不是零,则执行过程“处理 SyncSampleTable 数据结构”。

[1284] 7. 执行过程“处理 SampleDescrTable 数据结构”。

[1285] 过程“处理 SampleToChunkTable 数据结构”

[1286] SampleToChunkTable 数据结构包括样本表数据结构 3420, 其中, 成员“table array”中的每一个条目都包括 SampleToChunk 结构 3460。过程“处理 SampleToChunkTable 数据结构”包括下列步骤:

[1287] 1. 创建新的“stsc”元素 2656, 并将它附加到当前“stbl”元素 2628、2652。

[1288] 2. 将 SampleTables 成员 SampleToChunkTable 3406 指定到样本表结构 3420 “SampleToChunkTable”。

[1289] 3. 将 SampleToChunkTable 成员“version”3426 的值指定到新的“stsc”元素 2656 的“version”属性。

[1290] 4. 将 SampleToChunkTable 成员“flags”3426 的值指定到新的“stsc”元素 2656 的“flags”属性。

[1291] 5. 将 SampleToChunkTable 成员“numEntries”3428 的值指定到数量“numEntries”。将零值指定到数量“entry”, 并重复下列步骤, 直到数量“entry”的值等于数量“numEntries”的值:

[1292] 6. 创建新的“sampleToChunk”元素, 并将它附加到新的“stsc”元素 2656。

[1293] 7. 将 SampleToChunkTable 成员“table array”3430 中的条目“entry”指定到 SampleToChunk 结构 3460 “SampleToChunkTableEntry”。

[1294] 8. 将 SampleToChunkTableEntry 成员“firstChunk”3463 的值指定到新的“sampleToChunk”元素的“firstChunk”属性。

[1295] 9. 将 SampleToChunkTableEntry 成员“numSamples”3466 的值指定到新的“sampleToChunk”元素的“numSamples”属性。

[1296] 10. 将 SampleToChunkTableEntry 成员“description”3468 的值指定到新的“sampleToChunk”元素的“sampleDesc”属性。

[1297] 11. 将数量“entry”的值增大 1。

[1298] 过程“处理 ChunkOffsetTable 数据结构”

[1299] ChunkOffsetTable 数据结构 3408 包括样本表数据结构 3420, 其中, 成员“table array”中的每一个条目都包括整数序列。过程“处理 ChunkOffsetTable 数据结构”包括下列步骤:

[1300] 1. 创建新的“stco”元素 2664, 并将它附加到当前“stbl”元素 2652。

[1301] 2. 将 SampleTables 成员 ChunkOffsetTable 3408 指定到样本表结构 3420 “ChunkOffsetTable”。

[1302] 3. 将 ChunkOffsetTable 成员“version””的值指定到新的“stco”元素 2664 的“version”属性。

[1303] 4. 将 ChunkOffsetTable 成员“flags”的值指定到新的“stco”元素 2664 的“flags”属性。

[1304] 5. 将 ChunkOffsetTable 成员“numEntries”3428 的值指定到数量“numEntries”。将零值指定到数量“entry”, 并重复下列步骤, 直到数量“entry”的值等于数量“numEntries”的值:

[1305] 6. 创建新的“chunkOffset”元素, 并将它附加到新的“stco”元素 2664。

- [1306] 7. 将成员“table array”中的条目“entry”的值指定到数量“offset”。
- [1307] 8. 将数量“offset”的值指定到新的“chunkOffset”元素的“offset”属性。
- [1308] 9. 将数量“offset”的值与 QtInfo 成员“mdat array”3113 中的每一个条目 3160 中的成员“start”3166 和“size”3163 的值进行比较,以判断与指定的偏移值关联的 mdat 原子 706 的索引“mdatId”的值。数量“mdatId”的值通过“mdat array”3113 中的条目的索引来判断,数量“offset”的值大于成员“start”3166 的值,小于成员“start”3166 和“size”3163 的和。
- [1309] 10. 将 mdat 数组条目“mdatId”的成员“start”3163 的值指定到数量“mdatStart”。将“mdatStart”的值加“8”,并将数量“offset”的值减去所得的结果,并将结果指定到数量“mdatOffset”。
- [1310] 11. 将数量“mdatId”的值指定到新的“chunkOffset”元素的“mdatId”属性。
- [1311] 12. 将数量“mdatOffset”的值指定到新的“chunkOffset”元素的“mdatOffset”属性。
- [1312] 13. 将数量“entry”的值增大 1。
- [1313] 过程“处理 TimeToSampleTable 数据结构”
- [1314] TimeToSampleTable 数据结构 3410 包括样本表数据结构 3420,其中,成员“table array”中的每一个条目都包括 TimeToSample 结构 3470。过程“处理 TimeToSampleTable 数据结构”包括下列步骤:
- [1315] 1. 创建新的“stts”元素 2660,并将它附加到当前“stbl”元素。
- [1316] 2. 将 SampleTables 成员 TimeToSampleTable 3410 指定到样本表结构 3420 “TimeToSampleTable”。
- [1317] 3. 将 TimeToSampleTable 成员“version”3426 的值指定到新的“stts”元素 2660 的“version”属性。
- [1318] 4. 将 TimeToSampleTable 成员“flags”3426 的值指定到新的“stts”元素 2660 的“flags”属性。
- [1319] 5. 将 TimeToSampleTable 成员“numEntries”3428 的值指定到数量“numEntries”。将零值指定到数量“entry”,并重复下列步骤,直到数量“entry”的值等于数量“numEntries”的值:
- [1320] 6. 创建新的“timeToSample”元素,并将它附加到新的“stts”元素 2660。
- [1321] 7. 将 TimeToSampleTable 成员“table array”3430 中的条目“entry”指定到 TimeToSample 结构 3470 “TimeToSampleTableEntry”。
- [1322] 8. 将 TimeToSampleTableEntry 成员“count”3473 的值指定到新的“timeToSample”元素的“count”属性。
- [1323] 9. 将 TimeToSampleTableEntry 成员“duration”3476 的值指定到新的“timeToSample”元素的“duration”属性。
- [1324] 10. 将数量“entry”的值增大 1。
- [1325] 过程“处理 SampleSizeTable 数据结构”
- [1326] 过程“处理 SampleSizeTable 数据结构”包括下列步骤:
- [1327] 1. 创建新的“stsz”元素 2668,并将它附加到当前“stbl”元素 2652。

[1328] 2. 将 SampleTables 成员 SampleSizeTable 3413 指定到 SampleSizeTable 结构 3440 “SampleSizeTable”。

[1329] 3. 将 SampleSizeTable 成员 “version” 3446 的值指定到新的 “stsz” 元素 2668 的 “version” 属性。

[1330] 4. 将 SampleSizeTable 成员 “flags” 3446 的值指定到新的 “stsz” 元素 2668 的 “flags” 属性。

[1331] 5. 将 SampleSizeTable 成员 “numEntries” 3453 的值指定到数量 “numEntries”。

[1332] 6. 将数量 “numEntries” 的值指定到新的 “stsz” 元素 2668 的 “numSamples” 属性。

[1333] 7. 如果 SampleSizeTable 3413、3440 成员 “sampleSize” 3450 的值不是零, 则将 SampleSizeTable 3413、3440 成员 “sampleSize” 3450 的值指定到新的 “stsz” 元素的 “sizeForAll” 属性。在此情况下, 此过程完成。

[1334] 8. 如果 SampleSizeTable 3413、3440 成员 “sampleSize” 3450 的值为零, 则将零值指定到数量 “entry”, 并重复下列步骤, 直到数量 “entry” 的值等于数量 “numEntries” 的值:

[1335] 9. 创建新的 “sampleSize” 元素, 并将它附加到新的 “stsz” 元素 2668。

[1336] 10. 将 SampleSizeTable 成员 “table array” 3456 中的条目 “entry” 的值指定到新的 “sampleSize” 元素的 “size” 属性。

[1337] 11. 将数量 “entry” 的值增大 1。

[1338] 过程“处理 SyncSampleTable 数据结构”

[1339] SyncSampleTable 数据结构 3416 包括样本表数据结构 3420, 其中, 成员 “table array” 中的每一个条目都包括整数序列。过程 “处理 SyncSampleTable 数据结构” 包括下列步骤:

[1340] 1. 创建新的 “stss” 元素 2672, 并将它附加到当前 “stbl” 元素 2652。

[1341] 2. 将 SampleTables 成员 SyncSampleTable 3416 指定到样本表结构 3420 “SyncSampleTable”。

[1342] 3. 将 SyncSampleTable 成员 “version” 3426 的值指定到新的 “stss” 元素 2672 的属性 “version”。

[1343] 4. 将 SyncSampleTable 成员 “flags” 3426 的值指定到新的 “stss” 元素 2672 的属性 “flags”。

[1344] 5. 将 SyncSampleTable 成员 “numEntries” 3428 的值指定到数量 “numEntries”。将零值指定到数量 “entry”, 并重复下列步骤, 直到数量 “entry” 的值等于数量 “numEntries” 的值:

[1345] 6. 创建新的 “syncSample” 元素, 并将它附加到新的 “stss” 元素 2672。

[1346] 7. 将 SyncSampleTable 成员 “table array” 3430 中的条目 “entry” 的值指定到新的 “syncSample” 元素的属性 “sampleNumber”。

[1347] 8. 将数量 “entry” 的值增大 1。

[1348] 过程“处理 SampleDescrTable 数据结构”

[1349] SampleDescrTable 数据结构 3418 包括样本表数据结构 3420, 其中, 成员

“table array”中的每一个条目都包括 SampleDescription 结构 3480。过程“处理 SampleDescrTable 数据结构”包括下列步骤：

[1350] 1. 创建新的“stsd”元素 2676, 并将它附加到当前“stbl”元素 2652。

[1351] 2. 将 SampleTables 成员 SampleDescrTable 3418 指定到样本表结构 3420 “SampleDescrTable”。

[1352] 3. 将 SampleDescrTable 成员“version”3426 的值指定到新的“stsd”元素 2676 的属性“version”。

[1353] 4. 将 SampleDescrTable 成员“flags”3426 的值指定到新的“stsd”元素 2676 的属性“flags”。

[1354] 5. 将 SampleDescrTable 成员“numEntries”3428 的值指定到数量“numEntries”。将零值指定到数量“entry”, 并重复下列步骤, 直到数量“entry”的值等于数量“numEntries”的值：

[1355] 6. 将 SampleDescrTable 成员“table array”3430 中的条目“entry”指定到 SampleDescription 结构 3480 “SampleDescription”。

[1356] 7. 如果 SampleDescription 成员“dataFormat”的值对应于“mp4a”, 则创建新的“mp4a”元素, 并将它附加到新的“stsd”元素 2676。如果 SampleDescription 成员“dataFormat”的值对应于“mp4v”, 则创建新的“mp4v”元素, 并将它附加到新的“stsd”元素 2676。如果 SampleDescription 成员“dataFormat”的值对应于“mp4s”, 则创建新的“mp4s”元素, 并将它附加到新的“stsd”元素 2676。所产生的元素 (“mp4a”、“mp4v”, 或 “mp4s”) 将被称为“mp4*”元素 2680。

[1357] 8. 将 SampleDescription 成员“dataReference”3488 的值指定到新的“mp4*”元素 2680 的属性“dataRefIndex”。

[1358] 9. 创建新的“esds”元素 2684, 并将它附加到新的“mp4*”元素 2680。

[1359] 10. 将 SampleDescription 成员“esdVersion”3490 的值指定到新的“esds”元素 2684 的属性“version”。

[1360] 11. 将 SampleDescription 成员“esdData”3496 的值指定到表示字节数组的地址的数量“esdData”, 并将零值指定到数量“esdPos”。

[1361] 12. 执行过程“解码 esdData 数组”。

[1362] 13. 将数量“entry”的值增大 1。

[1363] 执行过程“解码 esdData 数组”

[1364] 过程“解码 esdData 数组”包括下列步骤：

[1365] 1. 将数组“esdData”中的条目“esdPos”的值指定到数量“EsDescTag”, 并将数量“esdPos”的值增大“1”。

[1366] 2. 使用过程“获取标记大小”, 以获得数量“numBytes”的值。

[1367] 3. 如果数量“EsDescTag”的值等于常量“ES_DescrTag”的值 (3), 则执行下列步骤。否则, 退出。

[1368] 4. 创建新的“ES_Descr”元素 2688、2700, 并将它附加到当前“esds”元素 2684。

[1369] 5. 将数量“numBytes”的值指定到新的“ES_Descr”元素 2688、2700 的属性“size”。

[1370] 6. 将数组“esdData”中的条目“esdPos”和“esdPos+1”的值组合为 16 位整数,将结果指定到新的“ES_Descr”元素 2688 的“ES_ID”属性 2700,并将数量“esdPos”的值增大“2”。

[1371] 7. 将数组“esdData”中的条目“esdPos”的值指定到数量“flagsAndPriority”,然后,将此值的高阶 5 位指定到数量“priority”,将低阶 3 位指定到数量“flags”,并将数量“esdPos”的值增大“1”。

[1372] 8. 将数量“priority”的值指定到新的“ES_Descr”元素 2688、2700 的属性“priority”。

[1373] 9. 将数量“flags”的值指定到新的“ES_Descr”元素 2688、2700 的属性“flags”。

[1374] 10. 如果数量“flags”的低阶位为“1”,将值“真”指定到新的“ES_Descr”元素 2688 的属性“OCRstreamFlag”。

[1375] 11. 重复下列步骤,直到数量“numBytes”的值小于“1”:

[1376] 12. 将数组“esdData”中的条目“esdPos”的值指定到数量“descrTag”,并将数量“numBytes”的值减少“1”,并将数量“esdPos”的值增大“1”。

[1377] 13. 使用过程“获取标记大小”来确定数量“descrSize”的值,并将数量“numBytes”的值减少用于指定此数量的值的字节数量。

[1378] 14. 将数量“numBytes”的值减少数量“descrSize”的值。

[1379] 15. 如果数量“descrTag”的值等于常量“DecoderConfigDescrTag”的值 (4),则执行过程“处理解码器配置描述符”。

[1380] 16. 如果数量“descrTag”的值等于常量“SLConfigDescrTag”的值 (6),则执行过程“处理 sl 配置描述符”。

[1381] 17. 否则 (即,数量“descrTag”的值不等于“DecoderConfigDescrTag”或者“SLConfigDescrTag”中的任何一个),则将数量“esdPos”的值增大数量“infoSize”的值。

[1382] 过程“处理解码器配置描述符”

[1383] 过程“处理解码器配置描述符”包括下列步骤:

[1384] 1. 创建新的“DecoderConfigDescriptor”元素 2710,并将它附加到当前“ES_Descr”元素 2700。

[1385] 2. 将数量“descrSize”的值指定到新的“DecoderConfigDescriptor”元素 2710 的“size”属性。

[1386] 3. 将数组“esdData”中的条目“esdPos”的值指定到数量“objectType”,并指定到新的“DecoderConfigDescriptor”元素 2710 的属性“objectType”,将数量“esdPos”的值增大“1”。

[1387] 4. 将数组“esdData”中的条目“esdPos”的值指定到数量“streamType”,并将数量“esdPos”的值增大“1”。

[1388] 5. 如果数量“streamType”的第二位 (“2”位) 不是零,则将值“真”指定到新的“DecoderConfigDescriptor”元素 2710 的属性“upstream”。

[1389] 6. 将数量“streamType”的值除以 4 (向右移两位),并将结果指定到数量“streamType”和新的“DecoderConfigDescriptor”元素 2710 的属性“streamType”。

[1390] 7. 将数组“esdData”中的下三个条目组合成 24 位值,并将结果指定到新的

“DecoderConfigDescriptor”元素 2710 的属性“bufferSize”，并将数量“esdPos”的值增大“3”。

[1391] 8. 将数组“esdData”中的下四个条目组合成 32 位值，并将结果指定到新的“DecoderConfigDescriptor”元素 2710 的属性“maxBitrate”，并将数量“esdPos”的值增大“4”。

[1392] 9. 将数组“esdData”中的下四个条目组合成 32 位值，并将结果指定到新的“DecoderConfigDescriptor”元素 2710 的属性“avgBitrate”，并将数量“esdPos”的值增大“4”。

[1393] 10. 将数量“descrSize”的值减少 13。

[1394] 11. 如果数量“descrSize”的值大于零，则重复下列步骤，直到数量“descrSize”的值小于 1：

[1395] 12. 将数组“esdData”中的下一条目指定到数量“infoTag”，并将数量“descrSize”的值减少“1”。

[1396] 13. 使用过程“获取标记大小”来确定跟随在指定到数量“infoTag”的值之后的标记大小值的值，并将结果指定到数量“infoSize”，并将数量“descrSize”的值减少用于指定此数量的值的字节数量。

[1397] 14. 将数量“descrSize”的值减少数量“infoSize”的值。

[1398] 15. 如果数量“infoTag”的值等于常量“DecoderSpecificInfoTag”的值 (5)，则执行过程“处理解码器特定信息”。否则，将数量“esdPos”的值增大数量“infoSize”的值。

[1399] 过程“处理 sl 配置描述符”

[1400] 过程“处理 sl 配置描述符”包括下列步骤：

[1401] 1. 创建新的“SLConfigDescriptor”元素 2760，并将它附加到当前“ES_Descr”元素 2700。

[1402] 2. 将数组“esdData”中的条目“esdPos”的值指定到新的“SLConfigDescriptor”元素 2760 的属性“predefined”，并将数量“esdPos”的值增大“1”。

[1403] 过程“处理解码器特定信息”

[1404] “DecoderSpecificInfoTag”之后的数组“esdData”中的条目的内容取决于数量“streamType”和“objectType”的值。

[1405] 如果数量“streamType”的值为“3”，则执行过程“处理 BIFS 配置”。

[1406] 如果数量“streamType”的值为“4”，并且数量“objectType”的值为“108”，则执行过程“处理 JPEG 配置”。

[1407] 如果数量“streamType”的值为“4”，并且数量“objectType”的值不是“108”，则执行过程“处理视频配置”。

[1408] 如果数量“streamType”的值为“5”，则执行过程“处理音频配置”。

[1409] 过程“处理 BIFS 配置”

[1410] 过程“处理 BIFS 配置”包括下列步骤：这些步骤将“esdData”数组中的条目的“infoSize”的序列，从条目“esdPos”开始，当作位的序列。

[1411] 1. 创建新的“BIFS_DecoderConfig”元素 2720，并将它附加到当前“DecoderConfigDescriptor”元素 2710。

[1412] 2. 如果数量“objectType”的值为“2”，则从数组“esdData”（从条目“esdPos”开始）提取下列数量的值，并将每一个所产生的值指定到新的 BIFS_DecoderConfig 元素的名称类似的属性：

[1413] a. use3DMeshCoding(1 位)，

[1414] b. usePredictiveMFField(1 位)，

[1415] c. numNodeIdBits(5 位)，

[1416] d. numRouteIdBits(5 位)，

[1417] e. numPROTOIdbits(5 位)，

[1418] 3. 否则（即，数量“objectType”的值不是“2”），从数组“esddata”（从条目“esdPos”开始）提取下列数量的值，并将每一个所产生的值指定到新的 BIFS_DecoderConfig 元素的名称类似的属性：

[1419] a. numNodeIdBits(5 位)，

[1420] b. numRouteIdBits(5 位)，

[1421] 4. 提取数组“esdData”中的下一位的值的值。如果结果是“0”，则将值“假”指定到新的 BIFS_DecoderConfig 元素的属性“isCommandStream”，并跳到步骤 8。如果结果是“1”，则将值“真”指定到属性“isCommandStream”，并执行下列步骤。

[1422] 5. 提取数组“esdData”中的下一位的值的值。如果结果是“0”，则将值“假”指定到新的 BIFS_DecoderConfig 元素的属性“pixelmetric”。否则，将值“真”指定到属性“pixel metric”。

[1423] 6. 提取数组“esdData”中的下一位的值的值。如果结果是“0”，则将值“假”指定到新的 BIFS_DecoderConfig 元素的属性“hasSize”，并跳到步骤 8。如果结果是“1”，则将值“真”指定到属性“hasSize”，并执行下列步骤。

[1424] 7. 从数组“esdData”（从条目“esdPos”开始）提取下列数量的值，并将每一个所产生的值指定到新的 BIFS_DecoderConfig 元素的名称类似的属性：

[1425] a. pixelwidth(16 位)，

[1426] b. pixelHeight(16 位)。

[1427] 8. 将数量“esdPos”的值增大数量“infoSize”的值。

[1428] 过程“处理 JPEG 配置”

[1429] 过程“处理 JPEG 配置”包括下列步骤：

[1430] 1. 创建新的“JPEG_DecoderConfig”元素 2730，并将它附加到当前元素。
“DecoderConfigDescriptor”2710

[1431] 2. 将数量“infoSize”的值指定到新的“JPEG_DecoderConfig”元素 2730 的“size”属性。

[1432] 3. 将数组“esdData”中的下两个条目组合成 16 位值，并将结果指定到新的“JPEG_DecoderConfig”元素 2730 的属性“headerLength”，并将数量“esdPos”的值增大“2”。

[1433] 4. 将数组“esdData”中的下两个条目组合成 16 位值，并将结果指定到新的“JPEG_DecoderConfig”元素 2730 的属性“Xdensity”，并将数量“esdPos”的值增大“2”。

[1434] 5. 将数组“esdData”中的下两个条目组合成 16 位值，并将结果指定到新的“JPEG_DecoderConfig”元素 2730 的属性“Ydensity”，并将数量“esdPos”的值增大“2”。

[1435] 6. 将数组“esdData”中的条目“esdPos”的值指定到新的“JPEG_DecoderConfig”元素 2730 的属性“numComponents”,并将数量“esdPos”的值增大“1”。

[1436] 过程“处理视频配置”

[1437] 过程“处理视频配置”包括下列步骤：

[1438] 1. 创建新的“VisualConfig”元素 2740,并将它附加到当前“DecoderConfigDescriptor”元素 2710。

[1439] 2. 将数量“infoSize”的值指定到新的“VisualConfig”元素 2740 的“size”属性。

[1440] 3. 将数量“esdPos”的值增大数量“infoSize”的值。

[1441] 过程“处理音频配置”

[1442] 过程“处理音频配置”包括下列步骤：

[1443] 1. 创建新的“AudioConfig”元素 2750,并将它附加到当前“DecoderConfigDescriptor”元素 2710。

[1444] 2. 将数量“infoSize”的值指定到新的“AudioConfig”元素 2750 的“size”属性。

[1445] 3. 将数量“esdPos”的值增大数量“infoSize”的值。

[1446] 基于中间 xml 文档对创建 XMT-A 文档。

[1447] 如图 59 所示,创建 XMT-A 文档 100 的过程包括下列步骤：

[1448] 1. 在操作 5900 中,使用标准 xml 装置创建新的空白 XMT-A 文档 100。

[1449] 2. 在操作 5920 中,执行过程“创建 XMT-A 标头”。

[1450] 3. 在操作 5940 中,使用标准 xml 装置创建新的空白 XMT-A Body 元素 120,并将它附加到新的 XMT-A 文档 100。

[1451] 4. 在操作 5960 中,执行过程“为 OD 命令创建 par 元素”。

[1452] 5. 在操作 5980 中,执行过程“为 BIFS 命令创建 par 元素”。

[1453] 下面将描述步骤 2、4 和 5。

[1454] 在这些步骤的准备过程中,将从属于 mp4file 文档的“moov”元素的“timeScale”属性的值指定到数量“timeScale”。

[1455] 过程“创建 XMT-A 标头元素”

[1456] 如图 60 所示,过程“创建 XMT-A 标头”包括下列步骤：

[1457] 1. 在操作 6000 中,使用标准 xml 装置创建新的 Header 元素 110,并将它附加到 XMT-A 文档 100。

[1458] 2. 在操作 6010 中,执行过程“创建 InitialObjectDescriptor”。

[1459] 3. 在操作 6020 中,执行过程“创建 profiles 元素”。

[1460] 4. 在操作 6030 中,执行过程“创建 Descr 元素”。

[1461] 5. 在操作 6040 中,使用标准 xml 装置选择 mp4file 文档 2300 中的从属于 mp4fiods 元素 2330、2360 的每一个元素。

[1462] 6. 在操作 6050 中,对于从属于 mp4fiods 元素 2330、2360 的每一个 EsIdInc 元素 2380,执行过程“处理 EsIdInc 元素”。

[1463] 步骤 2 中的过程“创建 InitialObjectDescriptor”包括下列操作：

[1464] a. 使用标准 xml 装置创建新的 InitialObjectDescriptor 元素 130,并将它附加

到新的 Header 元素 110。

[1465] b. 使用标准 xml 装置定位 mp4file 文档 2300 中包含的 moov 元素 2320。

[1466] c. 使用标准 xml 装置定位 moov 元素 2320 中包含的 mp4fiods 元素 2330、2360。

[1467] d. 创建新的字符串, 该字符串包括前缀“IODID:”, 后面是 mp4fiods 元素 2360 的 objectDescriptorID 属性 2370 的值, 并将此字符串的值指定到 InitialObjectDescriptor 元素 130 的 ODID 属性。

[1468] 步骤 3 中的过程“创建 Profiles 元素”包括下列操作:

[1469] a. 使用标准 xml 装置创建新的 profiles 元素 150, 并将它附加到新的 InitialObjectDescriptor 元素 130。

[1470] b. 将 mp4fiods 元素 2360 的 includeInlineProfilesFlag 属性的值指定到 Profiles 元素 150 的 includeInlineProfileLevelFlag 属性。

[1471] c. 基于 mp4fiods 元素 2360 的对应的属性的值指定到 Profiles 元素 150 的下列属性:

[1472] i. sceneProfileLevelIndication

[1473] ii. ODProfileLevelIndication

[1474] iii. audioProfileLevelIndication

[1475] iv. visualProfileLevelIndication

[1476] v. graphicsProfileLevelIndication

[1477] 指定到 Profiles 元素 150 的属性的值可以包括表示由 mp4fiods 元素 2360 的对应的属性表示的特定数值的字母数字字符串。例如, 可以使用字符串“None”表示数值“-1”或“255”。对于 mp4fiods 元素 2360 的属性允许的其他数值可以替换为 MPEG-4 系统规范中定义的字符串值。

[1478] 步骤 4 中的过程“创建 Descr 元素”包括下列操作:

[1479] a. 使用标准 xml 装置创建新的 Descr 元素 160, 并将它附加到新的 InitialObjectDescriptor 元素 130。

[1480] b. 使用标准 xml 装置创建新的 esDescr 元素 170, 并将它附加到新的 Descr 元素 160。

[1481] 步骤 5 包括使用标准 xml 装置选择 mp4file 文档 2300 中的从属于 mp4fiods 元素 2320、2360 的每一个元素 2380。此步骤中选定的元素预计是 EsIdInc 元素 2380。对于此步骤中标识的每一个 EsIdInc 元素 2380, 执行过程“处理 EsIdInc 元素”。

[1482] 过程“处理 EsIdInc 元素”

[1483] 对于从属于 mp4file 文档 2300 中的 mp4fiods 元素 2360 的每一个 EsIdInc 元素 2380, 执行下列操作:

[1484] 1. 在操作 6100 中, 将 EsIdInc 元素 2380 的 trackID 属性 2390 的值指定到数量“trackID”。

[1485] 2. 在操作 6110 中, 使用标准 xml 装置来查找 mp4file 文档 2300 中的从属于 moov 元素 2320 的 trak 元素 2350 (trak 元素 2350 的 trackID 属性的值匹配数量“trackID”的值)。

[1486] 3. 在操作 6120 中, 使用标准 xml 装置查找前面的步骤中标识的 trak 元素 2350、

2600 内包含的 ES_Descr 元素 2688。

[1487] 4. 在操作 6130 中,使用标准 xml 装置创建新的 ES_Descriptor 元素 180、190,并将它附加到 esDescr 元素 170。

[1488] 5. 在操作 6140 中,将在步骤“2”中确定的字符串量“trackType”的值与字符串“sdsm”进行比较。

[1489] 6. 在操作 6150 中,如果字符串量“trackType”的值为“sdsm”,则执行过程“为场景描述流创建 DecoderConfigDescriptor”。

[1490] 7. 在操作 6160 中,执行过程“创建 BIFSConfig”。

[1491] 8. 在操作 6170 中,如果字符串量“trackType”的值不是“sdsm”,将字符串量“trackType”的值与字符串“odsm”进行比较。

[1492] 9. 在操作 6180 中,如果字符串量“trackType”的值为“odsm”,则执行过程“为对象描述符流创建 DecoderConfigDescriptor”。

[1493] 10. 在操作 6190 中,执行过程“为 trak 从 tref 元素获取 mpodTrackIDs”。

[1494] 步骤 2 包括下列操作,这些操作在标识选择的 trak 元素之后执行:

[1495] a. 使用标准 xml 装置查找从属于指定的 trak 元素 2600 的 hdlr 元素 2608。

[1496] b. 将此 hdlr 元素的“name”属性的值指定到字符串量“trackName”。

[1497] c. 将此 hdlr 元素的“handlerType”属性的值指定到字符串量“trackType”。此数量的值预计是“sdsm”或“odsm”。

[1498] d. 通过将字符串“IOD”与数量“trackName”和字符串“ESID:1”的值连接来创建新的字符串量“ES_ID”,并将结果指定到新的 ES_Descriptor 元素 180、190 的“ES_ID”属性。

[1499] e. 创建新的 decConfigDescr 元素 646,并将它附加到 ES_Descriptor 元素 180、190、640。

[1500] f. 创建新的 DecoderConfigDescriptor 元素 650,并将它附加到 decConfigDescr 元素 646。

[1501] g. 创建新的 slConfigDescr 元素 660,并将它附加到 ES_Descriptor 元素 640。

[1502] h. 创建新的 slConfigDescr 元素 666,并将它附加到 slConfigDescr 元素 660。

[1503] i. 创建新的“predefined”元素,并将它附加到 SLConfigDescriptor 元素 666。

[1504] j. 将值“2”指定到“predefined”元素的“value”属性。

[1505] 为场景描述流创建 DecoderConfigDescriptor

[1506] 1. 使用标准 xml 装置查找 mp4file 文档 2300 中的所选择的 trak 元素 2600 内包含的 ES_Descr 元素 2688、2700 内包含的 DecoderConfigDescriptor 元素 2710。

[1507] 2. 将 DecoderConfigDescriptor 元素 2710 的“avgBitrate”和“maxBitrate”属性的值指定到 XMT-A 文档 100 中的新的 DecoderConfigDescriptor 元素 650 的对应的属性。

[1508] 3. 将 DecoderConfigDescriptor 元素 2710 的“bufferSize”属性的值指定到新的 DecoderConfigDescriptor 元素 650 的“bufferSizeDB”属性。

[1509] 4. 基于 mp4file 文档 2300 中的 DecoderConfigDescriptor 元素 2710 的“streamType”属性的数值,将字符串值指定到 XMT-A 文档 100 中的新的 DecoderConfigDescriptor 元素 650 的“streamType”属性。mp4file 文档 2300 中的

DecoderConfigDescriptor 元素 2710 的“streamType”属性的值预计匹配图 62A 中的第一列 6200 中的某一个条目。图 62A 中的第二列 6220 中的对应的条目被指定到 XMT-A 文档 100 中的新的 DecoderConfigDescriptor 元素 650 的“streamType”属性。

[1510] 在此情况下, mp4file 文档 2300 中的 DecoderConfigDescriptor 元素 2710 的 streamType 属性的值预计为“3”, 将值“SceneDescription”指定到 XMT-A 文档 100 中的新的 DecoderConfigDescriptor 元素 650 的“streamType”的属性。

[1511] 5. 基于 mp4file 文档 2300 中的 DecoderConfigDescriptor 元素 2710 的“streamType”属性的数值, 将字符串值指定到 XMT-A 文档 100 中的新的 DecoderConfigDescriptor 元素 650 的“objectTypeIndication”属性。mp4file 文档 2300 中的 DecoderConfigDescriptor 元素 2710 的“objectType”属性的值预计匹配图 62B 中的第一列 6240 中的某一个条目。图 62B 中的第二列 6260 中的对应的条目被指定到 XMT-A 文档 100 中的新的 DecoderConfigDescriptor 元素 650 的“objectTypeIndication”属性。

[1512] 6. 创建新的 decSpecificInfo 元素 656、680, 并将它附加到 DecoderConfigDescriptor 元素 650。

[1513] 创建 BiFSConfig

[1514] 过程“创建 BIFS 配置”包括下列步骤:

[1515] 1. 创建新的 BIFSConfig 元素 686, 并将它附加到新的 decSpecificInfo 元素 680。

[1516] 2. 使用标准 xml 装置查找从属于 DecoderConfigDescriptor 元素 2710 的 BIFS_DecoderConfig 元素 2720。

[1517] 3. 将 BIFS_DecoderConfig 元素 2720 的“nodeIdBits”属性的值指定到数量 nodeIdBits 以及 BIFSConfig 元素 686 的“nodeIdbits”属性。

[1518] 4. 将 BIFS_DecoderConfig 元素 2720 的“routeIdBits”属性的值指定到 BIFSConfig 元素 686 的“routeIdbits”属性。

[1519] 5. 如果数量“v2Config”的值为“真”, 则执行下列三个步骤:

[1520] a. 将 BIFS_DecoderConfig 元素 2720 的“protoIdbits”属性的值指定到 BIFSConfig 元素 686 的“PROTOIdbits”属性。

[1521] b. 将 BIFS_DecoderConfig 元素 2720 的“use3DMeshCoding”属性的值指定到 BIFSConfig 元素 686 的“use3DMeshCoding”属性。

[1522] c. 将 BIFS_DecoderConfig 元素的 2720 的“usePredictiveMFField”属性的值指定到 BIFSConfig 元素 686 的“usePredictiveMFField”属性。

[1523] 6. 如果 BIFS_DecoderConfig 元素 2710 的 commandStream”属性的值为“真”, 则执行下列步骤:

[1524] a. 创建新的“commandStream”元素 690, 并将它附加到 BIFSConfig 元素 686。

[1525] b. 将 BIFS_DecoderConfig 元素 2710 的“pixelMetric”属性的值指定到 commandStream 元素 686 的“pixelMetric”属性。

[1526] c. 创建新的“size”元素 696, 并将它附加到 commandStream 元素 690。

[1527] d. 将 BIFS_DecoderConfig 元素 2710 的“pixelHeight”属性的值指定到“size”元素 696 的“pixelHeight”属性。

[1528] e. 将 BIFS_DecoderConfig 元素 2710 的“pixelWidth”属性的值指定到“size”元

素 696 的“pixelWidth”属性。

[1529] 为对象描述符流创建 DecoderConfigDescriptor

[1530] 1. 使用标准 xml 装置查找 mp4file 文档 2300 中的所选择的 trak 元素 2600 内包含的 ES_Descr 元素 2688、2700 内包含的 DecoderConfigDescriptor 元素 2710。

[1531] 2. 将 DecoderConfigDescriptor 元素 2710 的“avgBitrate”和“maxBitrate”属性的值指定到 XMT-A 文档中的新的 DecoderConfigDescriptor 元素 650 的对应的属性。

[1532] 3. 将 mp4file 文档中的 trak 原子中的 DecoderConfigDescriptor 元素的“bufferSize”属性的值指定到新的 DecoderConfigDescriptor 元素 650 的“bufferSizeDB”属性。

[1533] 4. 基于 mp4file 文档 2300 中的 DecoderConfigDescriptor 元素 2710 的“streamType”属性的数值,将字符串值指定到 XMT-A 文档 100 中的新的 DecoderConCigDescriptor 元素 650 的“streamType”属性。mp4file 文档 2300 中的 DecoderConfigDescriptor 元素 2710 的“streamType”属性的值预计匹配图 62A 中的第一列 6200 中的某一个条目。图 62A 中的第二列 6220 中的对应的条目被指定到 XMT-A 文档 100 中的新的 DecoderConfigDescriptor 元素 650 的“streamType”属性。

[1534] 在此情况下,mp4file 文档 2300 中的 DecoderConfigDescriptor 元素 2710 的 streamType 属性的值预计为“1”,将值“ObjectDescriptor”指定到 XMT-A 文档 100 中的新的 DecoderConfigDescriptor 元素 650 的“streamType”的属性。

[1535] 5. 基于 mp4file 文档 2300 中的 DecoderConfigDescriptor 元素 2710 的“streamType”属性的数值,将字符串值指定到 XMT-A 文档 100 中的新的 DecoderConCigDescriptor 元素 650 的“objectTypeIndication”属性。mp4file 文档 2300 中的 DecoderConfigDescriptor 元素 2710 的“objectType”属性的值预计匹配图 62B 中的第一列 6240 中的某一个条目。图 62B 中的第二列 6260 中的对应的条目被指定到 XMT-A 文档 100 中的新的 DecoderConfigDescriptor 元素 650 的“obj ectTypeIndication”属性。

[1536] 为 trak 从 tref 元素获取 mpodTrackIDs

[1537] 过程“为 trak 从 tref 元素获取 mpodTrackIDs”包括下列步骤:

[1538] 1. 使用标准 xml 装置查找从属于选择的 trak 元素 2600 的 tref 元素 2636。

[1539] 2. 使用标准 xml 装置查找从属于选择的 tref 元素 2636 的 mpod 元素 2640。

[1540] 3. 将“mpod”元素 2640 的“trackID”属性的值指定到数量“mpodTrackIds”。

[1541] 过程“为 OD 命令创建 par 元素”

[1542] 表示 Object Descriptor 命令元素的 XMT-A “par”元素 140 是从表示 Object Descriptor 流 (odsm) 的 mp4file 文档 2300 的元素 2460 派生而来。XMT-A 文档 100 中表示的每一个 ObjectDescriptor 命令都还可以包括 ES_Descriptor 元素 630、其中每一个都是从 mp4file 文档 2300 中的 trak 元素 2350、2600 中包含的 ES_Descriptor 元素 2688 派生而来的。

[1543] 在为创建这些“par”元素进行准备的过程中,使用字符串量“mpodTrackIds”来创建 trackId 值的数组 pMpodTrackId。如果对于对应的 mp4 文件存在对象描述符流 (odsm),则在创建 XMT-AHeader 元素 110 的过程中,指定 mpodTrackIds 的值。如果没有为此数量指定值,那么没有对象描述符流,并且不生成“par”元素,以表示 Object Descriptor 命令元

素,此步骤完成。如果对应的 mp4 文件包括除 sdsm 和 odsm 外的一个或多个媒体数据流,那么,数量“mpodTrackIds”的值将包含整数 trackID 值的序列,数组 pMpodTrackId 的每一个元素都将表示此序列的对应的成员。

[1544] 如图 63 所示,用于创建表示 Object Descriptor 命令的“par”元素的过程包括下列步骤:

[1545] 1. 在操作 6300 中,使用标准 xml 装置查找 mp4file 文档 2300 内的全部 odsm 元素 2420。这通过首次使用标准 xml 装置来选择 mp4file 文档内的每一个 mdat 元素 2310 来实现。然后使用标准 xml 装置来选择每一个 mdat 元素 2400 内的每一个 odsm 元素 2420。

[1546] 2. 在操作 6310 中,使用标准 xml 装置来选择每一个 odsm 元素 2460 内的每一个 odsmChunk 元素 2470。

[1547] 3. 在操作 6320 中,使用标准 xml 装置来选择每一个 odsmChunk 元素 2500 内的每一个 odsmSample 元素 2510。

[1548] 4. 在操作 6330 中,使用标准 xml 装置来创建新的空白“par”元素 140,并将其附加到 XMT-A Body 元素 120。

[1549] 5. 在操作 6340 中,将 odsmSample 元素 2510 的“time”属性 2515、2525 的值除以数量“timeScale”的值,并将结果指定到新的“par”元素 140 的“begin”属性。

[1550] 6. 在操作 6350 中,使用标准 xml 装置来选择每一个 odsmSample 元素 2520 内包含的每一个 odsm-command 元素 2530。

[1551] 7. 在操作 6360 中,将每一个 odsm-command 元素 2530 的元素名称与字符串“ObjectDescrUpdate”进行比较。

[1552] 8. 在操作 6370 中,如果 odsm-command 元素的元素名称为“ObjectDescrUpdate”,则执行过程“创建 ObjectDescrUpdate 元素”。

[1553] 9. 在操作 6380 中,如果 odsm command 元素 2530 的元素名称不是“ObjectDescrUpdate”,则将 odsm-command 2530 的元素名称与“ObjectDescrRemove”进行比较。

[1554] 10. 在操作 6390 中,如果 odsm-command 元素的元素名称为“ObjectDescrRemove”,则执行过程“创建 ObjectDescrRemove 元素”。

[1555] 创建 ObjectDescriptorUpdate 元素

[1556] 过程“创建 ObjectDescriptorUpdate 元素”包括下列步骤:

[1557] 1. 使用标准 xml 装置创建新的“ObjectDescriptorUpdate”元素 220,并将它附加到当前“par”元素 210。

[1558] 2. 使用标准 xml 装置创建新的“OD”元素 230,并将它附加到当前“ObjectDescriptorUpdate”元素 220。

[1559] 3. 使用标准 xml 装置来查找从属于 ObjectDescrUpdateodsm-command 元素 2540 的“ObjectDescriptor”元素 2550。

[1560] 4. 使用标准 xml 装置创建新的“ObjectDescriptor”元素 240、600,并将它附加到当前“OD”元素 230。

[1561] 5. 将 ObjectDescriptor 元素 2550 的 ODID 属性 2555 的值指定到字符串量“ODID”。

- [1562] 6. 将字符串量“ODID”的值与前缀“ODID:”连接,并将结果指定到新的 ObjectDescriptor 元素 240、606 的 objectDescriptorID 属性,“objectDescriptorID”表示为“ODID”。
- [1563] 7. 使用标准 xml 装置创建新的“Descr”元素 610,并将它附加到新的 ObjectDescriptor 元素 600。
- [1564] 8. 使用标准 xml 装置创建新的“esDescr”元素 620,并将它附加到新的“Descr”元素 610。
- [1565] 9. 使用标准 xml 装置创建新的“ES_Descriptor”元素 630,并将它附加到当前元素。“esDescr”620
- [1566] 10. 使用标准 xml 装置选择从属于 ObjectDescriptor 元素 2550 的“EsIdRef”元素 2560。
- [1567] 11. 将“EsIdRef”元素 2560 的“EsId”属性 2565 的值指定到数量“EsId”。
- [1568] 12. 使用标准 xml 装置创建新的“decConfigDescr”元素 646,并将它附加到新的“ES_Descriptor”元素 630、640。
- [1569] 13. 使用标准 xml 装置创建新的“DecoderConfigDescriptor”元素 650,并将它附加到新的“decConfigDescr”元素 640。
- [1570] 14. 将数组 pMpodTrackId 中的条目 (EsId-1) 的值指定到数量“mpodTrackID”。例如,如果“EsId”的值为“1”,则通过数组 pMpodTrackId 中的条目“0”(第一个条目)确定 mpodTrackID 的值。
- [1571] 15. 使用标准 xml 装置来选择其 trackID 属性的值匹配数量“mpodTrackID”的值的 trak 元素 2350。
- [1572] 16. 使用标准 xml 装置选择从属于选择的 trak 元素 2350、2600 的 hdlr 元素 2608。
- [1573] 17. 将所选择的 hdlr 元素 2608 的“name”属性的值指定到字符串量“trackName”。
- [1574] 18. 将所选择的 hdlr 元素 2608 的“handlerType”属性的值指定到字符串量“trackType”。
- [1575] 19. 通过将字符串“ESID:”与数量“trackName”的值和数量“EsId”的值连接来创建新的字符串量,并将结果指定到新的“ES_Descriptor”元素 640 的“ES_ID”属性 636。
- [1576] 20. 使用标准 xml 装置创建新的“slConfigDescr”元素 660,并将它附加到当前“ES_Descriptor”元素 640。
- [1577] 21. 使用标准 xml 装置创建新的“SLConfigDescriptor”元素 666,并将它附加到新“slConfigDescr”元素 660。
- [1578] 22. 使用标准 xml 装置创建新的“predefined”元素,并将它附加到新的“SLConfigDescriptor”元素 666。
- [1579] 23. 将值“2”指定到新的“predefined”元素的“value”属性。
- [1580] 24. 执行过程“为媒体数据流创建 DecoderConfigDescriptor 元素”。
- [1581] 25. 执行过程“创建 StreamSource 元素”。
- [1582] 为媒体数据流创建 DecoderConfigDescriptor 元素
- [1583] 过程“为媒体数据流创建 DecoderConfigDescriptor 元素”包括下列步骤:
- [1584] 1. 使用标准 xml 装置选择过程“创建 ObjectDescriptorUpdate 元素”的步骤 15

中选定的 trak 元素 2350 内包含的 DecoderConfigDescriptor 元素 2710。

[1585] 2. 将字符串值指定到过程“创建 ObjectDescriptorUpdate 元素”的步骤 13 中创建的新的 DecoderConfigDescriptor 元素 650 的“streamType”属性。通过图 62A 中的第二列 6220 中的条目来确定此字符串的值。此条目对应于第一列 6200 上的匹配选定的 DecoderConfigDescriptor 2710 的“streamType”属性的数值的条目。

[1586] 3. 将字符串值指定到过程“创建 ObjectDescriptorUpdate 元素”的步骤 13 中创建的新的 DecoderConfigDescriptor 元素 650 的“objectTypeIndication”属性。通过图 62B 中的第二列 6260 中的条目来确定此字符串的值。此条目对应于第一列 6240 上的匹配选定的 DecoderConfigDescriptor 2710 的“objectType”属性的数值的条目。

[1587] 4. 将选定的 DecoderConfigDescriptor 元素 2710 的“avgBitrate”和“maxBitrate”属性的值指定到 XMT-A 文档中的新的 DecoderContigDescriptor 元素 650 的对应的属性。

[1588] 5. 将选定的 DecoderConfigDescriptor 元素 2710 的“bufferSize”属性的值指定到 XMT-A 文档中的新的 DecoderContigDescriptor 元素 650 的“bufferSizeDB”属性。

[1589] 6. 如果选定的 DecoderConfigDescriptor 元素 2710 包含从属的 JPEG_DecoderConfig 元素 2730、VisualConfig 元素 2740 或 AudioConfig 元素 2750,则执行下列步骤:

[1590] a. 使用标准 xml 装置创建新的“decSpecificInfo”元素 656,并将它附加到新的“DecoderConfigDescriptor”元素 650。

[1591] b. 使用标准 xml 装置创建新的“DecoderSpecificInfo”元素,并将它附加到当前“decSpecicInfo”元素 656。

[1592] c. 将值“auto”指定到新的 DecoderSpecificInfo 元素的“type”属性。

[1593] 创建 StreamSource 元素

[1594] 过程“创建 StreamSource 元素”包括下列步骤:

[1595] 1. 使用标准 xml 装置来选择 mp4file 文档 2300 中包含的每一个 mdat 元素 2310。

[1596] 2. 使用标准 xml 装置来选择每一个 mdat 元素 2310 中包含的每一个 mediaFile 元素 2430、2480。

[1597] 3. 将每一个所选择的 mediaFile 元素 2480 的“trackID”属性的值指定到数量“mdatTrackID”。

[1598] 4. 将数量“mdatTrackID”的值与数量“mpodTrackID”的当前值进行比较。

[1599] 5. 如果数量“mdatTrackID”的值匹配数量“mpodTrackID”的当前值,则执行下列步骤:

[1600] a. 使用标准 xml 装置创建新的“StreamSource”元素 670,并将它附加到当前“ES Descriptor”元素 640。

[1601] b. 将所选择的 mediaFile 元素 2480 的“name”属性的值与前缀“file:///”连接,并将结果指定到新的“StreamSource”元素 670 的“url”属性。

[1602] 创建 ObjectDescriptorRemove 元素

[1603] 过程“创建 ObjectDescriptorRemove 元素”包括下列步骤:

[1604] 1. 使用标准 xml 装置创建新的“ObjectDescriptorRemove”元素 250,并将它附加

到当前“par”元素 210。

[1605] 2. 将 odsm-sample 元素 2530 中的“ObjectDescriptorRemove”元素 2570 的“ODID”属性的值连接到前缀“ODID:”，并将结果指定到新的“ObjectDescriptorRemove”元素 250 的“objectDescriptorId”属性。XMT-A “ObjectDescriptorRemove”元素的“objectDescriptorId”属性表示为图 2B 中的“ODID”。

[1606] 过程“为 BIFS 命令创建 par 元素”

[1607] 如图 64 所示,过程“为 BIFS 命令创建 par 元素”包括下列步骤:

[1608] 1. 在操作 6400 中,使用标准 xml 装置来选择 mp4bifs 文档 2800 中的 bifsConfig 元素 2810。将“nodeIdBits”属性的值指定到数量“numNodeIdBits”。创建名为“pNodeNumber”的整数数组和名为“pNodeIdName”的字符串数组。这些数组中的每一个数组中的元素数量由 2 的 numNodeIdBits 次方指定。

[1609] 2. 在操作 6410 中,使用标准 xml 装置来选择 mp4bifs 文档 2800 中的每一个“commandFrame”元素 2820。

[1610] 3. 在操作 6420 中,使用标准 xml 装置来创建新的空白“par”元素 140,并将其附加到 XMT-A Body 元素 120。

[1611] 4. 在操作 6430 中,将 commandFrame 元素 2820 的“time”属性的值指定到“par”元素 140 的“begin”属性。

[1612] 5. 在操作 6440 中,使用当前 commandFrame 元素 2820 执行过程“建立节点号码和节点名称的数组”。

[1613] 6. 在操作 6450 中,使用当前 commandFrame 元素 2820 执行过程“将 mp4bifs commandFrame 元素转换为 XMT-A 元素”。

[1614] 建立节点号码和节点名称的数组

[1615] 图 65 中概述了过程“建立节点号码和节点名称的数组”。此过程包括下列步骤:

[1616] 1. 在操作 6500 中,使用标准 xml 装置选择当前 mp4bifscommandFrame 元素 2830 所拥有的每一个从属元素。每一个这样的从属元素都对应于 bifsCommand 元素 2840、2910。

[1617] 2. 在操作 6510 中,使用标准 xml 装置选择当前 bifsCommand 元素所拥有的每一个从属元素。如果所选择的从属元素对应于 bifsCommand 元素,则递归地重复此过程。否则,所选择的从属元素必须对应于 mp4bifsNode 元素,并执行下列步骤。

[1618] 3. 在操作 6520 中,将所选择的 mp4bifs Node 元素 2920、3000 的元素名称与字符串“ReusedNode”进行比较。如果所选择的 mp4bifs Node 元素 2920、3000 的元素名称为“ReusedNode”,则继续选择下面的 mp4bifs Node 元素 6510、6590(如果有的话)。

[1619] 4. 在操作 6530 中,如果所选择的 mp4bifs Node 元素 2920、3000 的元素名称不是“ReusedNode”,则使用标准 xml 装置来确定是否为所选择的 mp4bifs Node 元素 2920、3000 的“NodeId”属性指定了值。

[1620] 5. 在操作 6540 中,如果已经为所选择的 mp4bifs Node 元素 2920、3000 的“NodeId”属性指定了值,则将“NodeId”属性的数值指定到整型量“nodeId”。

[1621] 6. 在操作 6550 中,将所选择的 mp4bifs Node 元素 2920、3000 的名称与 MPEG-4 规范中定义的节点名称的数组中的每一个条目进行比较。将 MPEG-4 节点名称的数组中的匹配条目的索引指定到整型量 nodeNumber。将整型量 nodeNumber 的值指定到数组

pNodeNumber 中的条目 nodeId。

[1622] 7. 在操作 6560 中,使用标准 xml 装置来确定是否为所选择的 mp4bifsNode 元素 2920、3000 的“name”属性指定了值。

[1623] 8. 在操作 6570 中,如果已经为所选择的 mp4bifs Node 元素 2920、3000 的“name”属性指定了值,则将此“name”属性的值指定到数组 pNodeIdName 中的条目 nodeId。

[1624] 9. 在操作 6580 中,使用标准 xml 装置选择所选择的 mp4bifs Node 元素 2920、3000 所拥有的每一个从属元素 3030、3070。对于对应于 bifsCommand 元素 3070 的每一个从属元素,递归地重复步骤 26510。

[1625] 10. 在操作 6590 中,对于对应于 mp4bifs Node 元素 3030 的每一个从属元素,递归地重复步骤 36520。

[1626] 将 mp4bifs commandFrame 元素转换为 XMT-A 元素

[1627] 使用下列过程来将 mp4bifs commandFrame 元素 2830 的内容转换为 XMT-A BIFS 命令元素 300、310、320。在此情况下,“当前 mp4bi[^]元素”是在 6410 中选择的 mp4bifs commandFrame 元素 2830,“XMT-A 父元素”是在 6420 中创建的 XMT-A “par”元素。

[1628] 使用相同的过程来将 mp4fbifs Conditional Node 3040 转换为 XMT-A BIFS 命令元素 300、310、320。在此情况下,“当前 mp4bifs 元素”是从属于 mp4bifs 命令元素 2910 或 mp4bifs Node 元素 3000 的 mp4bifs Conditional Node 元素 3040,“XMT-A 父元素”是 XMT-A 命令元素 300、310、320 或 XMT-A CommandBuffernodeField 元素 410。

[1629] 1. 使用标准 xml 装置选择从属于当前 mp4bifs 元素的每一个 mp4bifs 元素。对每一个这样的从属元素都按下列步骤中的某一个步骤所描述的方式进行处理:

[1630] 2. 如果所选择的从属元素的元素名称为“InsertNode”,则执行过程“将 mp4bifs InsertNode 元素转换为 XMT-A 元素”。

[1631] 3. 如果所选择的从属元素的元素名称为“InsertIndexedValue”,则执行过程“将 mp4bifs InsertIndexedValue 元素转换为 XMT-A 元素”。

[1632] 4. 如果所选择的从属元素的元素名称为“InsertRoute”,则执行过程“将 mp4bifs InsertRoute 元素转换为 XMT-A 元素”。

[1633] 5. 如果所选择的从属元素的元素名称为“DeleteNode”,则执行过程“将 mp4bifs DeleteNode 元素转换为 XMT-A 元素”。

[1634] 6. 如果所选择的从属元素的元素名称为“DeleteIndexedValue”,则执行过程“将 mp4bifs DeleteIndexedValue 元素转换为 XMT-A 元素”。

[1635] 7. 如果所选择的从属元素的元素名称为“DeleteRoute”,则执行过程“将 mp4bifs DeleteRoute 元素转换为 XMT-A 元素”。

[1636] 8. 如果所选择的从属元素的元素名称为“ReplaceNode”,则执行过程“将 mp4bifs ReplaceNode 元素转换为 XMT-A 元素”。

[1637] 9. 如果所选择的从属元素的元素名称为“ReplaceIndexedValue”,则执行过程“将 mp4bifs ReplaceIndexedValue 元素转换为 XMT-A 元素”。

[1638] 10. 如果所选择的从属元素的元素名称为“ReplaceField”,则执行过程“将 mp4bifs ReplaceField 元素转换为 XMT-A 元素”。

[1639] 11. 如果所选择的从属元素的元素名称为“ReplaceRoute”,则执行过程“将

mp4bifs ReplaceRoute 元素转换为 XMT-A 元素”。

[1640] 12. 如果所选择的从属元素的元素名称为“ReplaceScene”，则执行过程“将 mp4bifs ReplaceScene 元素转换为 XMT-A 元素”。

[1641] 将 mp4bifs InsertNode 元素转换为 XMT-A 元素

[1642] 过程“将 mp4bifs InsertNode 元素转换为 XMT-A 元素”包括下列步骤：

[1643] 1. 创建新的 XMT-A “Insert”元素 300，并将它附加到 XMT-A 父元素。

[1644] 2. 处理 parentId 属性：

[1645] a. 将 mp4bifs“InsertNode”元素的“parentId”属性的值指定到整型量“nodeId”。

[1646] b. 将数组 pNodeIdName 中的条目 nodeId 的值指定到字符串量“nodeIdName”。

[1647] c. 如果字符串量“nodeIdName”的值为空值或没有内容（零长度），则将数量“nodeId”的值附加到前缀“nodeId:”，并将结果指定到数量“nodeIdName”。

[1648] d. 将数量“nodeIdName”的值指定到 XMT-A “Insert”元素 300 的“atNode”属性。

[1649] 3. 处理 insertionPosition 属性：

[1650] a. 将 mp4bifs “InsertNode”元素的“insertionPosition”属性的值指定到字符串量“position”。

[1651] b. 如果数量“position”的值为“0”，则将 mp4bifs “InsertNode”元素的“position”属性的值指定到 XMT-A “Insert”元素 300 的“position”属性。

[1652] c. 如果数量“position”的值为“2”，则将值“BEGIN”指定到 XMT-A “Insert”元素 300 的“position”属性。

[1653] d. 如果数量“position”的值为“3”，则将值“END”指定到 XMT-A “Insert”元素 300 的“position”属性。

[1654] 4. 处理从属 mp4bifs Node 元素：

[1655] a. 使用标准 xml 装置选择当前 bifsCommand 元素 2910 所拥有的每一个从属 Node 元素 2920。

[1656] b. 对于每一个这样的所选择的从属 Node 元素 2920，则执行过程“将 mp4bifs Node 元素转换为 XMT-A Node 元素”。

[1657] 将 mp4bifs InsertIndexedValue 元素转换为 XMT-A 元素

[1658] 过程“将 mp4bifs InsertIndexedValue 元素转换为 XMT-A 元素”包括下列步骤：

[1659] 1. 创建新的 XMT-A “Insert”元素 300，并将它附加到当前 XMT-A 父元素。

[1660] 2. 处理 nodeId 属性：

[1661] a. 将 mp4bifs “InsertNode”元素的“nodeId”属性的值指定到整型量“nodeId”。

[1662] b. 将数组 pNodeNumber 中的条目 nodeId 的值指定到整型量“nodeNumber”。

[1663] c. 将数组 pNodeIdName 中的条目 nodeId 的值指定到字符串量“nodeIdName”。

[1664] d. 如果字符串量“nodeIdName”的值为空值或没有内容（零长度），则将数量“nodeId”的值附加到前缀“nodeId:”，并将结果指定到数量“nodeIdName”。

[1665] e. 将数量“nodeIdName”的值指定到 XMT-A “Insert”元素的“atNode”属性。

[1666] 3. 处理 inFieldName 和 value 属性：

[1667] a. 将 mp4bifs “InsertIndexedvalue”元素的“inFieldName”属性的值指定到 XMT-A “Insert”元素的“atField”属性。

[1668] b. 将 mp4bifs “InsertIndexedValue” 元素的“value”属性的值指定到 XMT-A “Insert”元素的“value”属性。

[1669] 4. 处理 insertionPosition 属性：

[1670] a. 将 mp4bifs “InsertIndexedValue”元素的“insertionPosition”属性的值指定到字符串量“position”。

[1671] b. 如果数量“position”的值为“0”，则将 mp4bifs InsertNode”元素的“position”属性的值指定到 XMT-A “Insert”元素 300 的“position”属性。

[1672] c. 如果数量“position”的值为“2”，则将值“BEGIN”指定到 XMT-A “Insert”元素 300 的“position”属性。

[1673] d. 如果数量“position”的值为“3”，则将值“END”指定到 XMT-A “Insert”元素 300 的“position”属性。

[1674] 5. 处理从属 mp4bifs Node 元素：

[1675] a. 使用标准 xml 装置选择当前 bifsCommand 元素 2910 所拥有的每一个从属 Node 元素 2920。

[1676] b. 对于每一个这样的所选择的从属 Node 元素 2920，则执行过程“将 mp4bifs Node 元素转换为 XMT-A Node 元素”。

[1677] 将 mp4bifs InsertRoute 元素转换为 XMT-A 元素

[1678] 过程“将 mp4bifs InsertRoute 元素转换为 XMT-A 元素”包括下列步骤：

[1679] 1. 创建新的 XMT-A “Insert”元素 300，并将它附加到当前 XMT-A 父元素。

[1680] 2. 创建新的 XMT-A “Route”元素 340，并将它附加到新的 XMT-A “Insert”元素 300。

[1681] 3. 处理 routeId 属性：

[1682] a. 将 mp4bifs “InsertRoute”元素的“routeId”属性的值指定到字符串量“iRoute”。

[1683] b. 将 mp4bifs “InsertRoute”元素的“name”属性的值指定到字符串量“routeName”。

[1684] c. 如果字符串量“routeName”的值不是空值并包括至少一个字符，则将数量“routeName”指定到 XMT-A “ROUTE”元素 340 的“DEF”属性。否则，如果字符串量“iRoute”的值不是空值并包括至少一个字符，则将数量“iRoute”的值附加到前缀“route:”，并将结果指定到 XMT-A “ROUTE”元素 340 的“DEF”属性。

[1685] 4. 基于 mp4bifs “InsertRoute”元素，执行过程“将 mp4bifsRoute 元素的属性转换为 XMT-A ROUTE 元素的属性”。

[1686] 将 mp4bifs DeleteNode 元素转换为 XMT-A 元素

[1687] 过程“将 mp4bifs DeleteNode 元素转换为 XMT-A 元素”包括下列步骤：

[1688] 1. 创建新的 XMT-A “Delete”元素 310，并将它附加到当前 XMT-A 父元素。

[1689] 2. 处理 nodeId 属性：

[1690] a. 将 mp4bifs “DeleteNode”元素的“nodeId”属性的值指定到整型量“nodeId”。

[1691] b. 将数组 pNodeIdName 中的条目 nodeId 的值指定到字符串量“nodeIdName”。

[1692] c. 如果字符串量“nodeIdName”的值为空值或没有内容（零长度），则将数量

“nodeId”的值附加到前缀“nodeId:”，并将结果指定到数量“nodeIdName”。

[1693] d. 将数量“nodeIdName”的值指定到 XMT-A “Delete”元素 310 的“atNode”属性。

[1694] 将 mp4bifs DeleteIndexedValue 元素转换为 XMT-A 元素

[1695] 过程“将 mp4bifs DeleteIndexedValue 元素转换为 XMT-A 元素”包括下列步骤：

[1696] 1. 创建新的 XMT-A “Delete”元素 310，并将它附加到当前 XMT-A 父元素。

[1697] 2. 处理 nodeId 属性：

[1698] a. 将 mp4bifs “DeleteIndexedValue”元素的“nodeId”属性的值指定到整型量“nodeId”。

[1699] b. 将数组 pNodeNumber 中的条目 nodeId 的值指定到整型量“nodeNumber”。

[1700] c. 将数组 pNodeIdName 中的条目 nodeId 的值指定到字符串量“nodeIdName”。

[1701] d. 如果字符串量“nodeIdName”的值为空值或没有内容（零长度），则将数量“nodeId”的值附加到前缀“nodeId:”，并将结果指定到数量“nodeIdName”。

[1702] e. 将数量“nodeIdName”的值指定到 XMT-A “Delete”元素 310 的“atNode”属性。

[1703] 3. 处理 infield 属性：

[1704] a. 将 mp4bifs “DeleteIndexedValue”元素的“infield”属性的值指定到整型量“field”。

[1705] b. 将具有索引“nodeNumber”的 BIFS 节点的 MPEG-4 规范中定义的字段名称的数组中的条目“field”的值指定到字符串量“fieldName”。

[1706] c. 将数量“FieldName”的值指定到 XMT-A “Delete”元素 310 的“atField”属性。

[1707] 4. 处理 deletionPosition 属性：

[1708] a. 将 mp4bifs “DeleteIndexedValue”元素的“deletionPosition”属性的值指定到字符串量“position”。

[1709] b. 如果数量“position”的值为“0”，则将 mp4bifs “InsertNode”元素的“position”属性的值指定到 XMT-A “Delete”元素 310 的“position”属性。

[1710] c. 如果数量“position”的值为“2”，则将值“BEGIN”指定到 XMT-A “Delete”元素 310 的“position”属性。

[1711] d. 如果数量“position”的值为“3”，则将值“END”指定到 XMT-A “Delete”元素 310 的“position”属性。

[1712] 将 mp4bifs DeleteRoute 元素转换为 XMT-A 元素

[1713] 过程“将 mp4bifs DeleteRoute 元素转换为 XMT-A 元素”包括下列步骤：

[1714] 1. 创建新的 XMT-A “Delete”元素 310，并将它附加到当前 XMT-A 父元素。

[1715] 2. 将 mp4bifs “DeleteRoute”元素的“routeId”属性的值指定到字符串量“routeId”。

[1716] 3. 将数量“routeId”的值附加到前缀“route:”，并将结果指定到 XMT-A “Delete”元素 310 的“atRoute”属性。

[1717] 将 mp4bifs ReplaceNode 元素转换为 XMT-A 元素

[1718] 过程“将 mp4bifs ReplaceNode 元素转换为 XMT-A 元素”包括下列步骤：

[1719] 1. 创建新的 XMT-A “Replace”元素 320，并将它附加到当前 XMT-A 父元素。

[1720] 2. 处理 nodeId 属性：

- [1721] a. 将 mp4bifs “ReplaceNode”元素的“nodeId”属性的值指定到整型量“nodeId”。
- [1722] b. 将数组 pNodeIdName 中的条目 nodeId 的值指定到字符串量“nodeIdName”。
- [1723] c. 如果字符串量“nodeIdName”的值为空值或没有内容（零长度），则将数量“nodeId”的值附加到前缀“nodeId:”，并将结果指定到数量“nodeIdName”。
- [1724] d. 将数量“nodeIdName”的值指定到 XMT-A “Replace”元素 320 的“atNode”属性。
- [1725] 3. 处理从属 mp4bifs Node 元素：
- [1726] a. 使用标准 xml 装置选择当前 bifsCommand 元素 2910 所拥有的每一个从属 Node 元素 2920。
- [1727] b. 对于每一个这样的所选择的从属 Node 元素 2920，则执行过程“将 mp4bifs Node 元素转换为 XMT-A Node 元素”。
- [1728] 将 mp4bifs ReplaceIndexedValue 元素转换为 XMT-A 元素
- [1729] 过程“将 mp4bifs ReplaceIndexedValue 元素转换为 XMT-A 元素”包括下列步骤：
- [1730] 1. 创建新的 XMT-A “Replace”元素 320，并将它附加到当前 XMT-A 父元素（图 3C）。
- [1731] 2. 处理 nodeId 属性：
- [1732] a. 将 mp4bifs “ReplaceIndexedValue”元素的“nodeId”属性的值指定到整型量“nodeId”。
- [1733] b. 将数组 pNodeNumber 中的条目 nodeId 的值指定到整型量“nodeNumber”。
- [1734] c. 将数组 pNodeIdName 中的条目 nodeId 的值指定到字符串量“nodeIdName”。
- [1735] d. 如果字符串量“nodeIdName”的值为空值或没有内容（零长度），则将数量“nodeId”的值附加到前缀“nodeId:”，并将结果指定到数量“nodeIdName”。
- [1736] e. 将数量“nodeIdName”的值指定到 XMT-A “Replace”元素 320 的“atNode”属性。
- [1737] 3. 处理 inFieldName 和 value 属性：
- [1738] a. 将 mp4bifs “ReplaceIndexedValue”元素的“inFieldName”属性的值指定到 XMT-A “Replace”元素的“atField”属性。
- [1739] b. 将 mp4bifs “ReplaceIndexedValue”元素的“value”属性的值指定到 XMT-A “Replace”元素 320 的“value”属性。
- [1740] 4. 处理 replacementPosition 属性：
- [1741] a. 将 mp4bifs “ReplaceIndexedValue”元素的“replacementPosition”属性的值指定到字符串量“position”。
- [1742] b. 如果数量“position”的值为“0”，则将 mp4bifs “InsertNode”元素的“position”属性的值指定到 XMT-A “Replace”元素 320 的“position”属性。
- [1743] c. 如果数量“position”的值为“2”，则将值“BEGIN”指定到 XMT-A “Replace”元素 320 的“position”属性。
- [1744] d. 如果数量“position”的值为“3”，则将值“END”指定到 XMT-A “Replace”元素 330 的“position”属性。
- [1745] 5. 处理从属 mp4bifs Node 元素：
- [1746] a. 使用标准 xml 装置选择当前 bifsCommand 元素 2910 所拥有的每一个从属 Node

元素 2920。

[1747] b. 对于每一个这样的所选择的从属 Node 元素 2920, 则执行过程“将 mp4bifs Node 元素转换为 XMT-A Node 元素”。

[1748] 将 mp4bifs ReplaceField 元素转换为 XMT-A 元素

[1749] 过程“将 mp4bifs ReplaceField 元素转换为 XMT-A 元素”包括下列步骤：

[1750] 1. 创建新的 XMT-A “Replace”元素 320, 并将它附加到当前 XMT-A 父元素。

[1751] 2. 处理 nodeId 属性：

[1752] a. 将 mp4bifs “ReplaceField”元素的“nodeId”属性的值指定到整型量“nodeId”。

[1753] b. 将数组 pNodeNumber 中的条目 nodeId 的值指定到整型量“nodeNumber”。

[1754] c. 将数组 pNodeIdName 中的条目 nodeId 的值指定到字符串量“nodeIdName”。

[1755] d. 如果字符串量“nodeIdName”的值为空值或没有内容（零长度），则将数量“nodeId”的值附加到前缀“nodeId:”，并将结果指定到数量“nodeIdName”。

[1756] e. 将数量“nodeIdName”的值指定到 XMT-A “Replace”元素 320 的“atNode”属性。

[1757] 3. 处理 inFieldName 和 value 属性：

[1758] a. 将 mp4bifs “ReplaceField”元素的“inFieldName”属性的值指定到 XMT-A “Replace”元素的“atField”属性。

[1759] b. 将 mp4bifs “ReplaceField”元素的“value”属性的值指定到 XMT-A “Replace”元素 320 的“value”属性。

[1760] 4. 处理从属 mp4bifs Node 元素：

[1761] a. 使用标准 xml 装置选择当前 bifsCommand 元素 2910 所拥有的每一个从属 Node 元素 2920。

[1762] b. 对于每一个这样的所选择的从属 Node 元素 2920, 则执行过程“将 mp4bifs Node 元素转换为 XMT-A Node 元素”。

[1763] 将 mp4bifs ReplaceRoute 元素转换为 XMT-A 元素

[1764] 过程“将 mp4bifs ReplaceRoute 元素转换为 XMT-A 元素”包括下列步骤：

[1765] 1. 创建新的 XMT-A “Replace”元素 320, 并将它附加到当前 XMT-A 父元素。

[1766] 2. 将 mp4bifs “ReplaceRoute”元素的“routeId”属性的值指定到字符串量“routeId”。

[1767] 3. 将数量“routeId”的值附加到前缀“route:”，并将结果指定到 XMT-A “Replace”元素 310 的“atRoute”属性。

[1768] 4. 基于 mp4bifs “ReplaceRoute”元素, 执行过程“将 mp4bifsRoute 元素的属性转换为 XMT-A ROUTE 元素的属性”。

[1769] 将 mp4bifs ReplaceScene 元素转换为 XMT-A 元素

[1770] 过程“将 mp4bifs ReplaceScene 元素转换为 XMT-A 元素”包括下列步骤：

[1771] 1. 创建新的 XMT-A “Replace”元素 320, 并将它附加到当前 XMT-A 父元素。

[1772] 2. 创建新的 XMT-A “Scene”元素 370, 并将它附加到新的 XMT-A “Replace”元素 320。

[1773] 3. 将 ReplaceScene 元素 2930 的“USENAMES”属性的值指定到字符串量

“useNames”。如果数量 useNames 的值为“真”，将数量 useNames 的值指定到新的 XMT-A Scene 元素 370 的“USENAMES”属性。

[1774] 4. 使用标准 xml 装置选择从属于 ReplaceScene 元素 2930 所拥有的的每一个从属元素。对于每一个这样的所选择的从属元素，执行下列操作：

[1775] 5. 如果所选择的从属元素的名称为“Routes”2950，则执行下列步骤。否则，执行过程“将 mp4bifs Node 元素转换为 XMT-A Node 元素”。

[1776] 6. 使用标准 xml 装置选择选定的“Routes”元素 2950 所拥有的的每一个从属元素。每一个这样的从属元素预计为“Route”元素 2960。对于每一个这样的“Route”元素 2960，执行下列步骤：

[1777] 7. 创建新的 XMT-A “ROUTE”元素 390，并将它附加到新的 XMT-A “Scene”元素 370。

[1778] 8. 处理可选的 routeId 属性：

[1779] a. 将选定的“Route”元素 2960 的“routeId”属性的值指定到字符串量“iRoute”。

[1780] b. 将 mp4bifs “Route”元素的“name”属性的值指定到字符串量“routeName”。

[1781] c. 如果字符串量“routeName”的值不是空值并包括至少一个字符，则将数量“routeName”指定到 XMT-A “ROUTE”元素 390 的“DEF”属性。否则，如果字符串量“iRoute”的值不是空值并包括至少一个字符，则将数量“iRoute”的值附加到前缀“route:”，并将结果指定到 XMT-A “ROUTE”元素 390 的“DEF”属性。

[1782] 9. 基于 mp4bifs “Route”元素，执行过程“将 mp4bifs Route 元素的属性转换为 XMT-A ROUTE 元素的属性”。

[1783] 将 mp4bifs Route 元素的属性转换为 XMT-A ROUTE 元素的属性

[1784] 过程“将 mp4bifs Route 元素的属性转换为 XMT-A ROUTE 元素的属性”包括下列步骤。

[1785] 在此过程中引用的 mp4bifs Route 元素可以是 mp4bifs “Route”元素 2960、mp4bifs “InsertRoute”bifsCommand 元素 2910，或 mp4bifs “ReplaceRoute”bifsCommand 元素 2910。

[1786] 1. 处理 departureNode 值：

[1787] a. 将 mp4bifs Route 元素的“departureNode”属性的值指定到整型量“nodeId”。

[1788] b. 将数组 pNodeNumber 中的条目 nodeId 的值指定到整型量“nodeNumber”。

[1789] c. 将数组 pNodeIdName 中的条目 nodeId 的值指定到字符串量“nodeIdName”。

[1790] d. 如果字符串量“nodeIdName”的值为空值或没有内容（零长度），则将数量“nodeId”的值附加到前缀“nodeId:”，并将结果指定到数量“nodeIdName”。

[1791] e. 将数量“nodeIdName”的值指定到 XMT-A “ROUTE”元素 360 或 390 的“fromNode”属性。

[1792] 2. 处理 departureField 值：

[1793] a. 将 mp4bifs Route 元素的“departureField”属性的值指定到整型量“field”。

[1794] b. 将具有索引“nodeNumber”的 BIFS 节点的 MPEG-4 规范中定义的字段名称的数组中的条目“field”的值指定到字符串量“fieldName”。

[1795] c. 将数量“fieldName”的值指定到 XMT-A “ROUTE”元素 360 或 390 的“fromField”属性。

- [1796] 3. 处理 arrivalNode 值：
- [1797] a. 将 mp4bifs Route 元素的“arrivalNode”属性的值指定到整型量“nodeId”。
- [1798] b. 将数组 pNodeNumber 中的条目 nodeId 的值指定到整型量“nodeNumber”。
- [1799] c. 将数组 pNodeIdName 中的条目 nodeId 的值指定到字符串量“nodeIdName”。
- [1800] d. 如果字符串量“nodeIdName”的值为空值或没有内容（零长度），则将数量“nodeId”的值附加到前缀“nodeId:”，并将结果指定到数量“nodeIdName”。
- [1801] e. 将数量“nodeIdName”的值指定到 XMT-A “ROUTE”元素 360 或 390 的“toNode”属性。
- [1802] 4. 处理 arrivalField 值：
- [1803] a. 将 mp4bifs Route 元素的“arrivalField”属性的值指定到整型量“field”。
- [1804] b. 将具有索引“nodeNumber”的 BIFS 节点的 MPEG-4 规范中定义的字段名称的数组中的条目“field”的值指定到字符串量“fieldName”。
- [1805] c. 将数量“fieldName”的值指定到 XMT-A “ROUTE”元素 360 或 390 的“toField”属性。
- [1806] 将 mp4bifs Node 元素转换为 XMT-A Node 元素
- [1807] 每一个 mp4bifs Node 元素 3000 都由从属于特定 XMT-A 父元素的新的 XMT-A Node 元素 400 表示。此 XMT-A 父元素的身份是由 mp4bifs Node 3000 元素所从属的 mp4bifs 元素确定的：
- [1808] a. 如果 mp4bifs Node 元素 2920 从属于 InsertNodebifsCommand 元素 2910 或 InsertIndexedValue bifsCommand 元素 2910，那么，XMT-A 父元素由对应的 XMT-A “Insert”元素 300 进行定义。
- [1809] b. 如果 mp4bifs Node 元素 2920 从属于 ReplaceNodebifsCommand 元素 2910、ReplaceField bifsCommand 元素 2910 或 ReplaceIndexedValue bifsCommand 元素 2910，那么，XMT-A 父元素由对应的 XMT-A “Replace”元素 320 进行定义。
- [1810] c. 如果 mp4bifs Node 元素 2940 从属于 ReplaceScenebifsCommand 元素 2930，那么，XMT-A 由对应的 XMT-A “Scene”元素 370 进行定义。
- [1811] d. 如果 mp4bifs Node 元素 3030 从属于另一个 mp4bifsNode 元素 3000 或 InsertIndexedValue bifsCommand 元素 2910，那么，XMT-A 父元素由对应的 XMT-A “nodeField”元素 410 进行定义。
- [1812] 如果 mp4bifs Node 元素的元素名称为“ReusedNode”3080，则执行过程“为 mp4bifs ReusedNode 元素创建 XMT-A Node 元素”。否则，执行下列步骤。
- [1813] 1. 确定整型量“nodeNumber”的值。by 此值是由 MPEG-4 规范中定义的节点名称的数组中的条目号码（所指出的条目的值匹配当前 mp4bifs Node 元素 3000 的节点名称）确定。
- [1814] 2. 用相同元素名称 (NodeName) 作为 mp4bifs Node 元素 3000 创建新的 XMT-A Node 元素 400，并将它附加到 XMT-A 父元素。此新的 XMT-A Node 元素被定义为“当前 XMT-A Node 元素”。
- [1815] 3. 如果需要，确定“DEF”属性的值：
- [1816] a. 将 mp4bifs Node 元素的“name”属性 3016 的值指定到字符串量“nodeIdName”。

[1817] b. 如果数量“nodeName”的值不是空值并且由此数量表示的字符串至少具有一个字符,则将数量“nodeName”的值指定到新的 XMT-A Node 元素 400 的“DEF”属性。

[1818] c. 如果数量“nodeName”的值是空值或由此数量表示的字符串小于一个字符,则将 mp4bifs Node 元素 3000 的“NodeId”属性 3010 的值指定到字符串量“nodeId”。

[1819] d. 如果数量“nodeName”的值不是空值并且由此数量表示的字符串至少具有一个字符,则将数量“nodeName”的值附加到新的 XMT-A Node 元素 400 的“DEF”属性。

[1820] 4. 使用标准 xml 装置选择为当前 mp4bifs Node 元素 3000 指定的每一个属性字段属性值 3020。

[1821] 5. 将当前属性值的名称的值指定到字符串量“attributeName”。

[1822] 6. 对于 mp4bifs Node 元素 3000 的每一个属性字段属性 3020,执行下列步骤。

[1823] 5. 确定字段数据类型:

[1824] a. 确定整型量“field”的值。此值等于匹配“attributeName”的值字段名称的表的成员的条目编号的值。

[1825] 字段名称的表是由 MPEG-4 规范为具有与当前 mp4bifsNode 元素关联的节点名称和节点号码的 BIFS 节点定义的。

[1826] b. 将 fieldDataType 值的表中的条目“field”的值指定到整型量“fieldDataType”。fieldDataType 值的表是由 MPEG-4 规范为具有与当前 mp4bifs Node 元素 3000 关联的节点名称和节点号码的 BIFS 节点定义的。

[1827] 6. 检查 URL 字段数据类型。如果字段数据类型等于 MPEG-4 规范定义的表示“URL”字段数据类型的值,则执行下列操作:

[1828] a. 将 mp4bifs Node 元素的“attributeName”属性的值指定到字符串量“urlString”。

[1829] 将由数量“urlString”表示的字符串的开头五个字符替换为前缀:“odid://ODID:”

[1830] c. 将所产生的字符串值指定到当前 XMT-A Node 元素 400 的“attributeName”属性。

[1831] 7. 检查 Node 字段数据类型。如果字段数据类型等于 MPEG-4 规范定义的表示“Node”字段数据类型的值,则执行下列操作:

[1832] a. 创建新的具有元素名称“attributeName”的 XMT-A 元素,并将它附加到 XMT-A Node 元素。此元素被定义为“当前 XMT-A nodeField 元素”。

[1833] b. 使用标准 xml 装置选择当前 mp4bifs Node 元素所拥有的每一个从属元素。这些从属元素中的每一个从属元素预计为另一个 mp4bifs Node 元素。这些从属元素被指定为“子”元素,当前 mp4bifsNode 元素被指定为“父”元素。

[1834] c. 使用当前 XMT-A nodeField 元素作为“父”元素,对每一个子元素执行过程“将 mp4bifs Node 元素转换为 XMT-A 元素”。

[1835] 8. 检查 Command Buffer 字段数据类型。如果字段数据类型等于 MPEG-4 规范定义的表示“Command Buffer”字段数据类型的值,则执行下列操作:

[1836] a. 创建新的具有元素名称“attributeName”的 XMT-A 元素,并将它附加到 XMT-A Node 元素。此元素被定义为“当前 XMT-A nodeField 元素”400。

[1837] b. 使用当前 XMT-A nodeField 元素作为“父”元素, 执行过程“将 mp4bifs commandFrame 元素转换为 XMT-A 元素”。

[1838] 9. 处理其他属性。如果字段数据类型不是在前面的步骤中标识的其中一个值, 则执行下列操作:

[1839] a. 将 mp4bifs Node 元素的“attributeName”属性的值指定到字符串量“attributeValue”。

[1840] b. 将数量“attributeValue”的值指定到 XMT-A Node 元素 400 的“attributeName”属性。

[1841] 为 mp4bifs ReusedNode 元素创建 XMT-A Node 元素

[1842] 在 mp4bifs ReusedNode 元素 3080 的情况下, nodeRef 值 3090 是必需的属性。对于每一个 mp4bifs ReusedNode 元素 3080, 执行下列步骤:

[1843] 1. 将 nodeRef 属性 3090 的值指定到整型量“nodeId”。

[1844] 2. 将数组 pNodeNumbers 中的条目“nodeId”的值指定到整型量“nodeNumber”。

[1845] 3. 将 MPEG-4 规范中定义的节点名称的表中的条目“nodeNumber”的值指定到字符串量“nodeName”。

[1846] 4. 创建类型“nodeName”的新的 XMT-A node 元素 400, 并将它附加到当前 XMT-A 父元素。

[1847] 5. 将数量 nodeId 的值指定到新的 XMT-A Node 元素 400 的“USE”属性。

[1848] 前面的对本发明的简要说明只是为了说明和描述。它不可以是详尽的描述, 也不将本发明限制为所说明的准确形式, 根据上述原理, 其他修改和变化也是可以的。例如, 对 XMT-A 文件的精确定义可以随着时间而变化或发展。同样, 对 MPEG-4 中间格式的精确定义也可以随着时间变化或发展。这里所描述的本发明不仅限于上文所引用的文档中指定的特定定义。如此, 此发明的原理还可以适用于其他不相关的数据结构。在 MPEG-4 系统规范中还定义了 SFNode 数据结构的其它扩展形式。如下列实施例中所描述的扩展本发明以涵盖这样的情况装置, 对于精通本技术的那些人是显而易见的。如此, 选择和描述了所说明的实施例, 以便最好地说明本发明的原理, 以及其实际应用, 从而使精通本技术的其他人最好地在各种实施例和各种变体中利用本发明, 只要它们适合于特定用途。

[1849] 此外, 所附的权利要求应该被解释为涵盖本发明的其他替代实施例, 除非被现有技术限制。

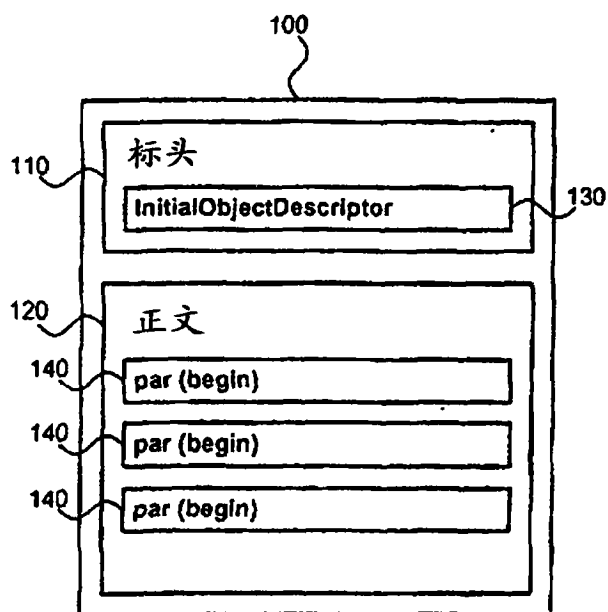


图 1A

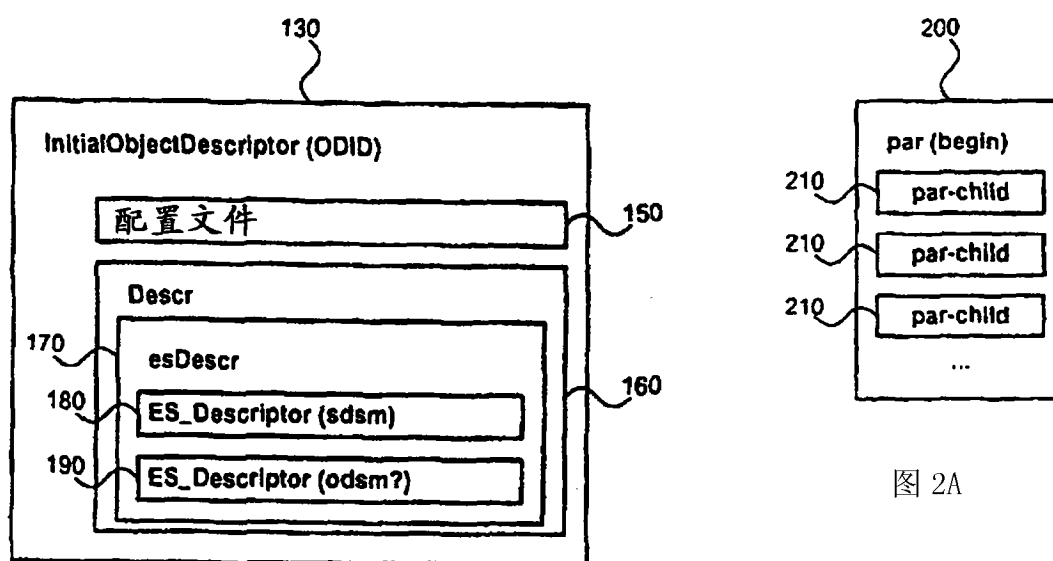


图 2A

图 1B

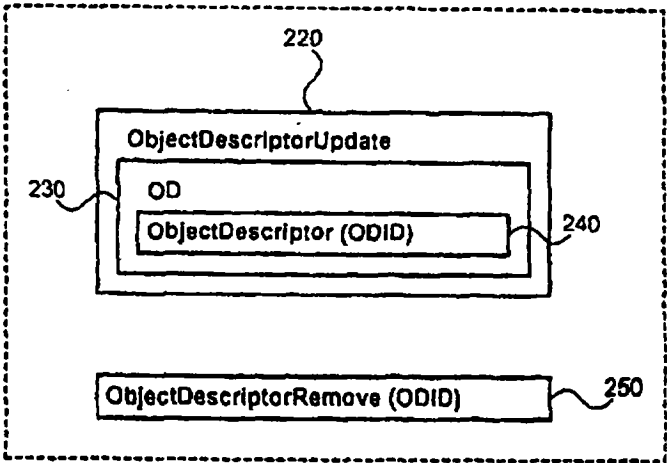


图 2B

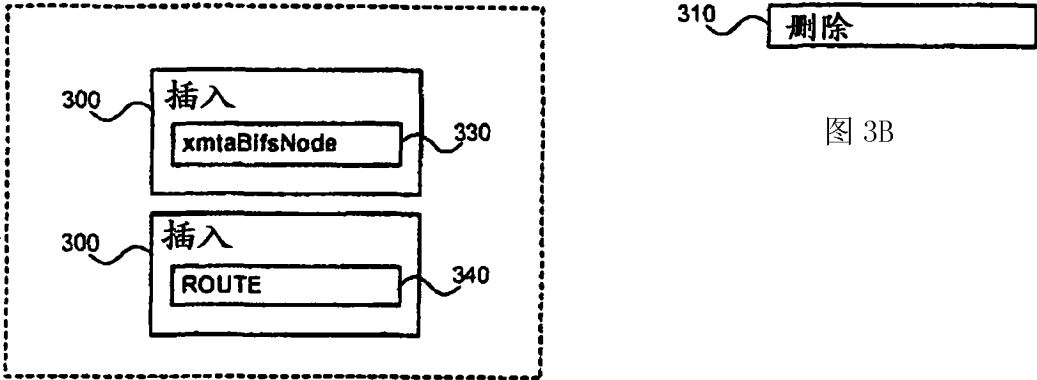


图 3A

图 3B

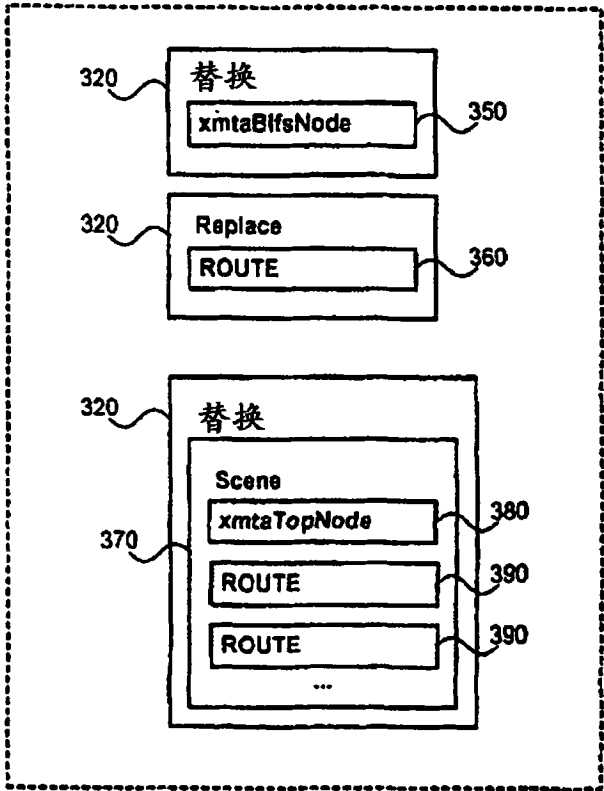


图 3C

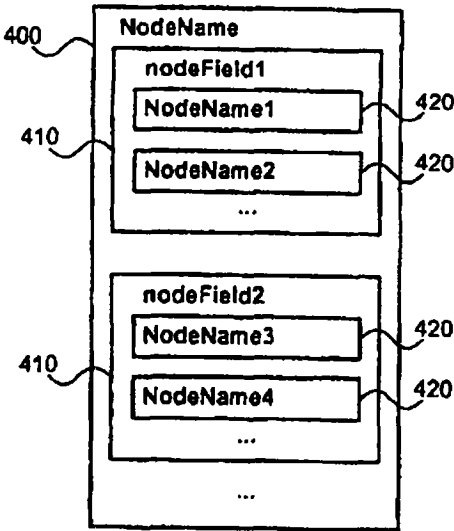


图 4

500 ~~~~~ <nodeName
510 ~~~~~ DEF = "nodeIDstring"
520 ~~~~~ field0 = "value0"
520 ~~~~~ field2 = "value2"
520 ~~~~~ field3 = "value3"
520 ~~~~~ field5 = "value5" ... >

530 ~~~~~ <field1>
550 ~~~~~ <nodeName1 ... />
550 ~~~~~ <nodeName2 ... />
540 ~~~~~ </field1>

530 ~~~~~ <field4>
550 ~~~~~ <nodeName3 ... />
540 ~~~~~ </field4>

560 ~~~~~ </nodeName>

图 5A

570 ~~~~~
580 ~~~~~ <nodeName
590 ~~~~~ USE = "nodeIDstring" />

图 5B

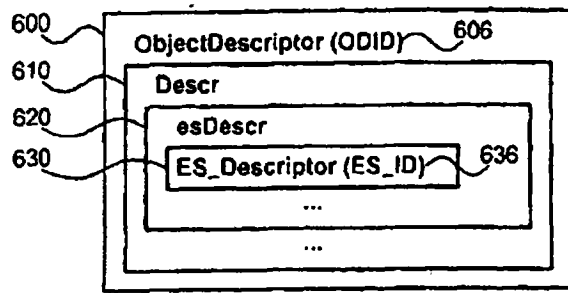


图 6A

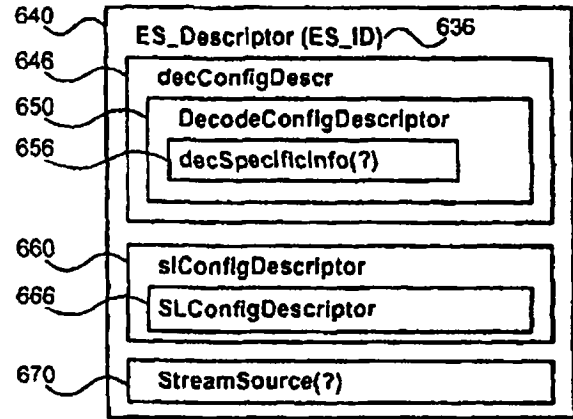


图 6B

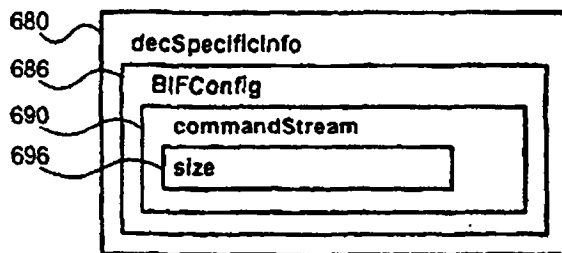


图 6C

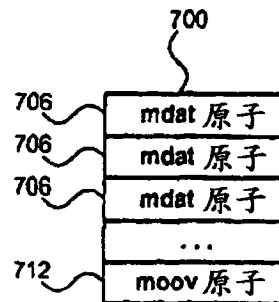


图 7A

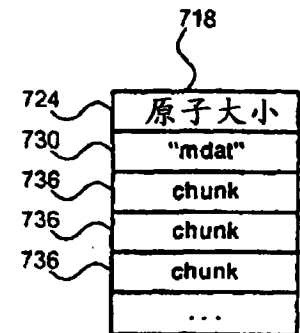


图 7B

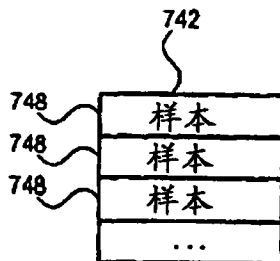


图 7C

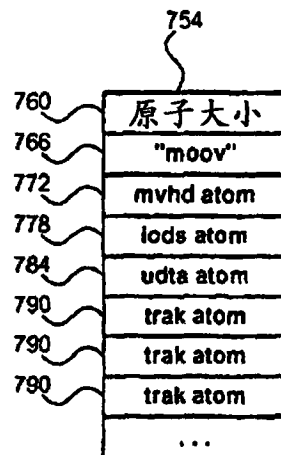


图 7D

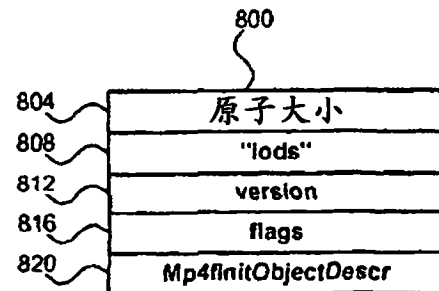


图 8A

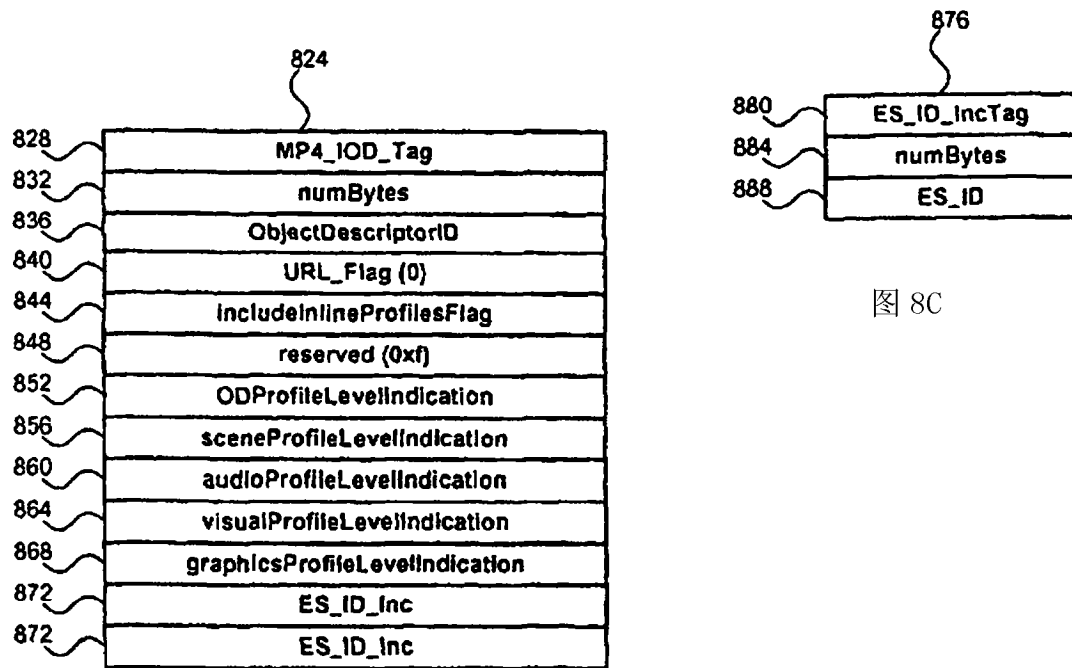


图 8C

图 8B

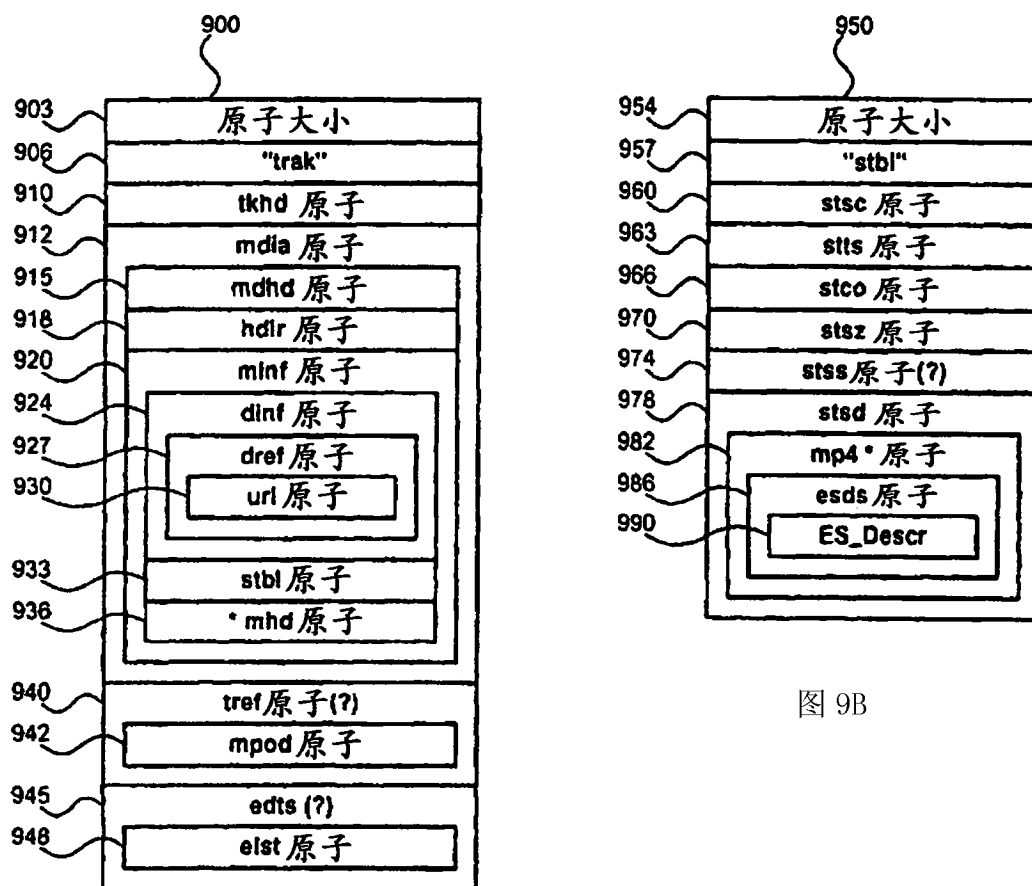


图 9B

图 9A

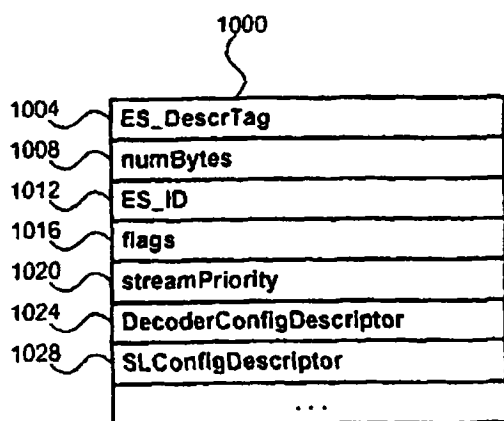


图 10A

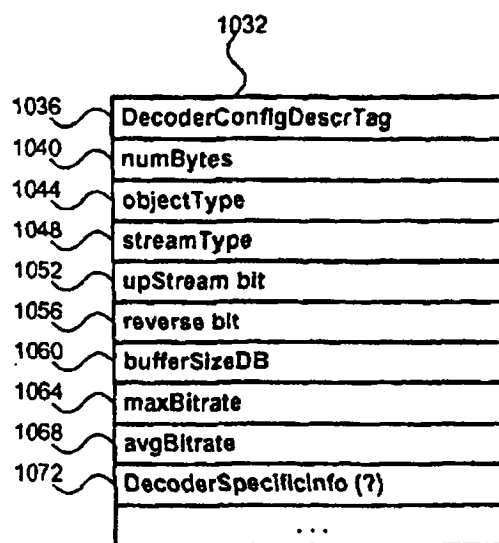


图 10B

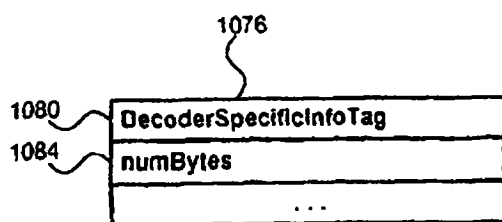


图 10C

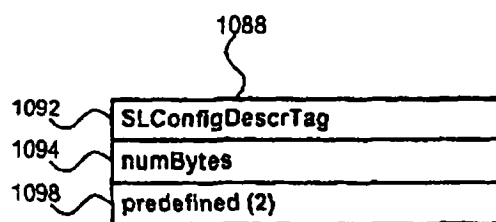


图 10D

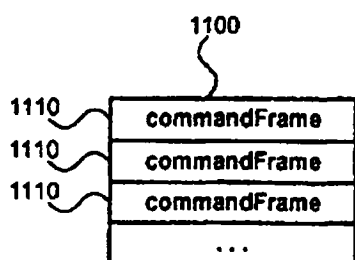


图 11A

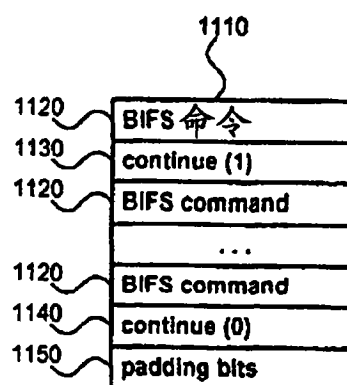


图 11B

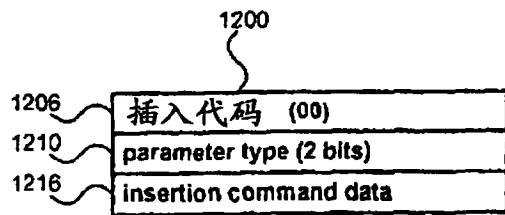


图 12A

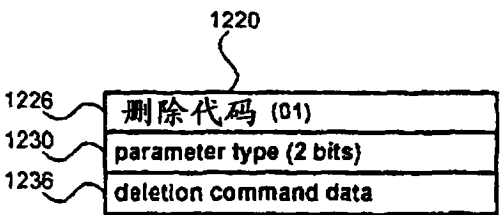


图 12B

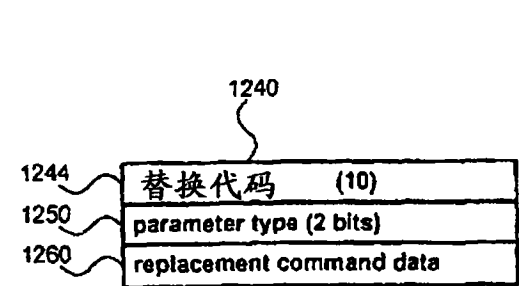


图 12C

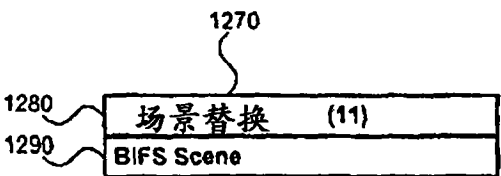


图 12D

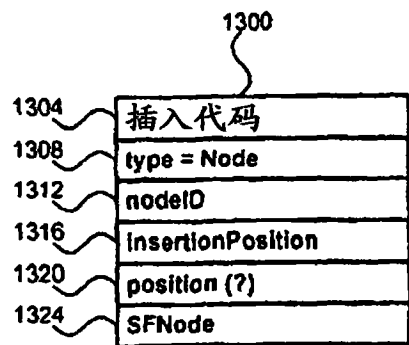


图 13A

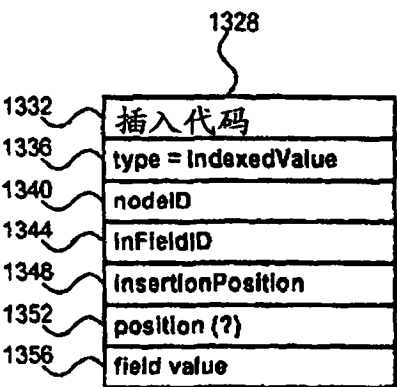


图 13B

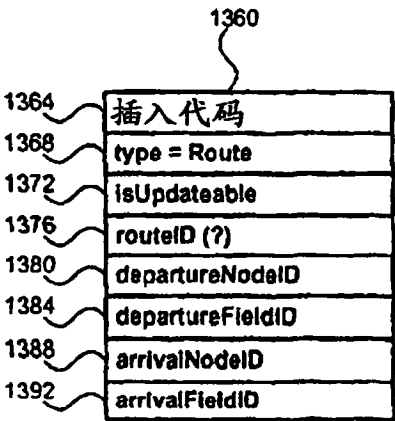


图 13C

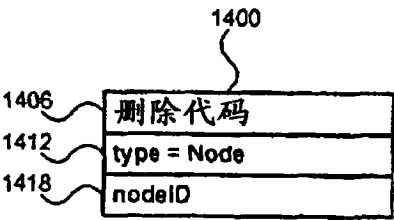


图 14A

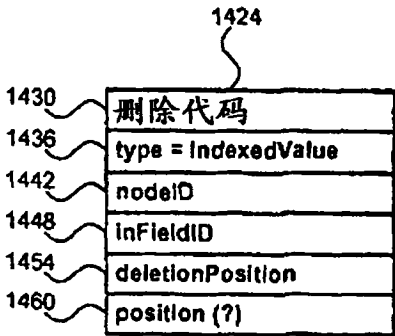


图 14B

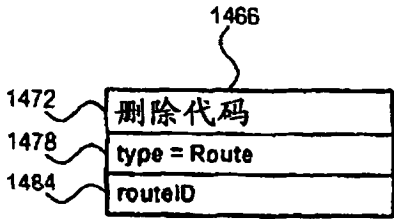


图 14C

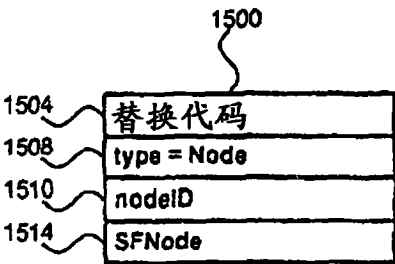


图 15A

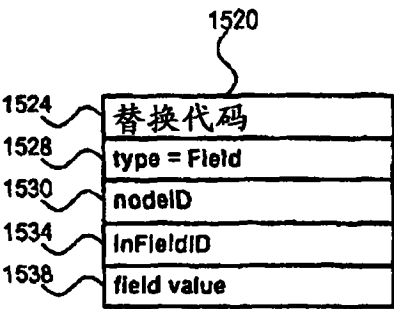


图 15B

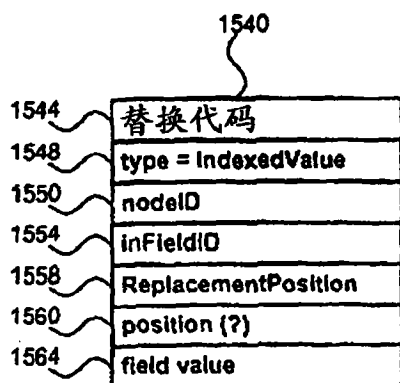


图 15C

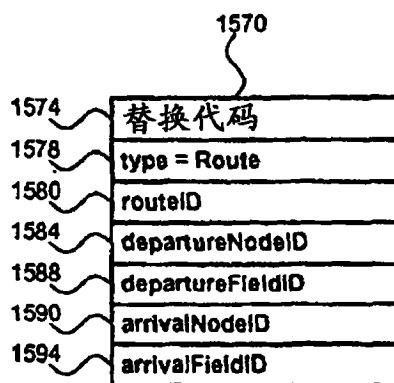


图 15D

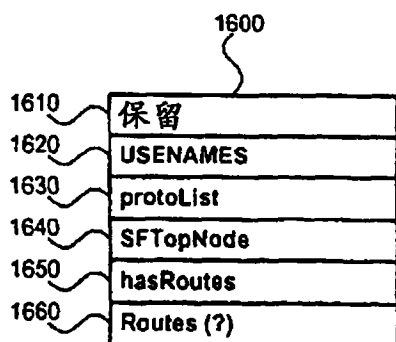


图 16

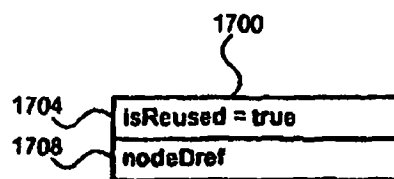


图 17A

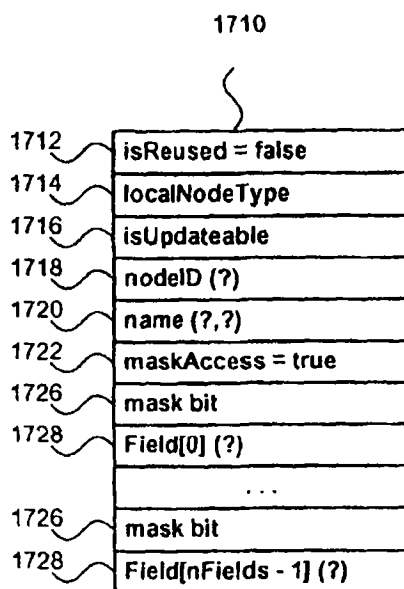


图 17B

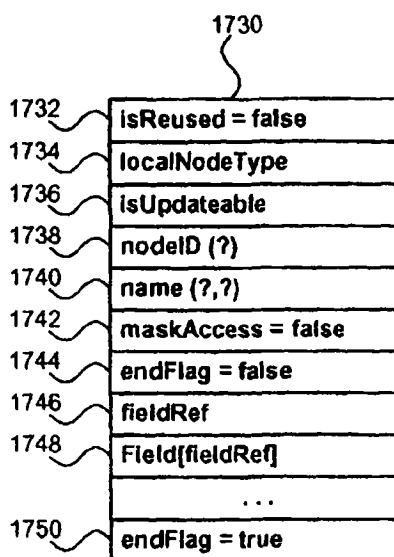


图 17C

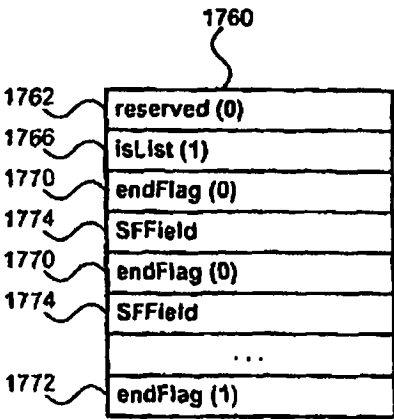


图 17D

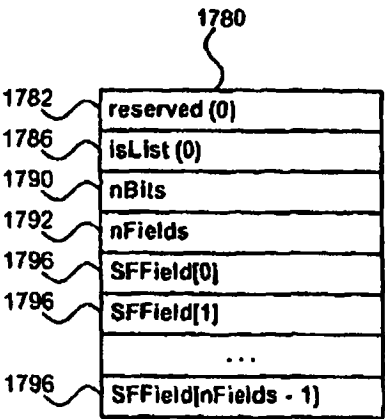


图 17E

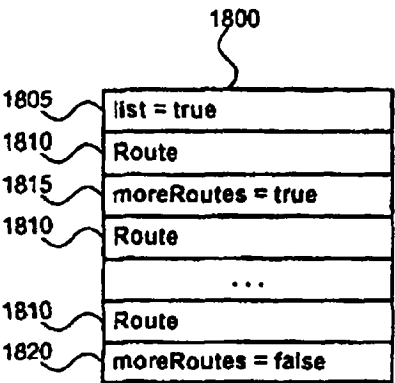


图 18A

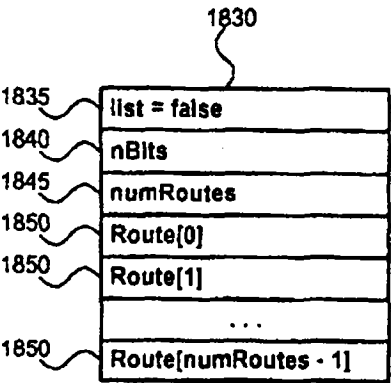


图 18B

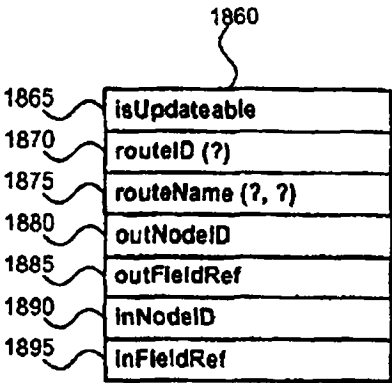


图 18C

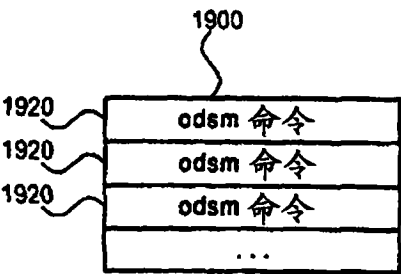


图 19A

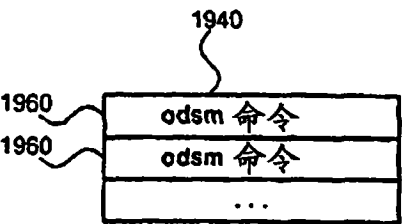


图 19B

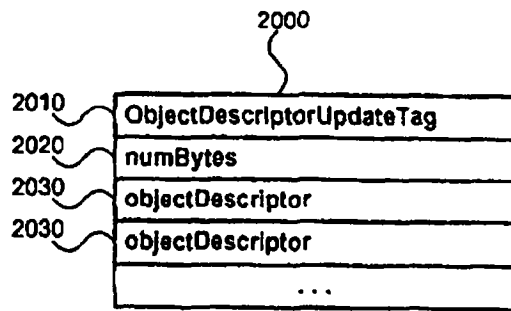


图 20A

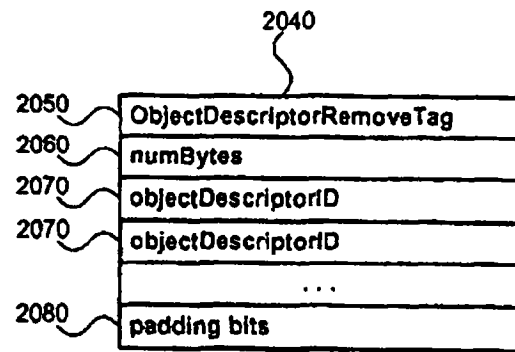


图 20B

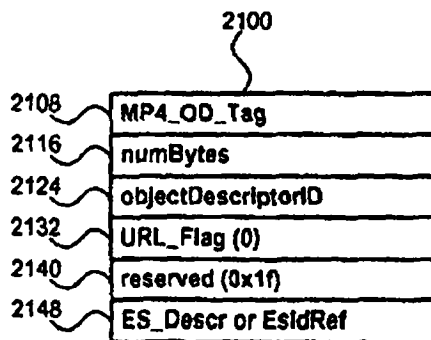


图 21A

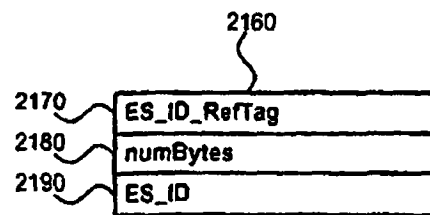


图 21B

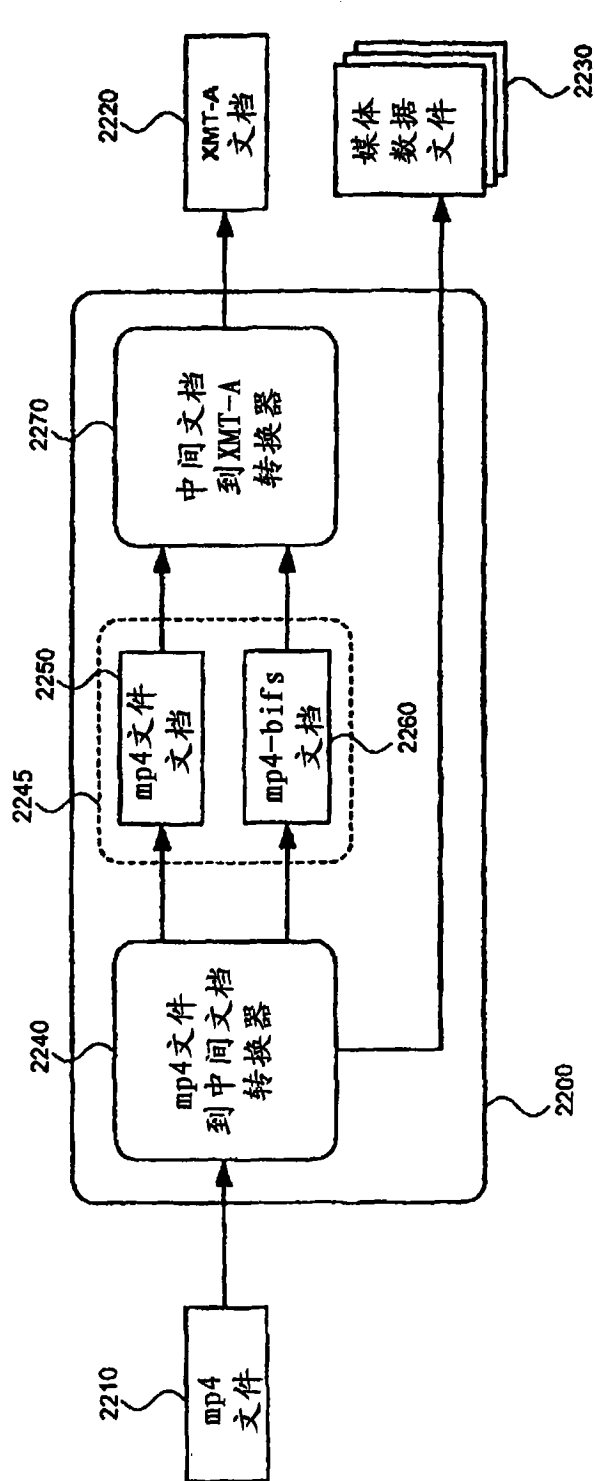


图 22

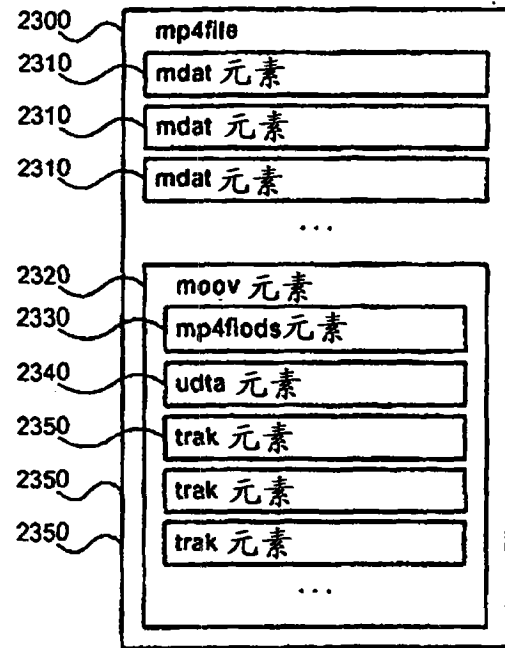


图 23A

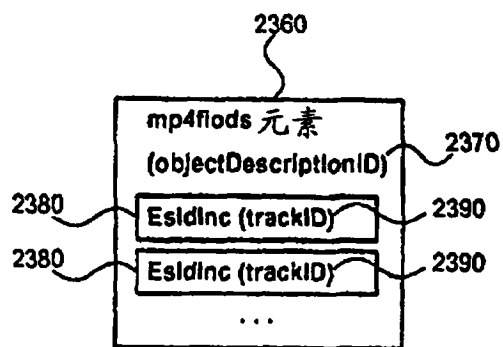


图 23B

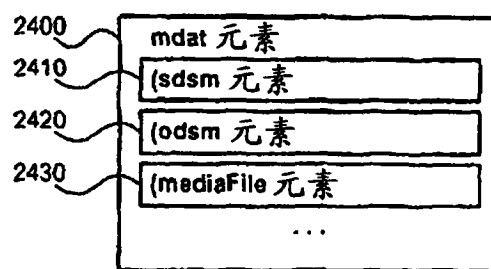


图 24A

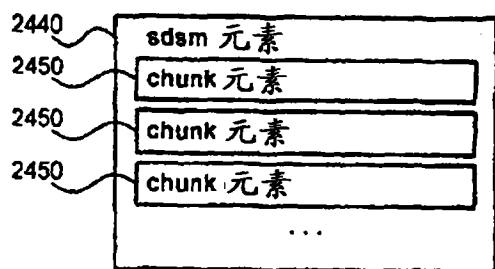


图 24B

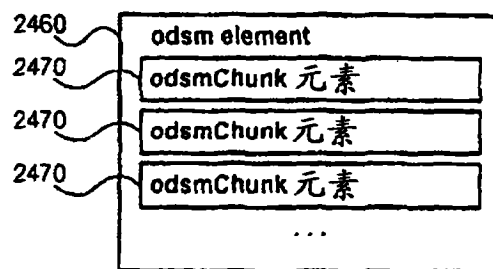


图 24C

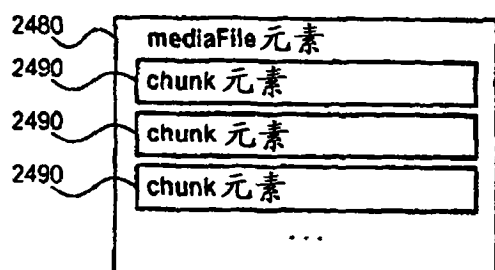


图 24D

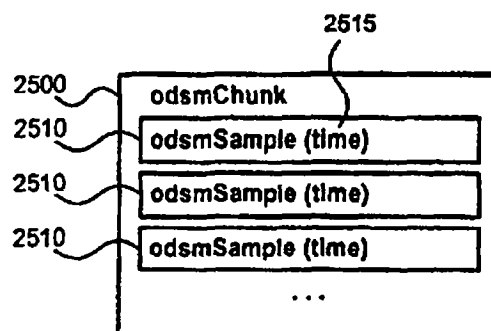


图 25A

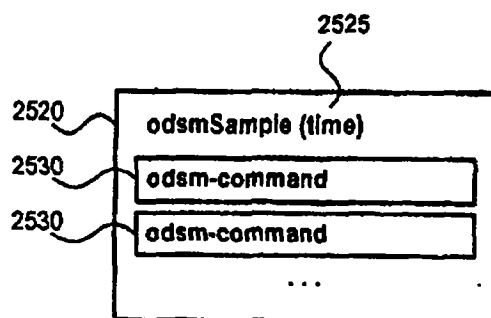


图 25B

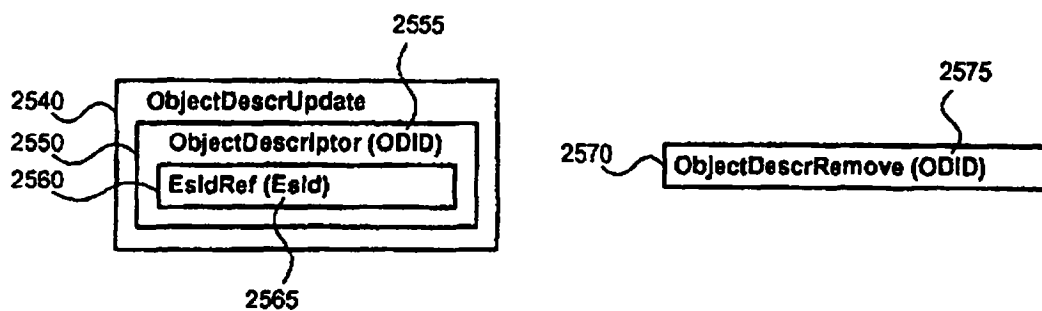


图 25C

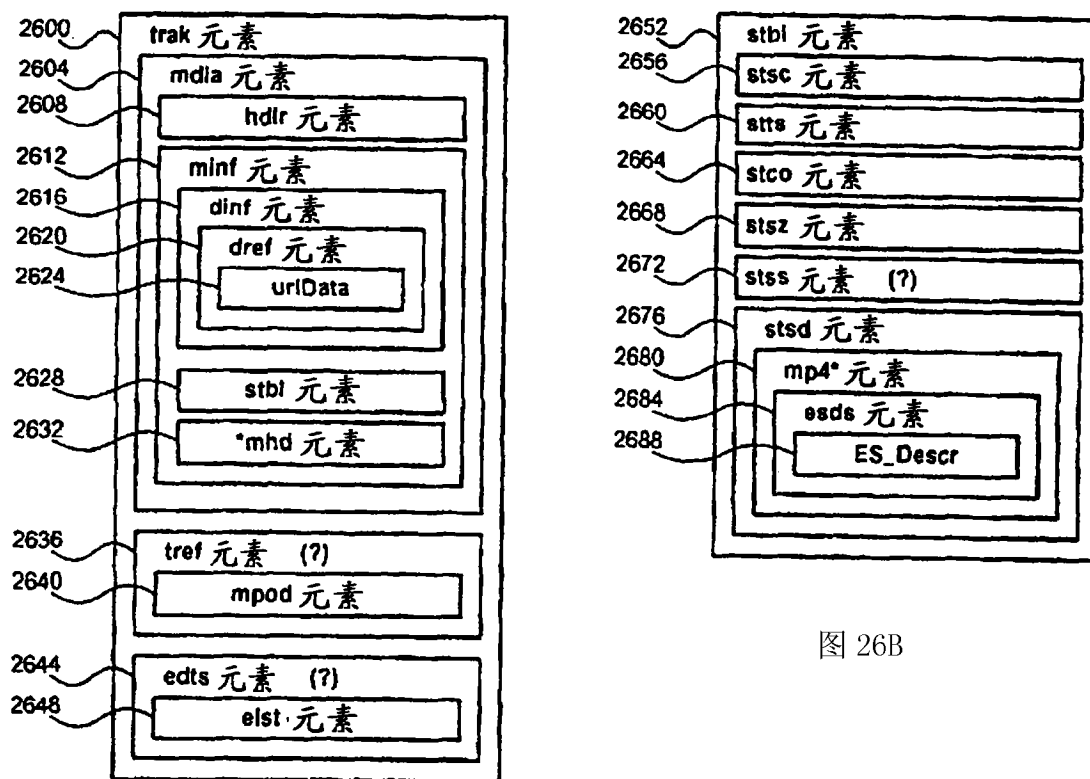


图 26B

图 26A

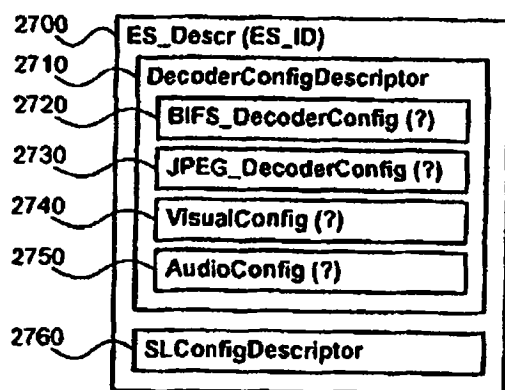


图 27

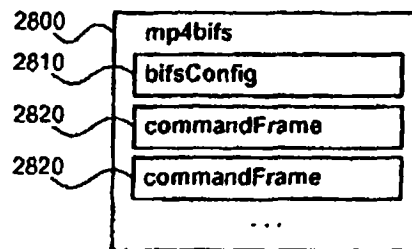


图 28A

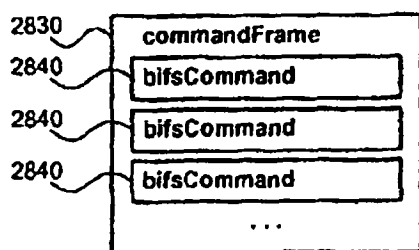


图 28B

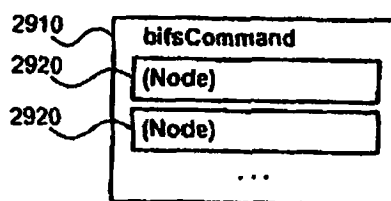


图 29A

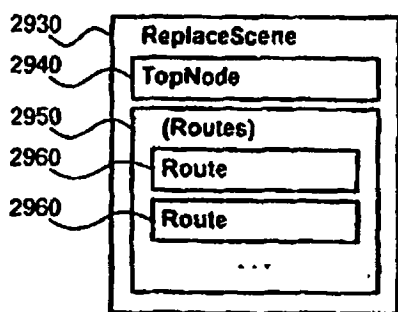


图 29B

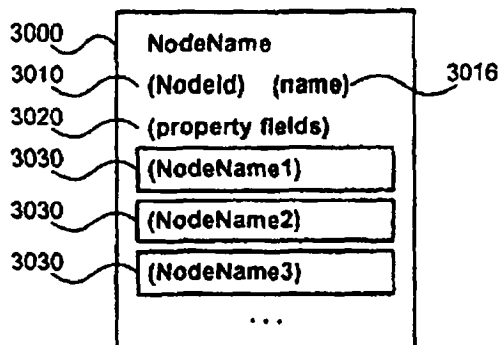


图 30A

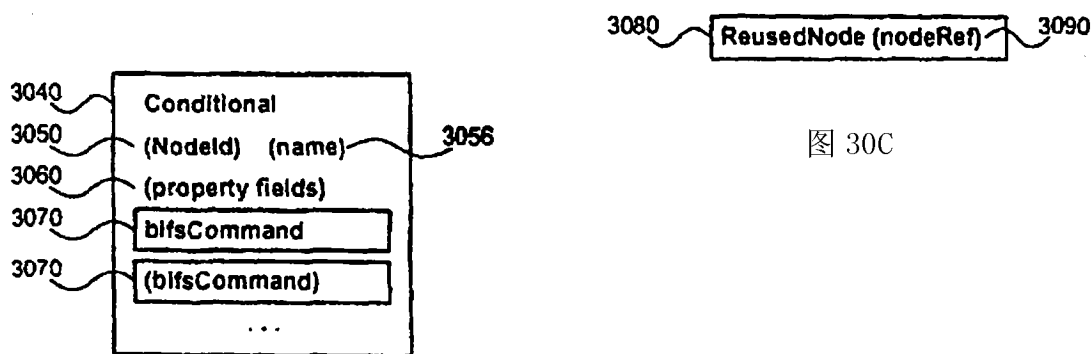


图 30C

图 30B

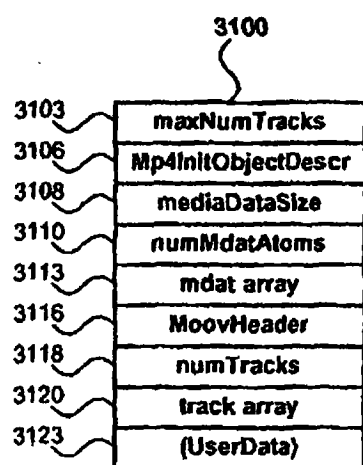


图 31A

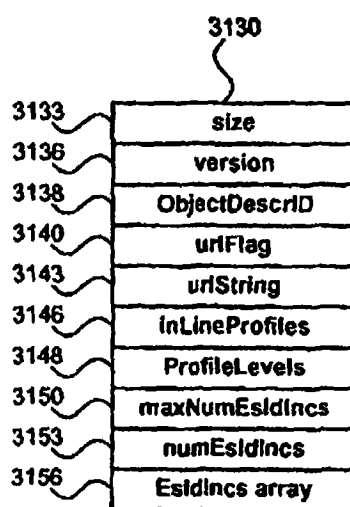


图 31B

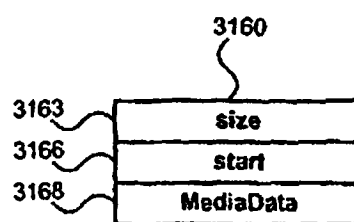


图 31C

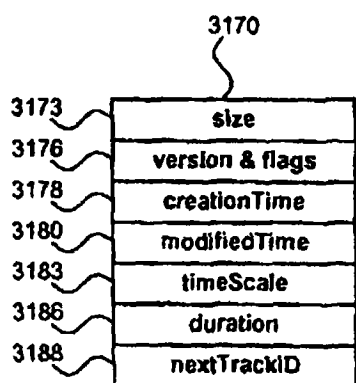


图 31D

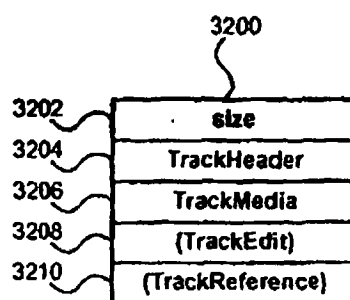


图 32A

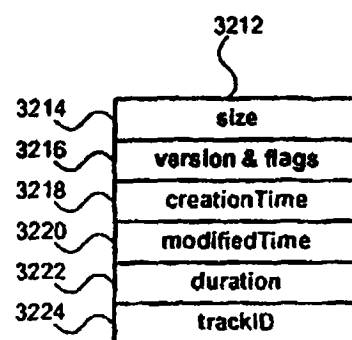


图 32B

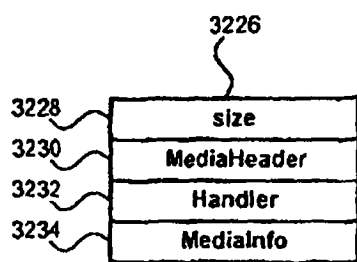


图 32C

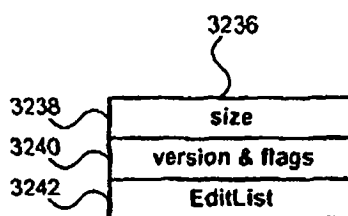


图 32D

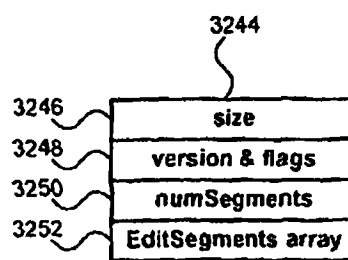


图 32E

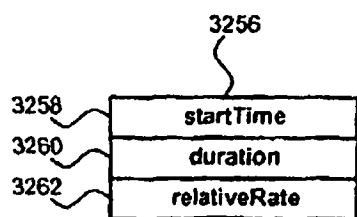


图 32F

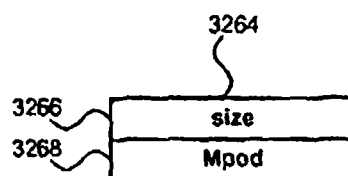


图 32G

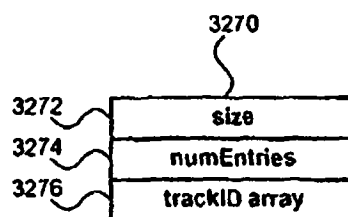


图 32H

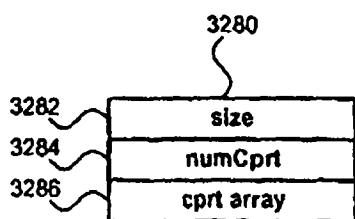


图 32I

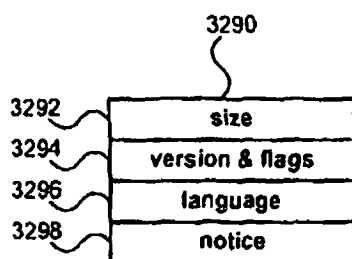


图 32J

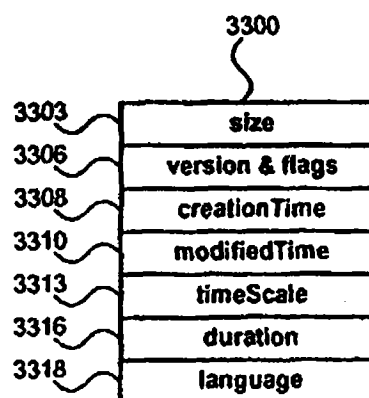


图 33A

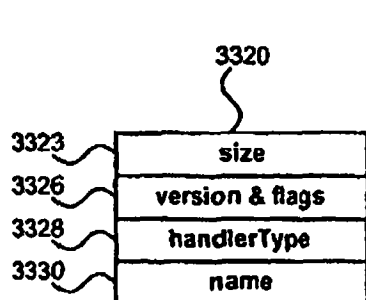


图 33B

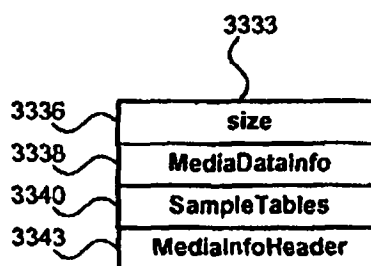


图 33C

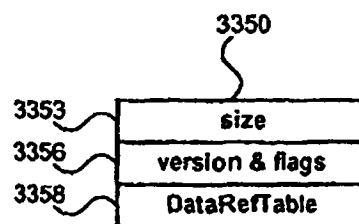


图 33D

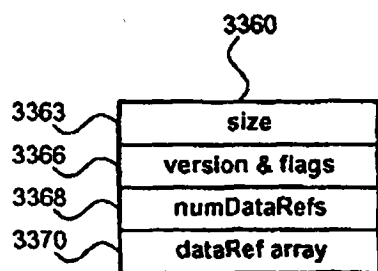


图 33E

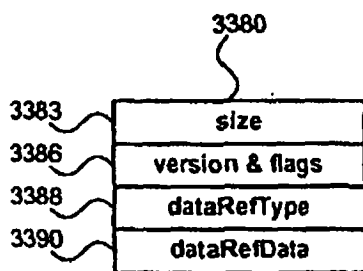


图 33F

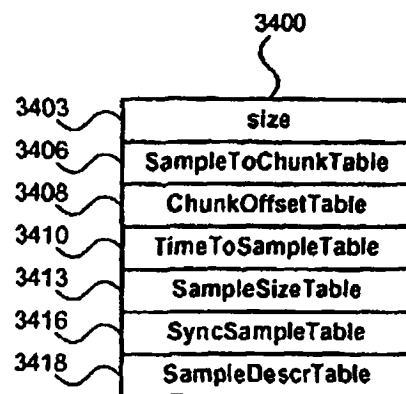


图 34A

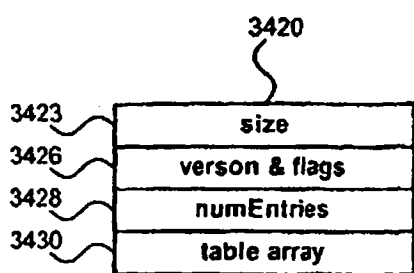


图 34B

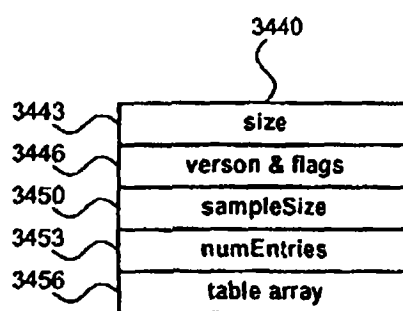


图 34C

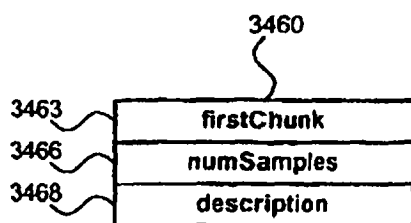


图 34D

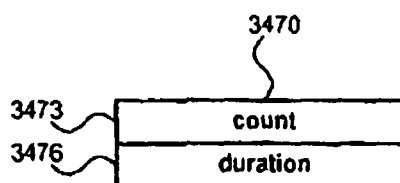


图 34E

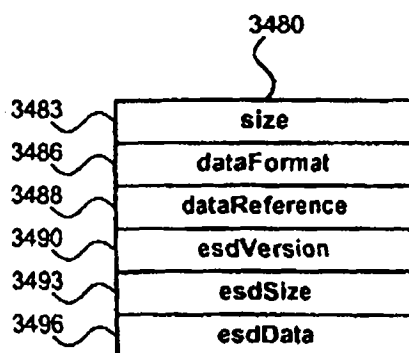


图 34F

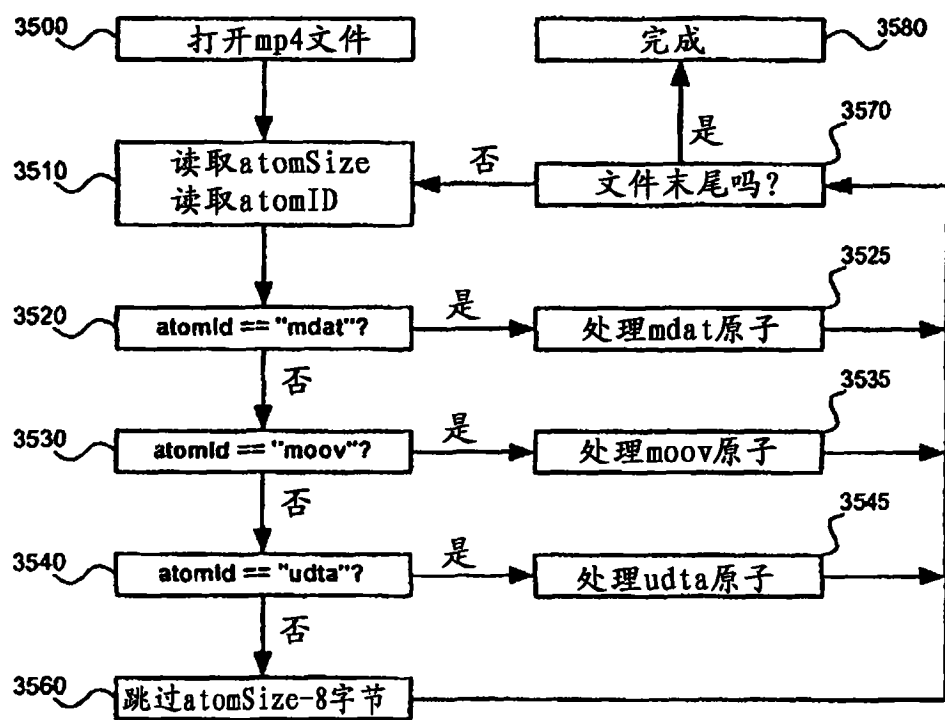


图 35

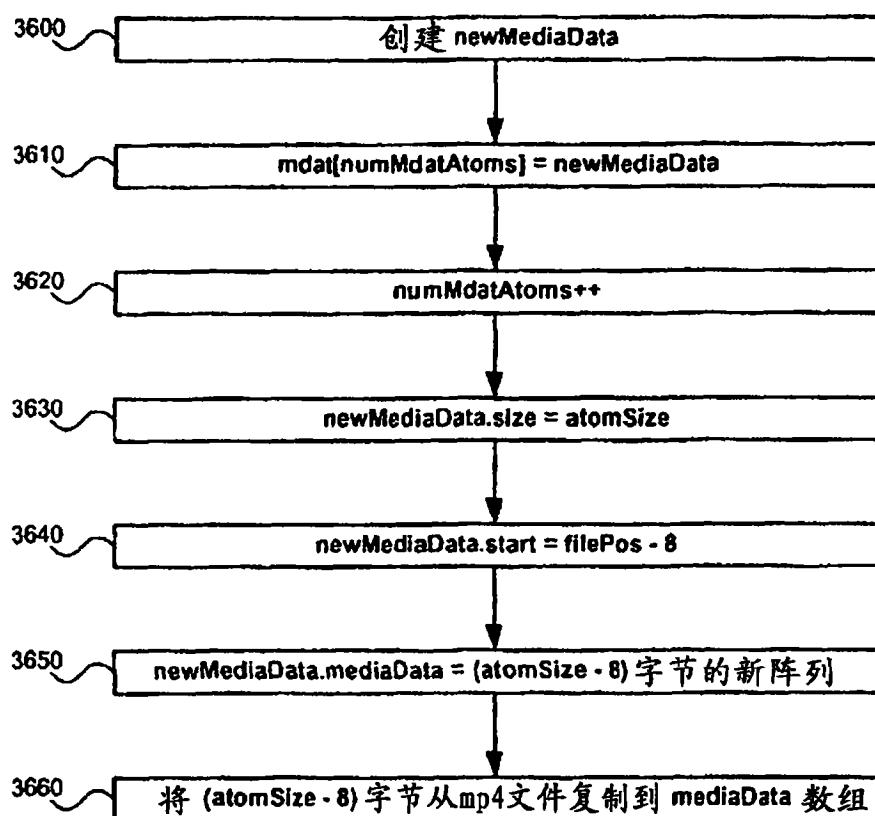


图 36

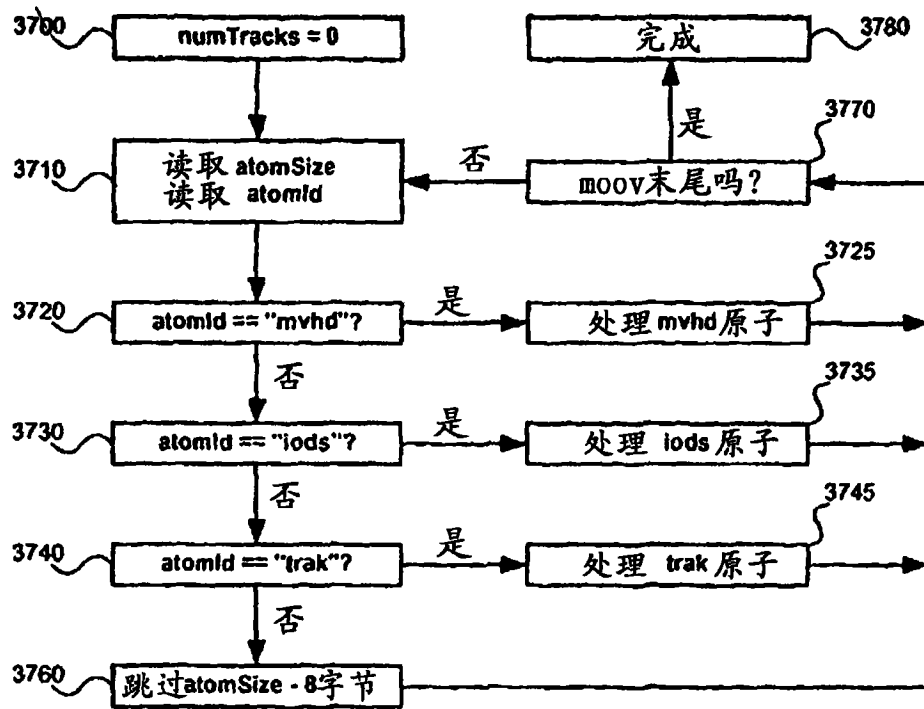


图 37

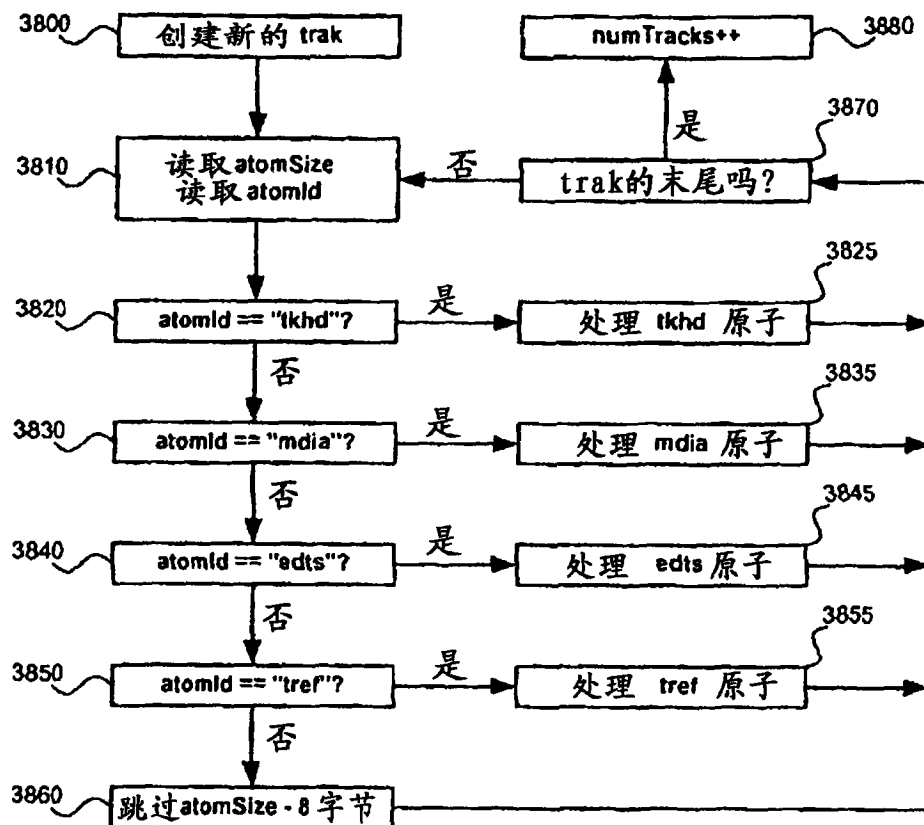


图 38

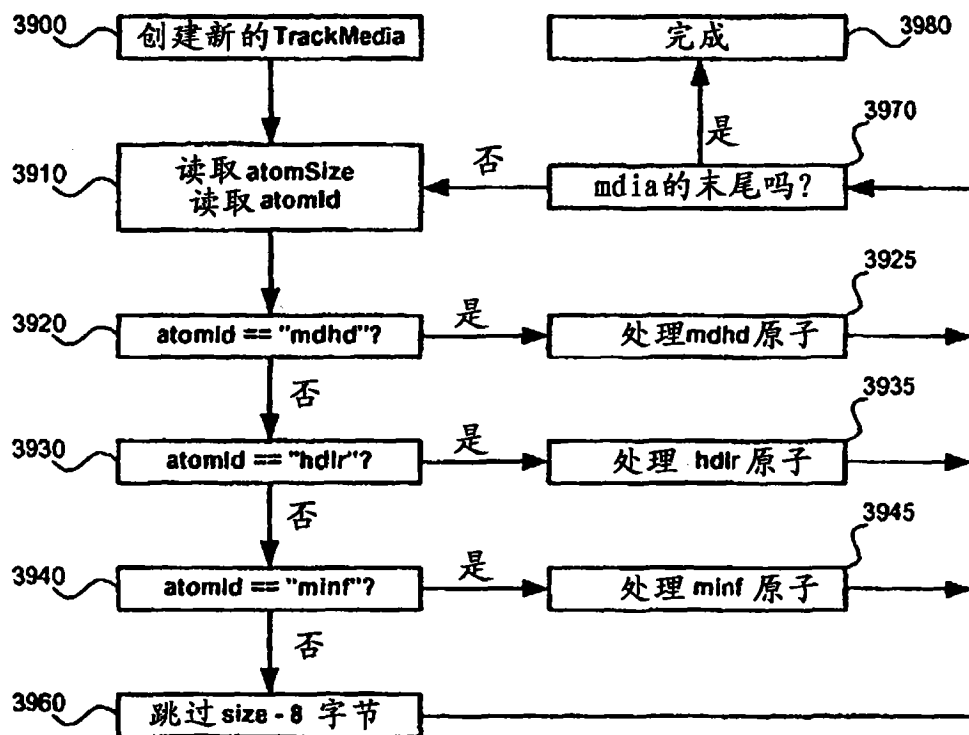


图 39

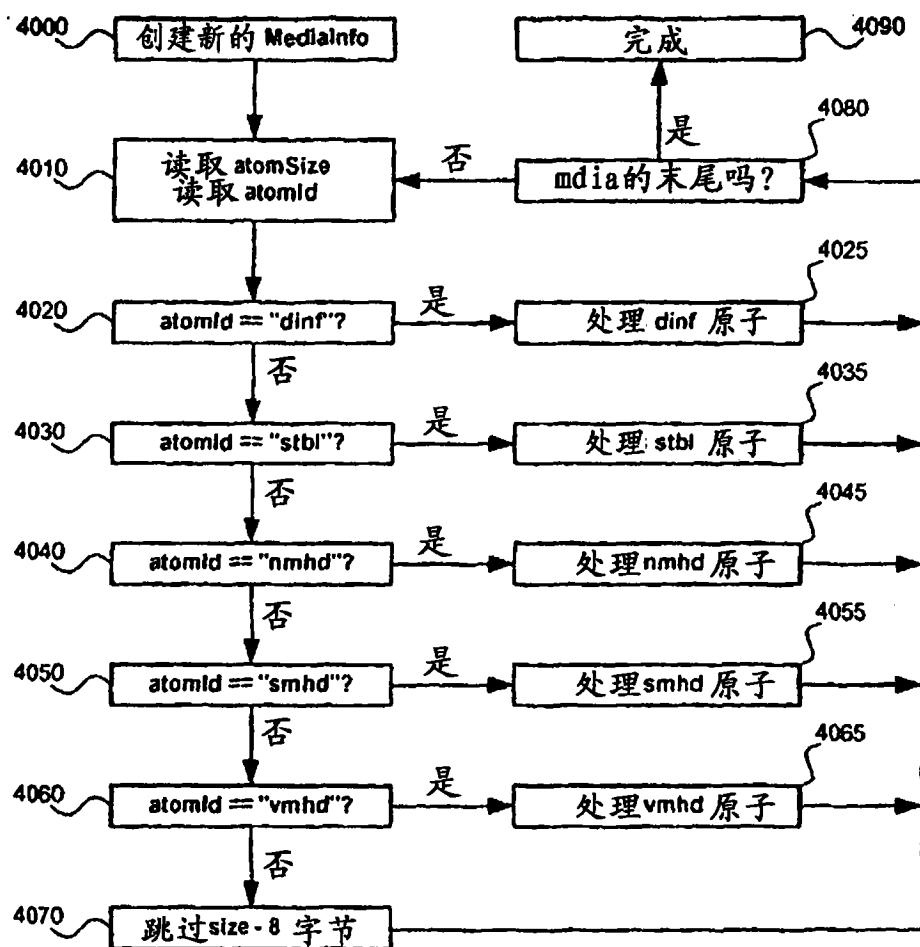


图 40

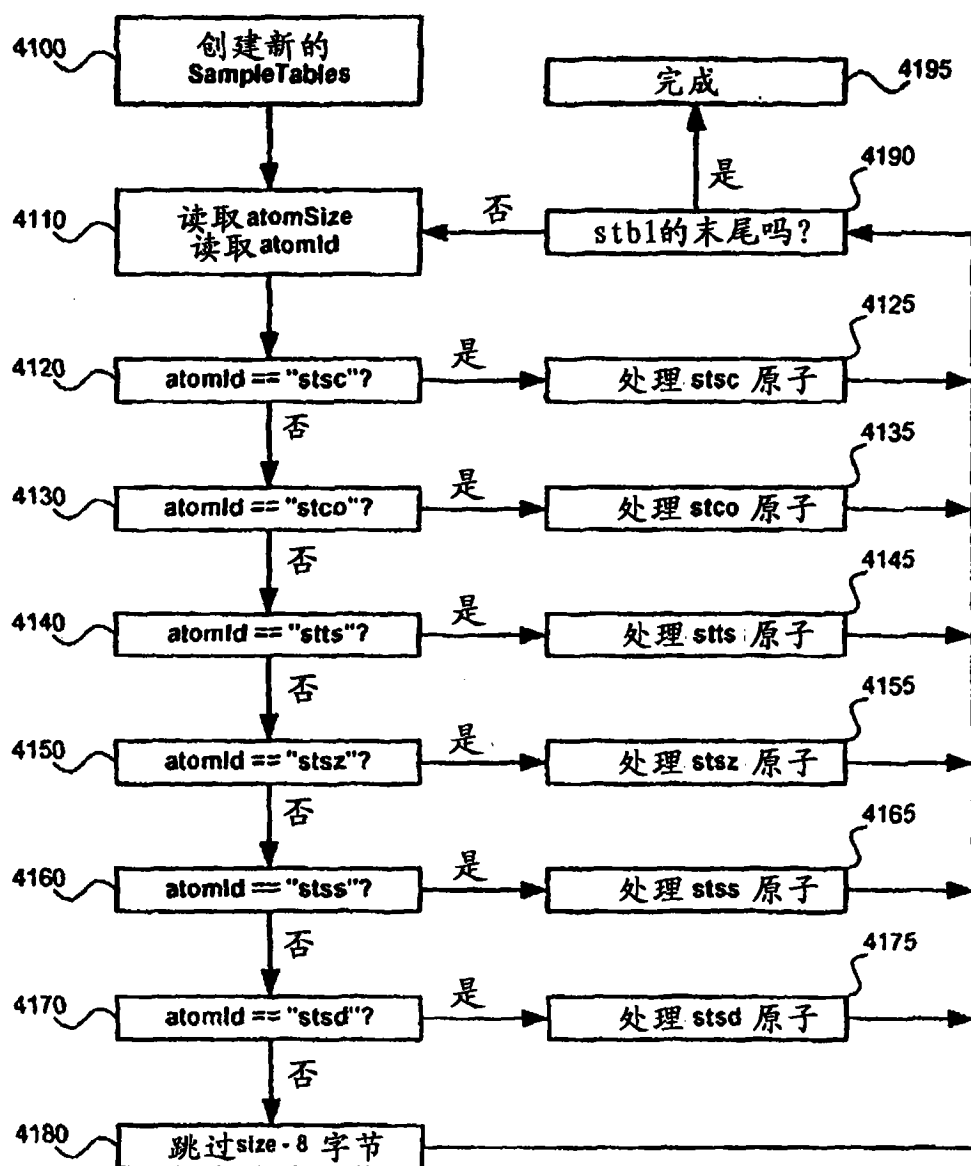


图 41

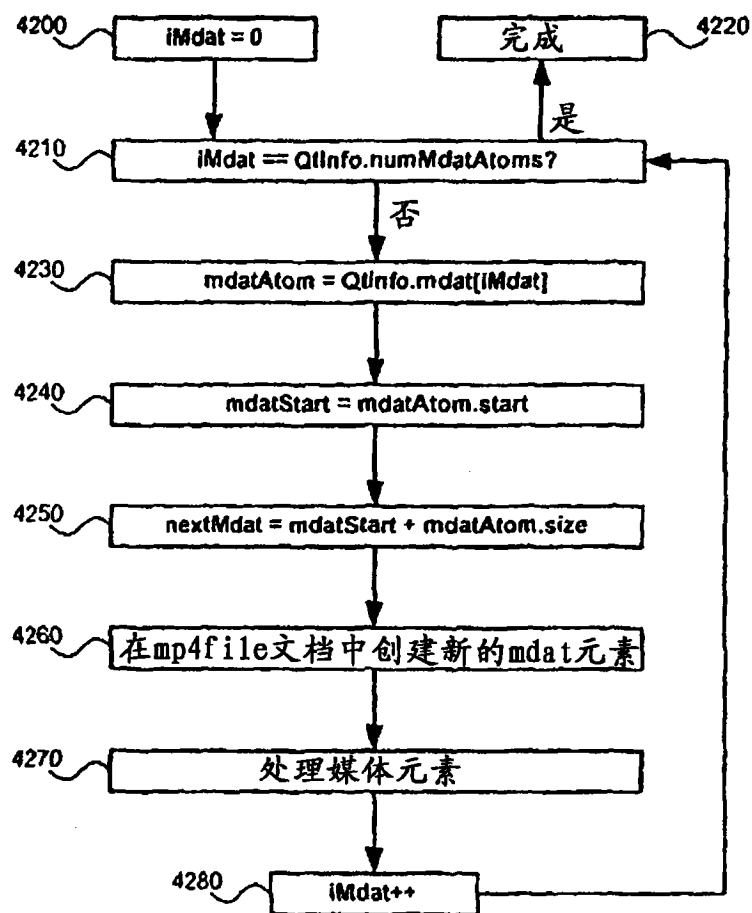


图 42

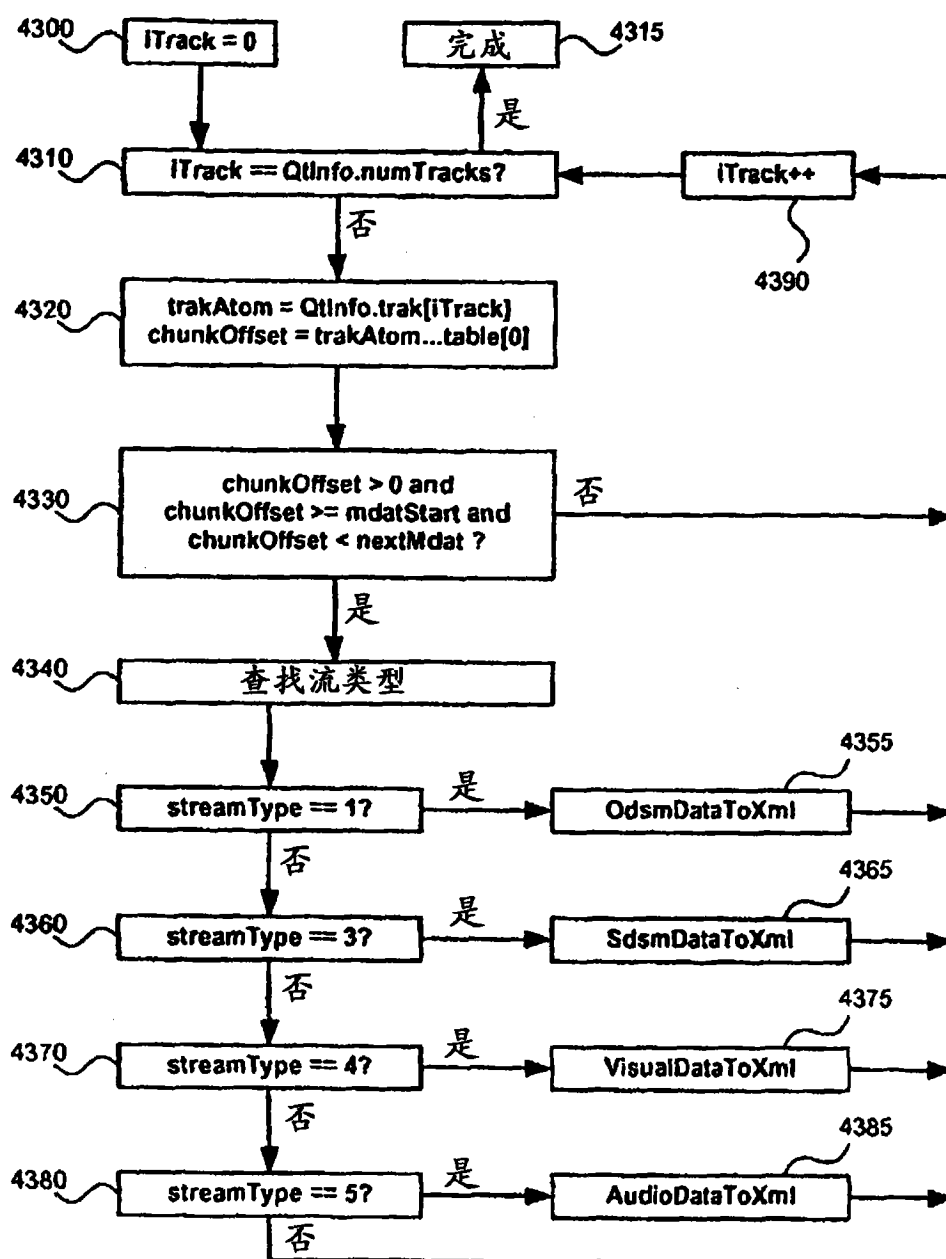


图 43

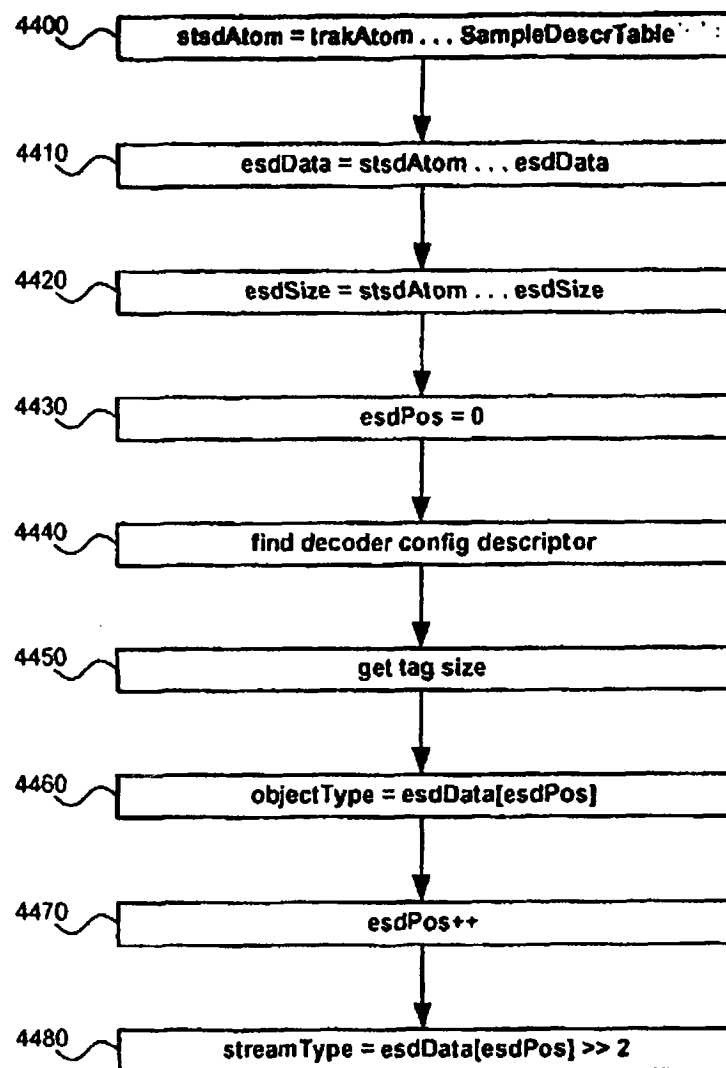


图 44

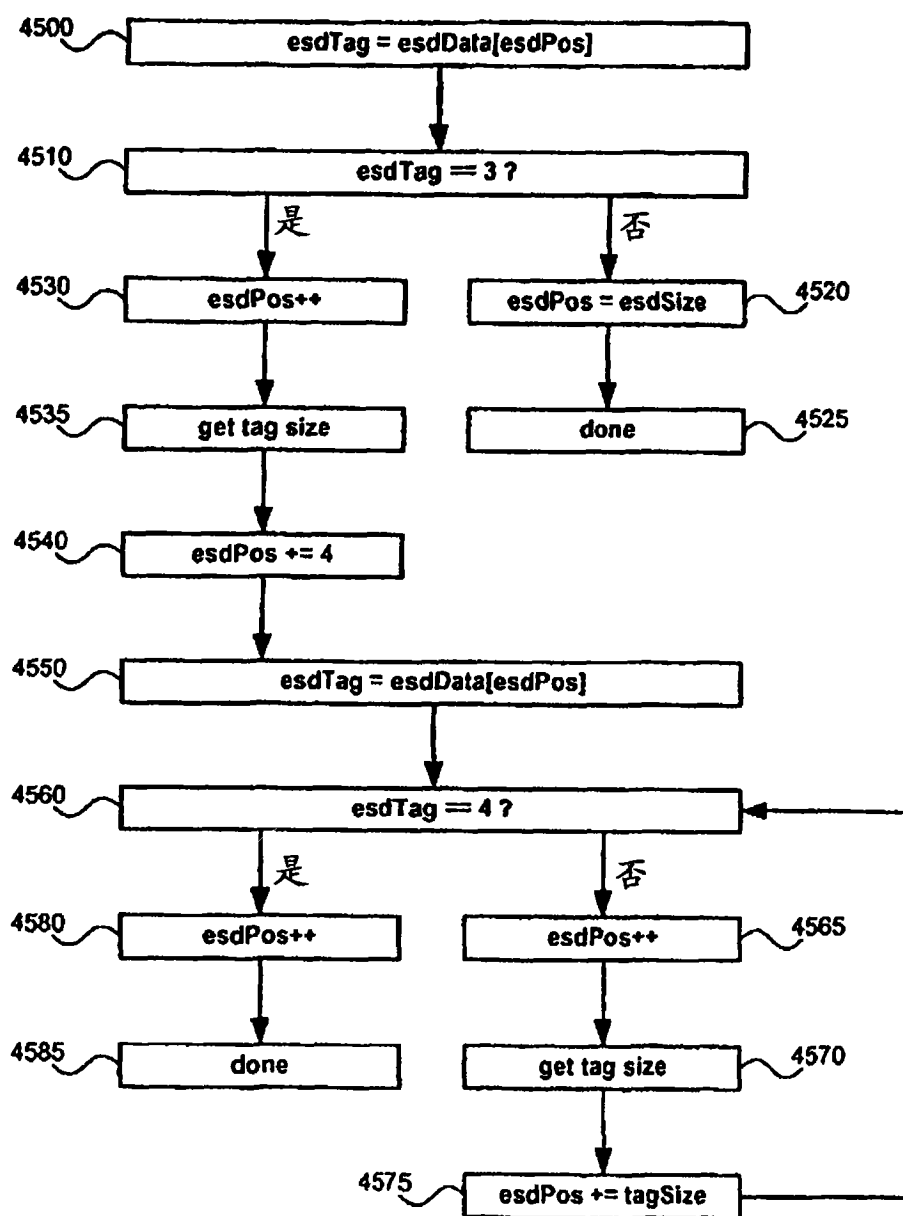


图 45

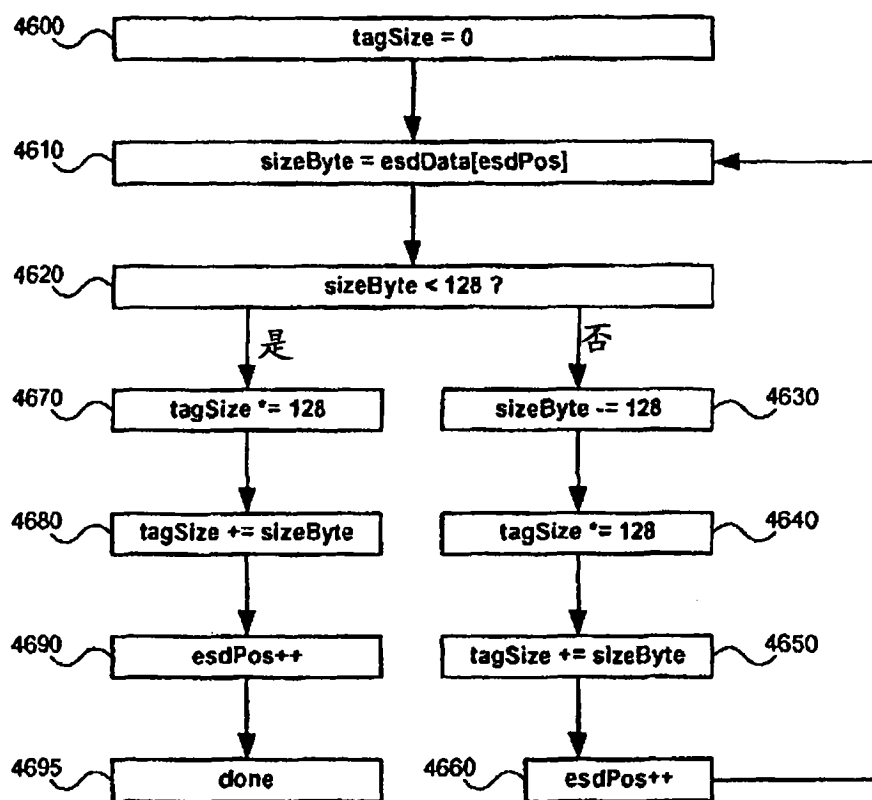


图 46

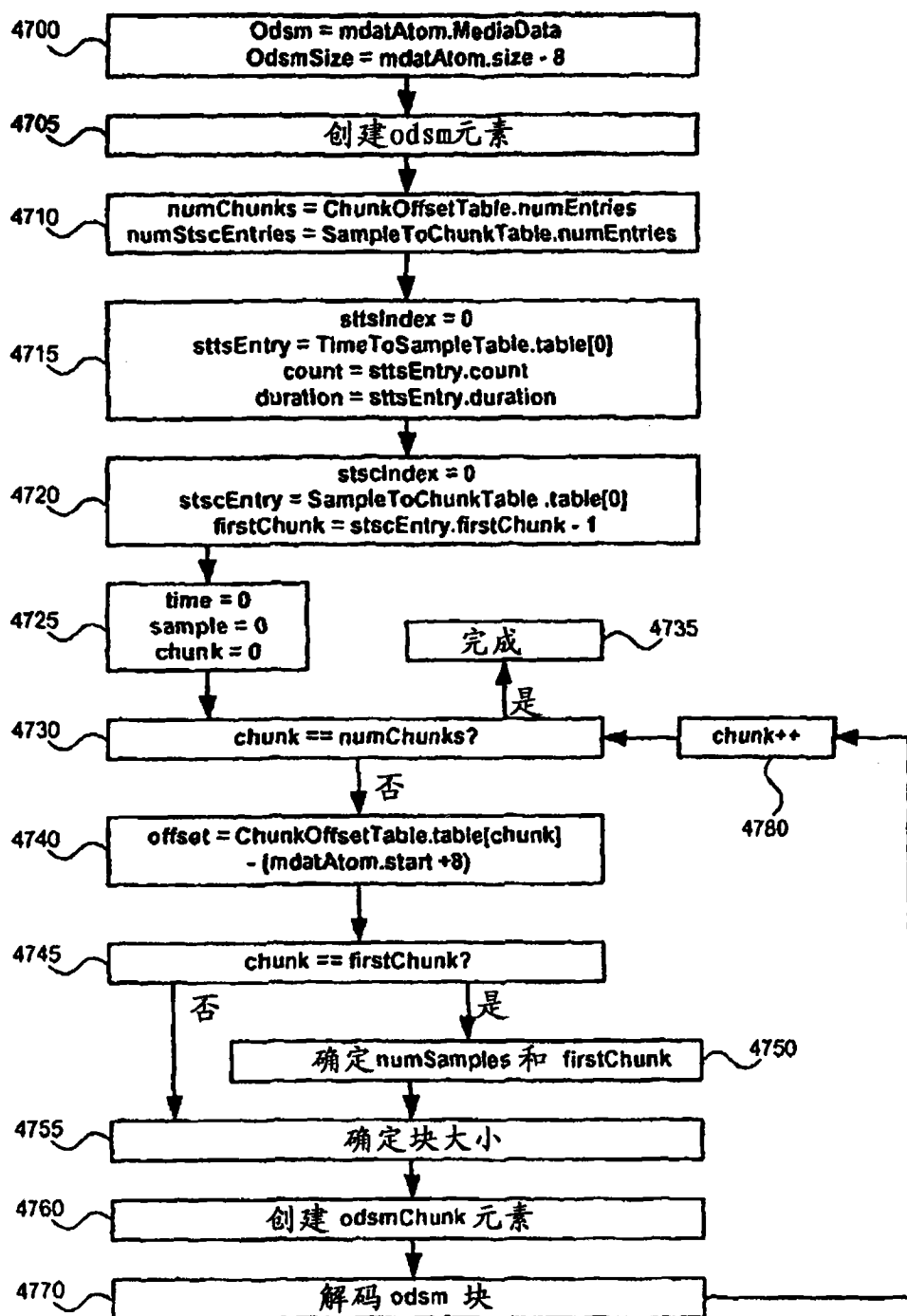


图 47

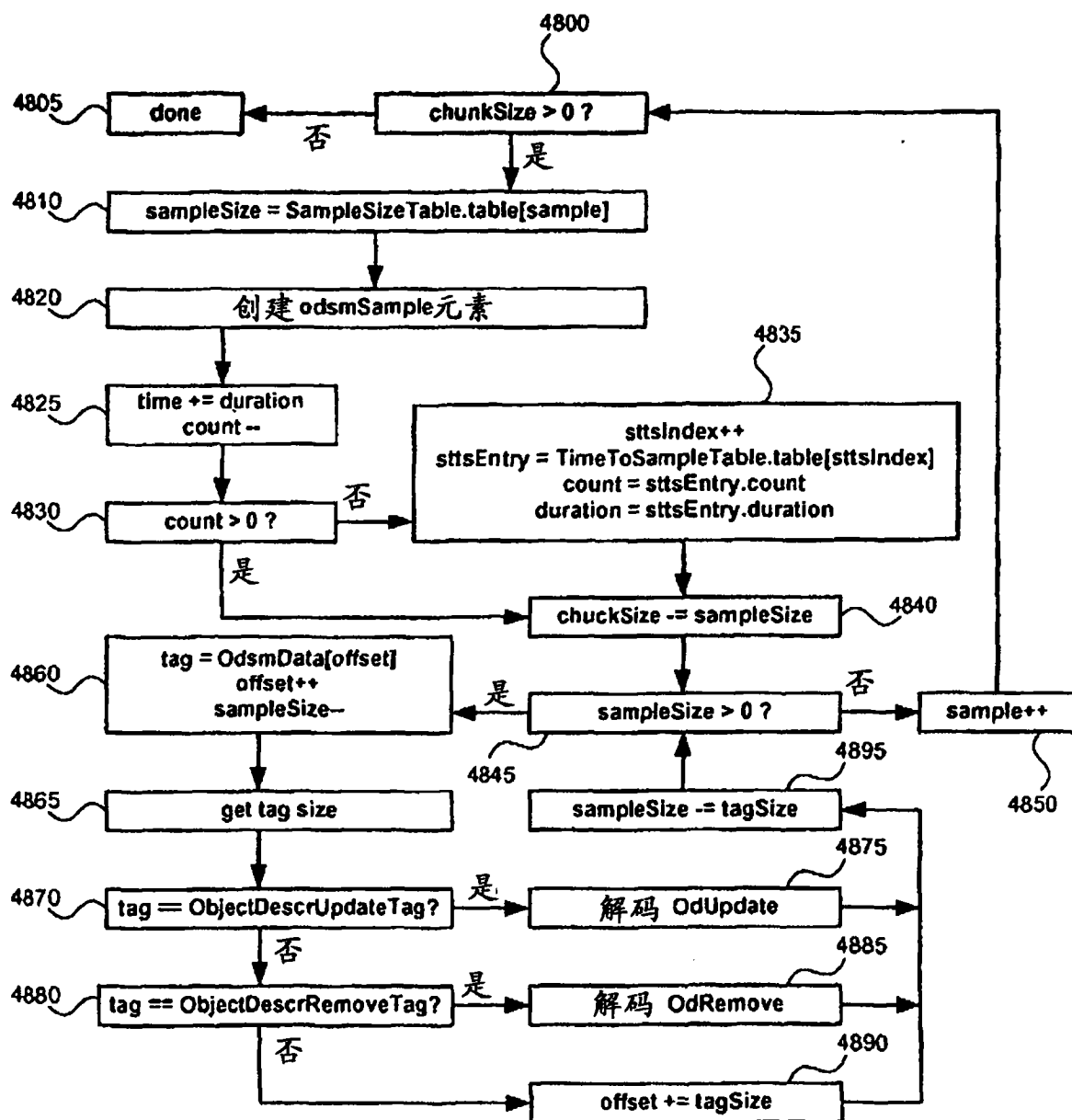


图 48

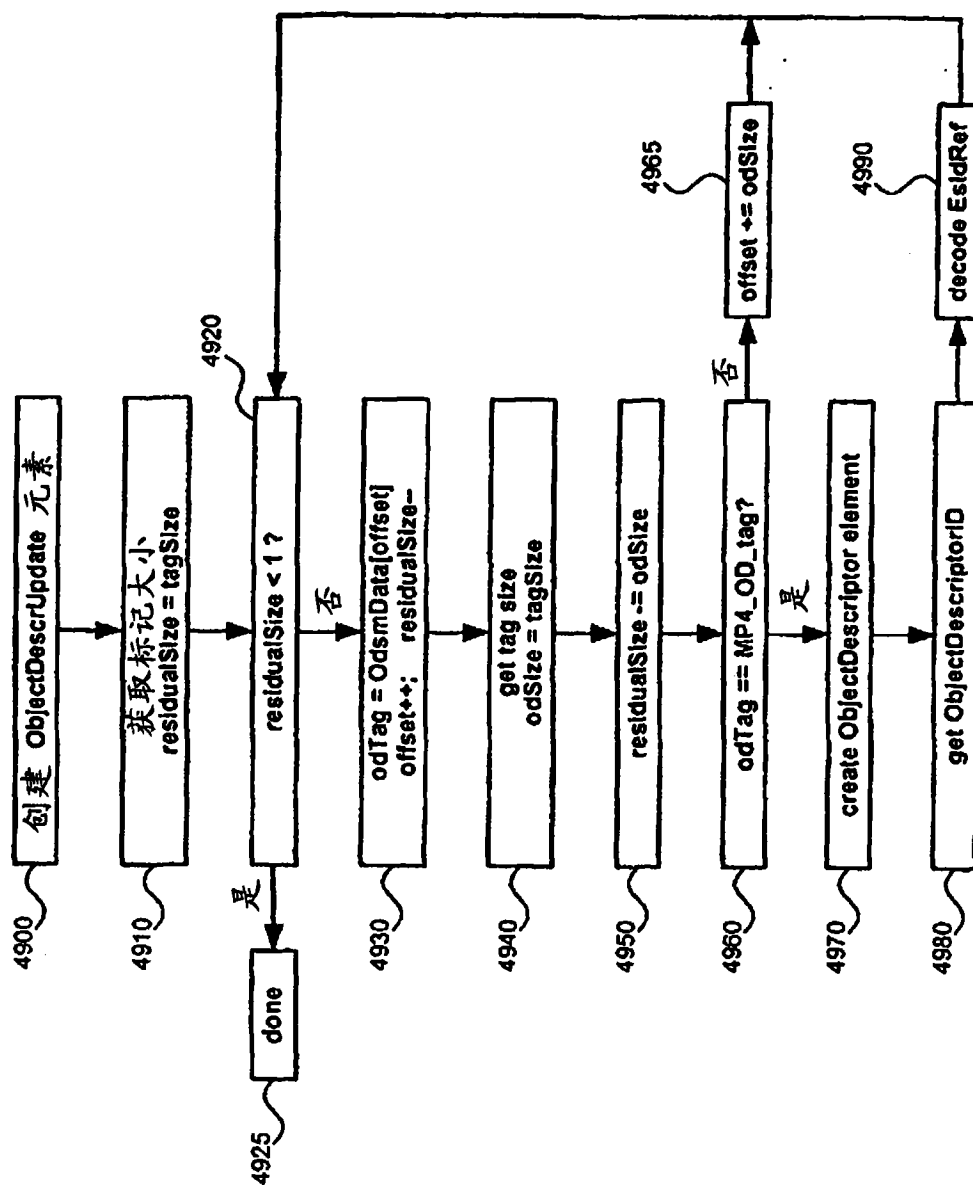


图 49

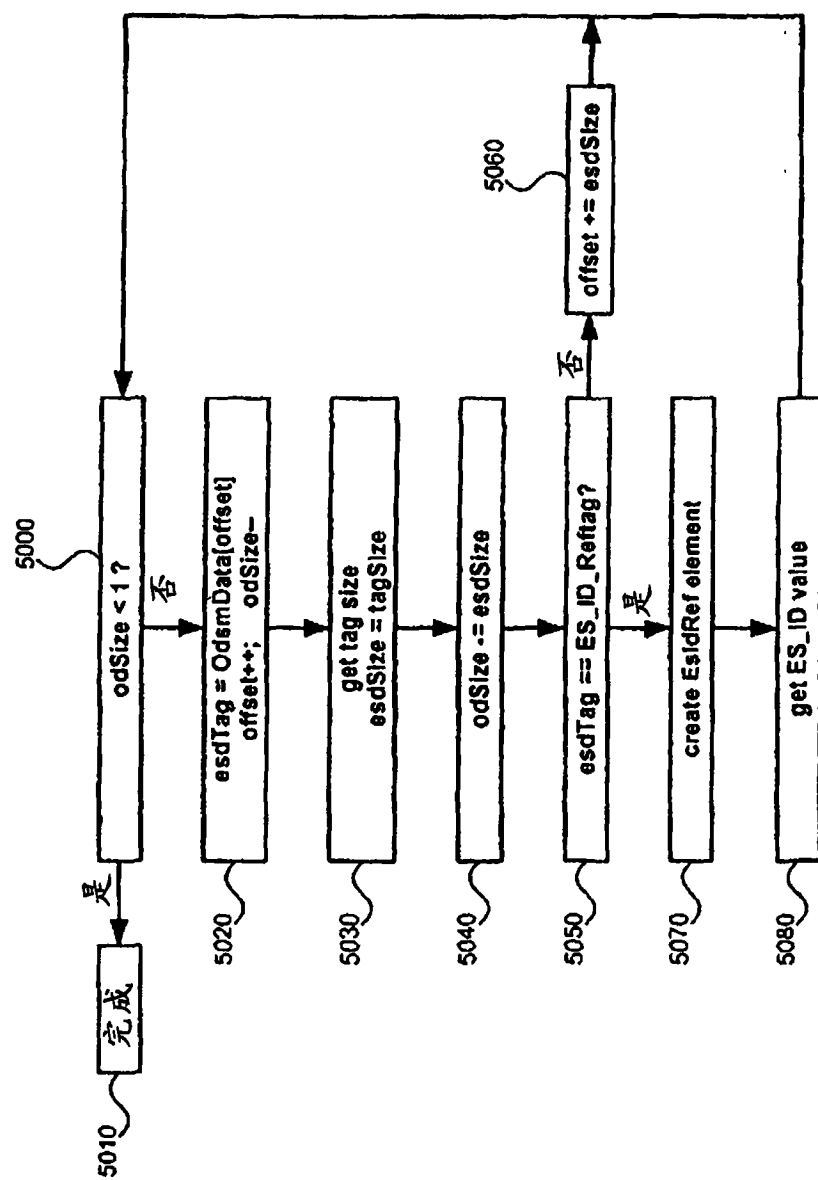


图 50

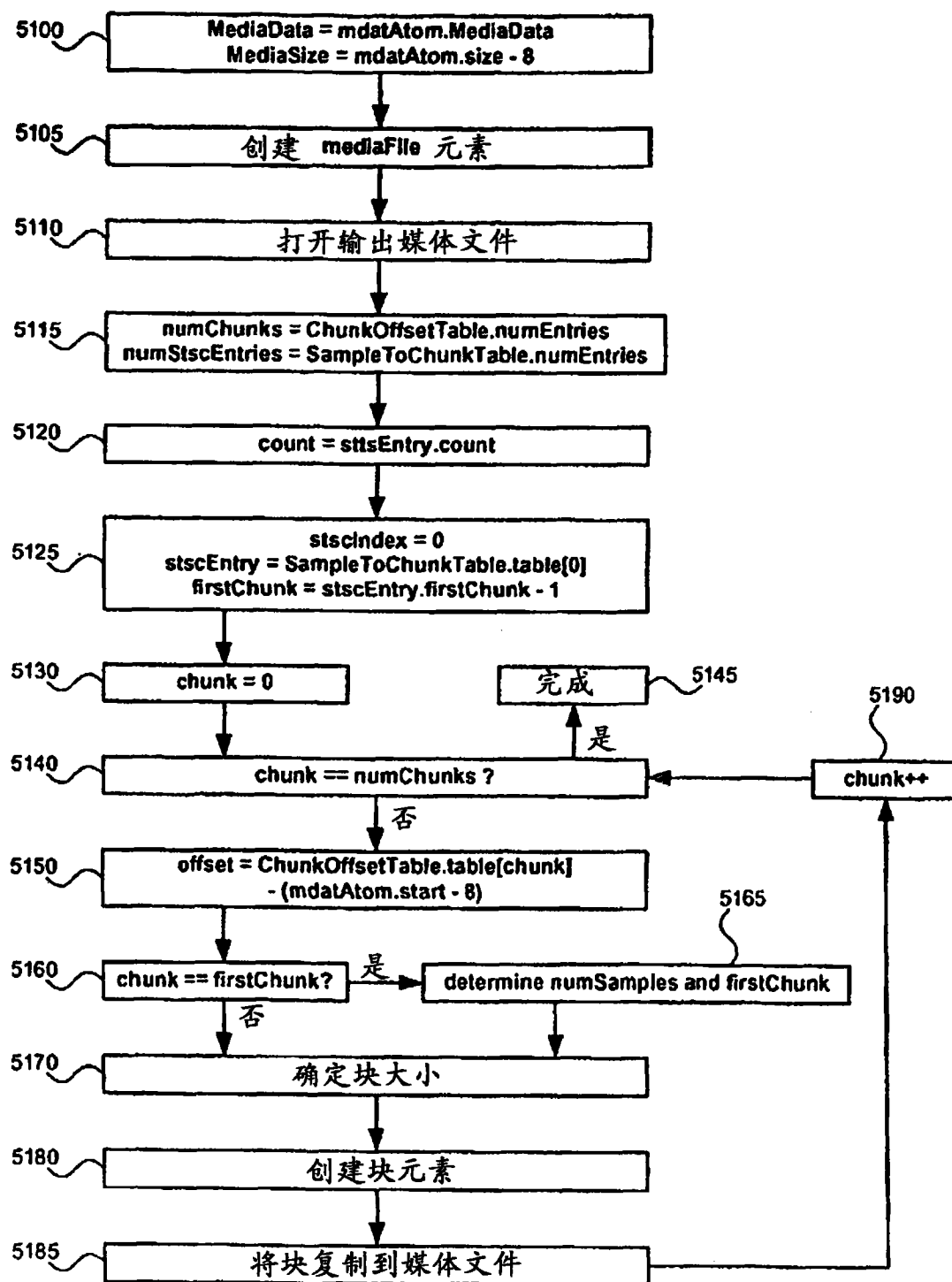


图 51

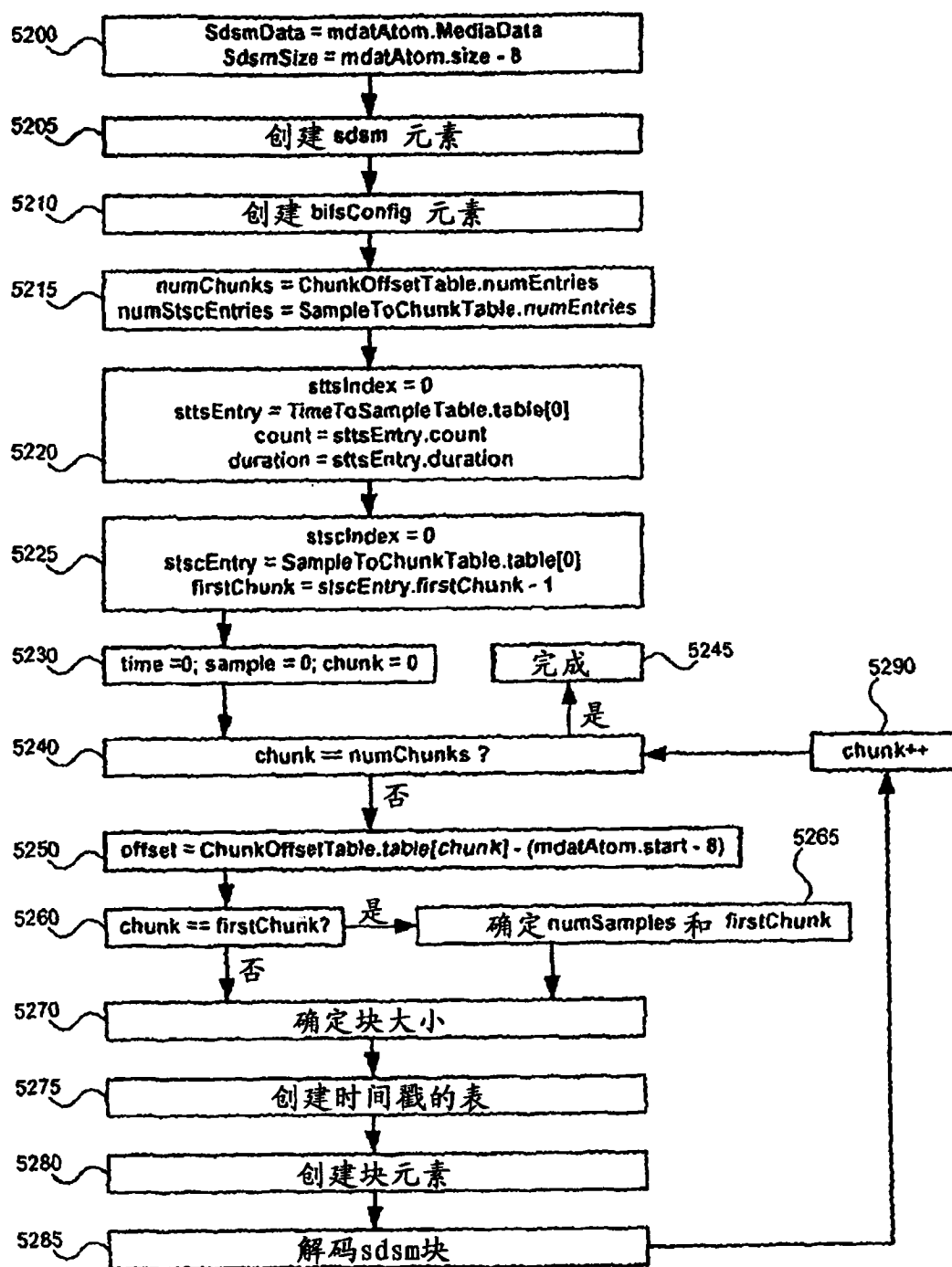


图 52

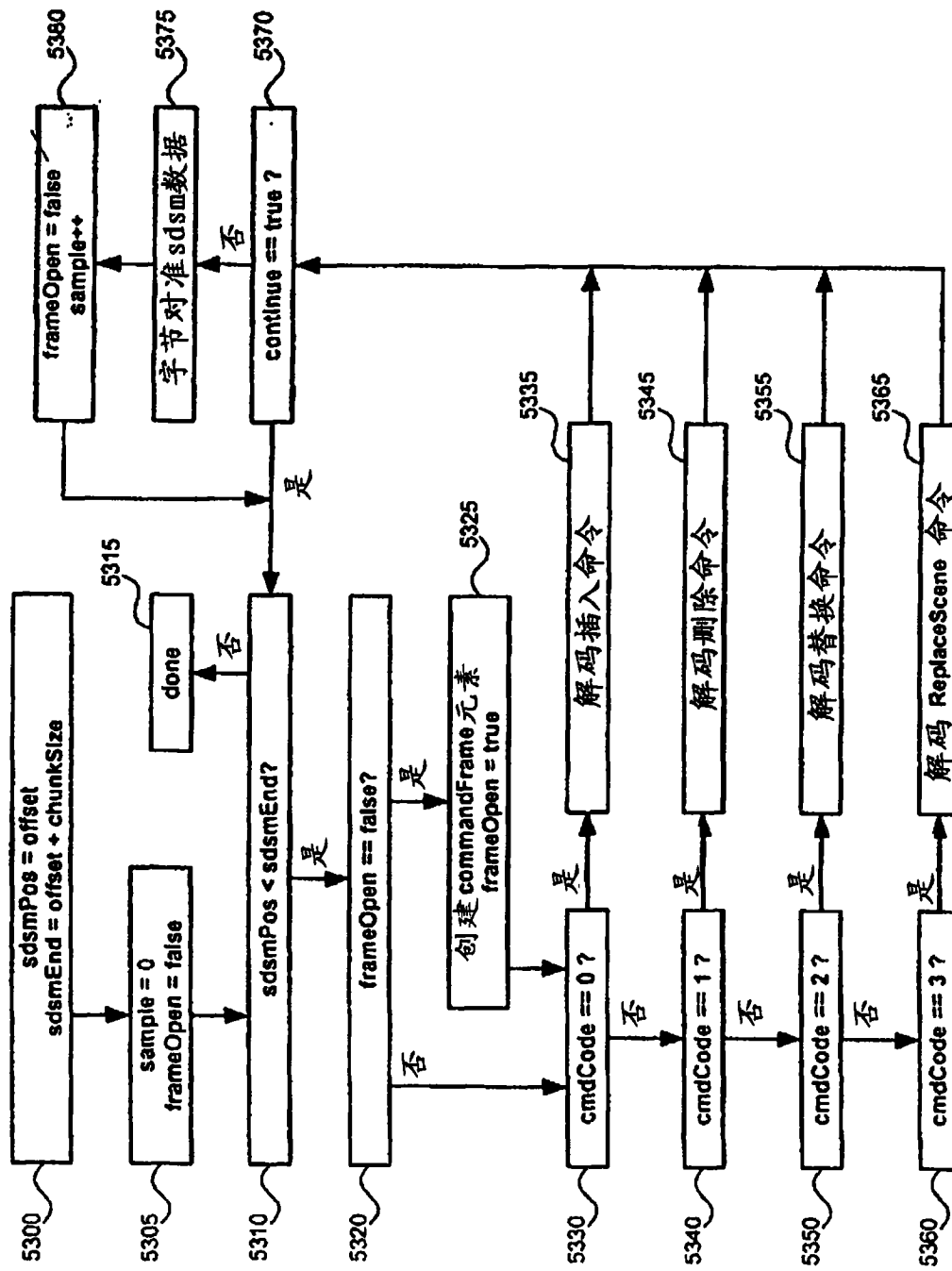


图 53

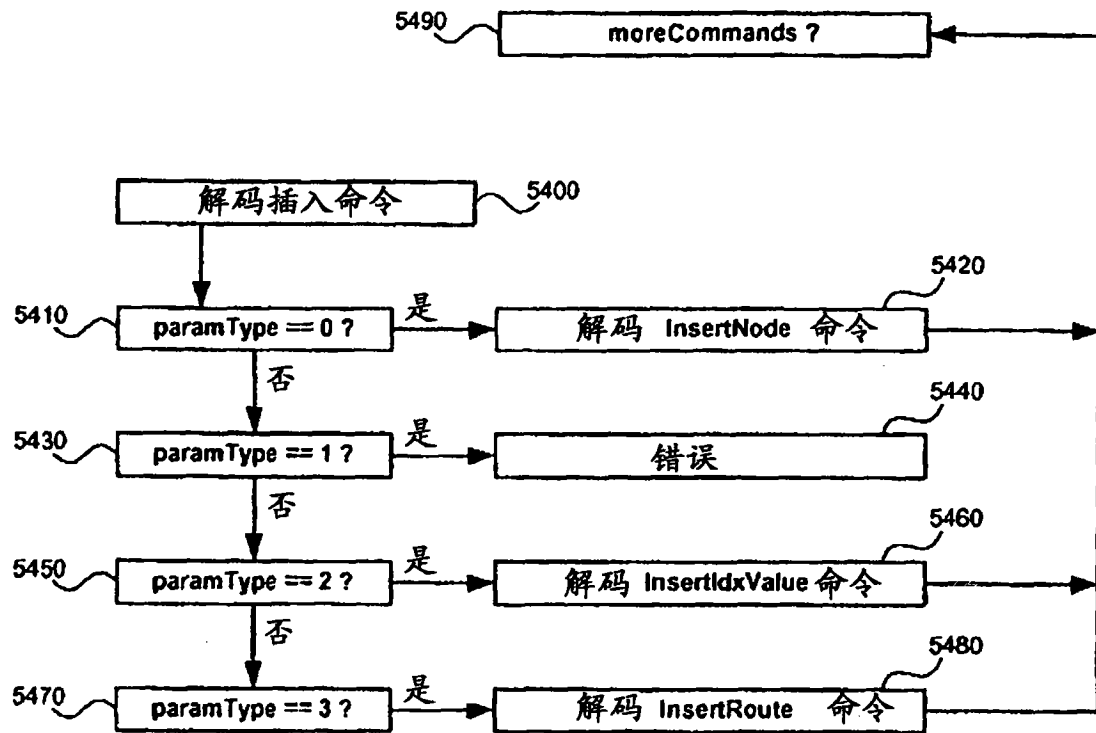


图 54

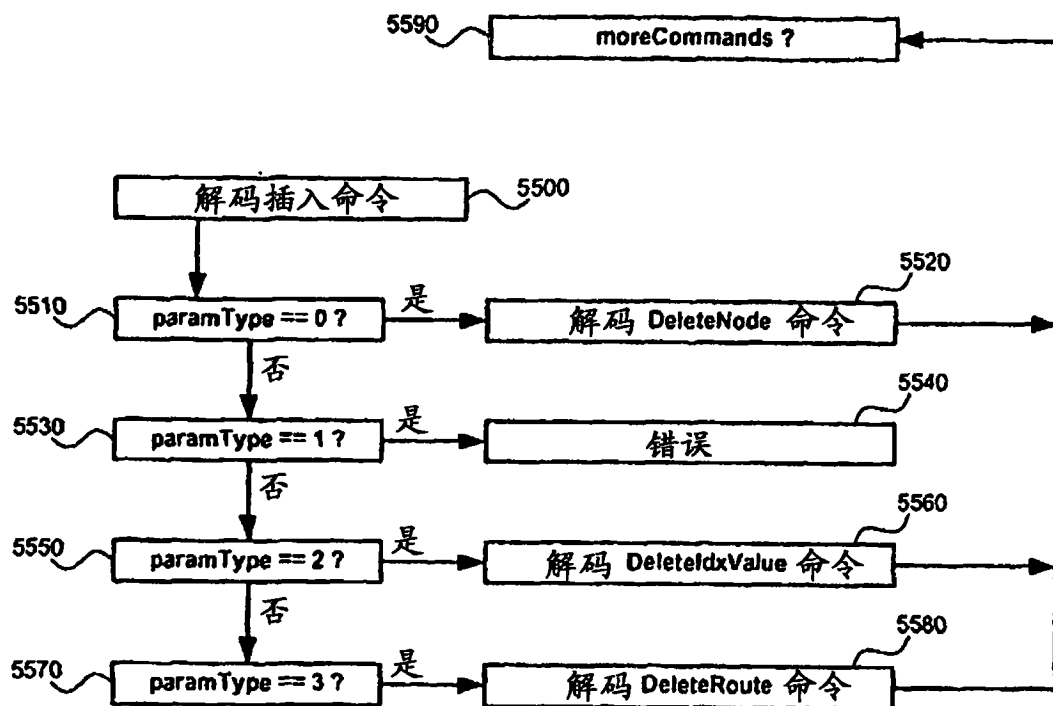


图 55

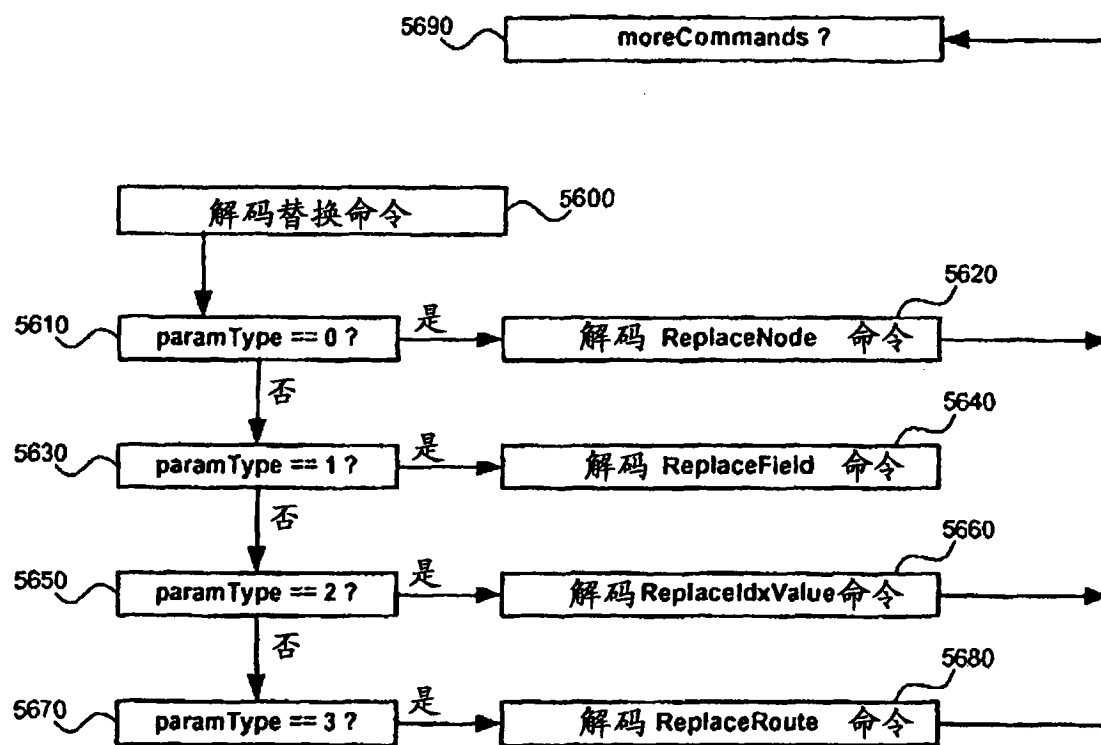


图 56

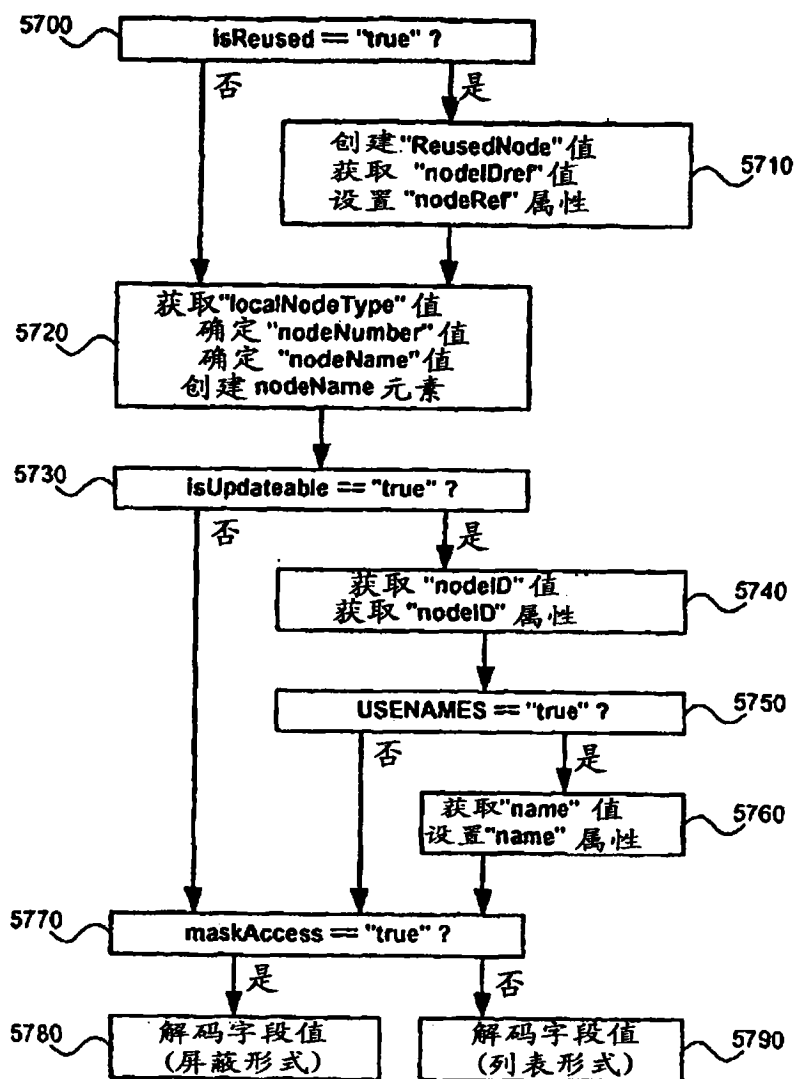


图 57

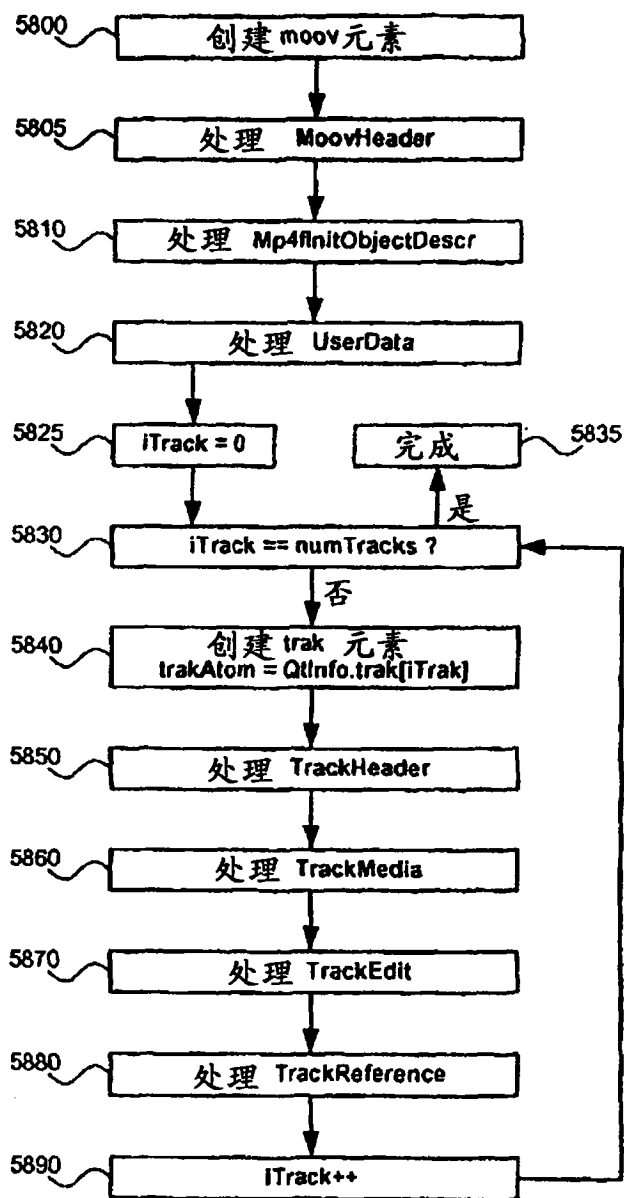


图 58

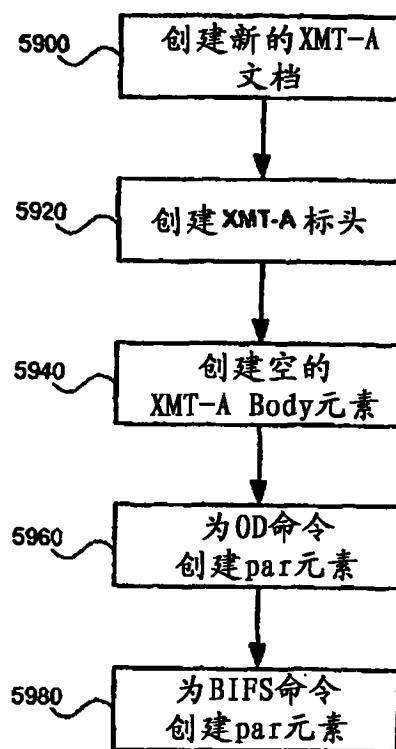


图 59

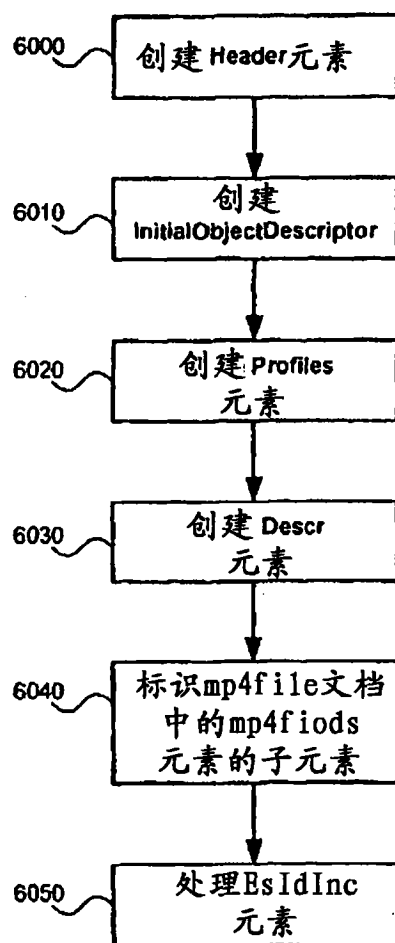


图 60

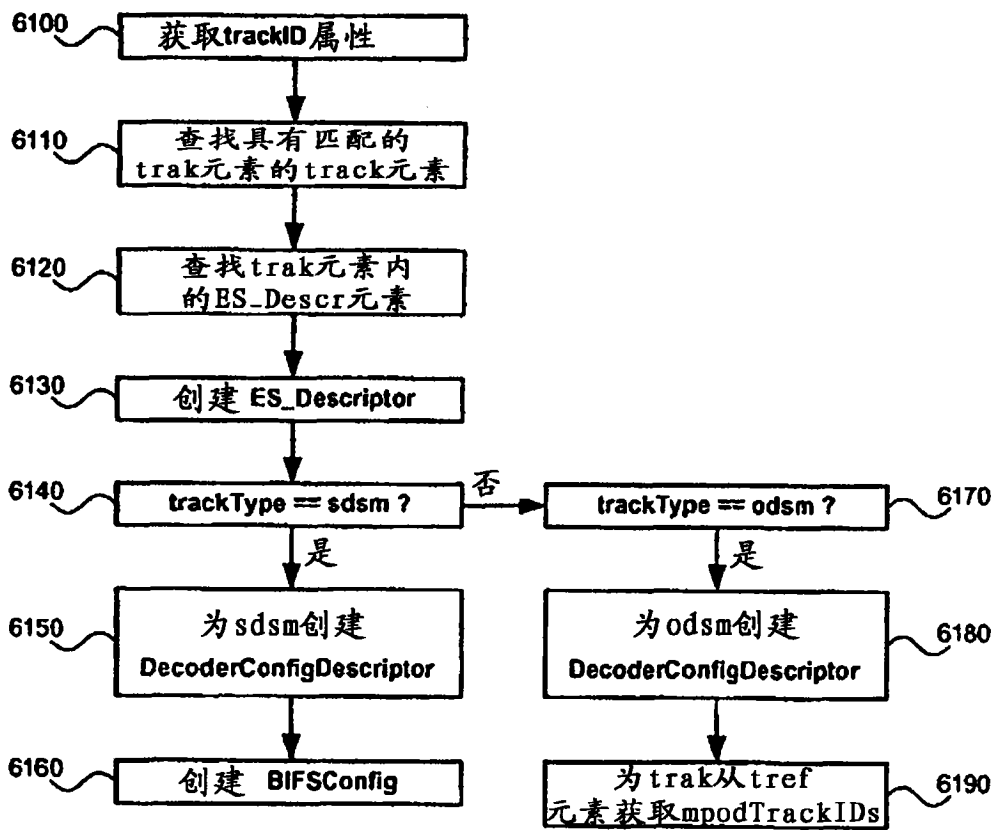


图 61

6200 mp4file	6220 XMT-A
"1"	"ObjectDescriptor"
"2"	"ClockReference"
"3"	"SceneDescription"
"4"	"Visual"
"5"	"Audio"
"6"	"MPEG7"
"7"	"IPMP"
"8"	"OCI"
"9"	"MPEGJ"

图 62A

6240 mp4file	6260 XMT-A
"1"	"MPEG4Systems1"
"2"	"MPEG4Systems2"
"32"	"MPEG4Visual"
"64"	"MPEG4Audio"
"108"	"JPEG"
"255"	"Unspecified"

图 62B

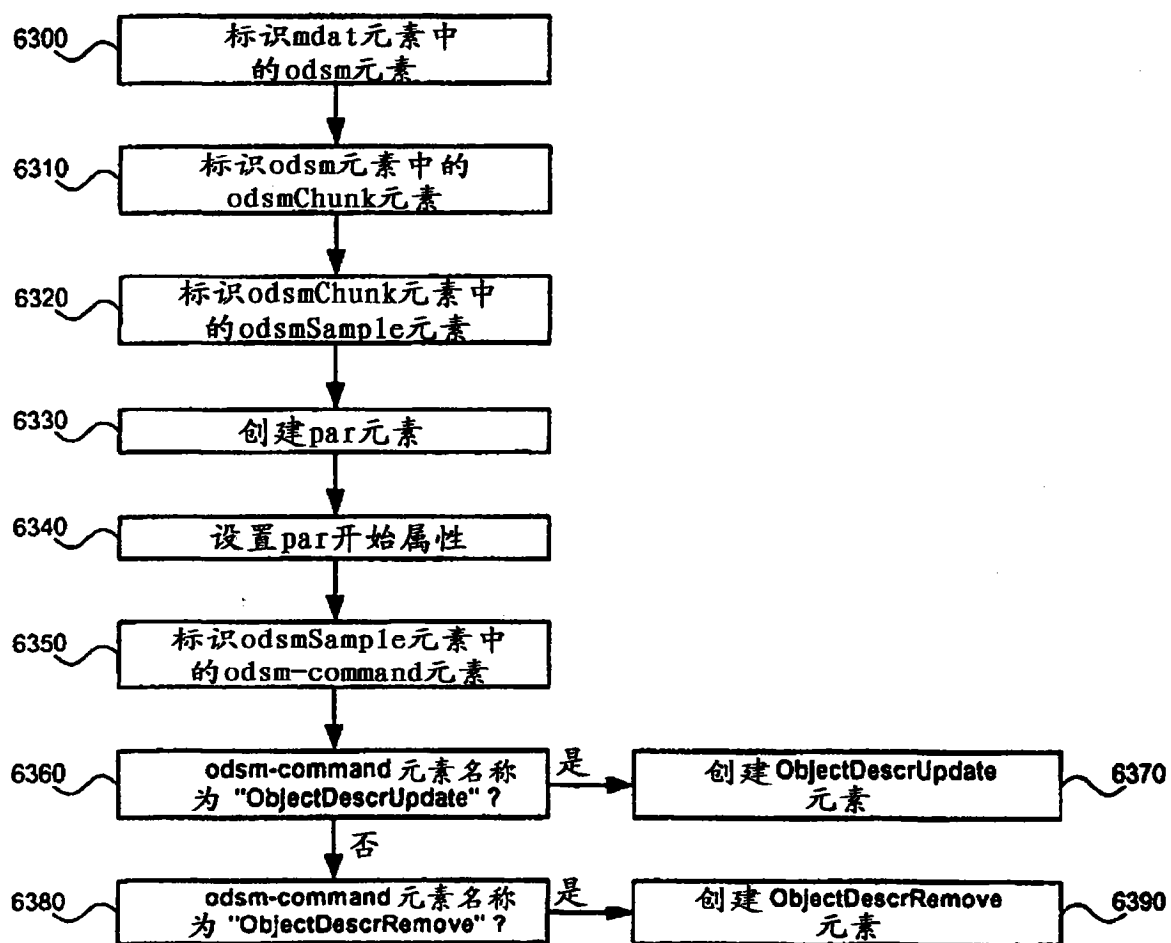


图 63

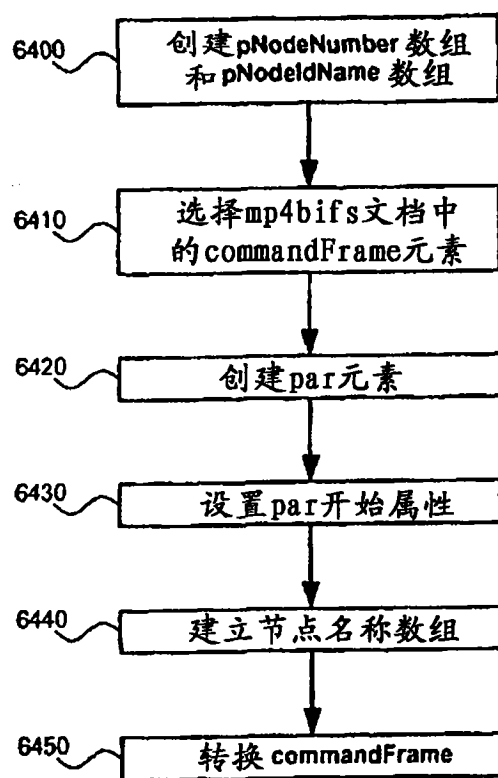


图 64

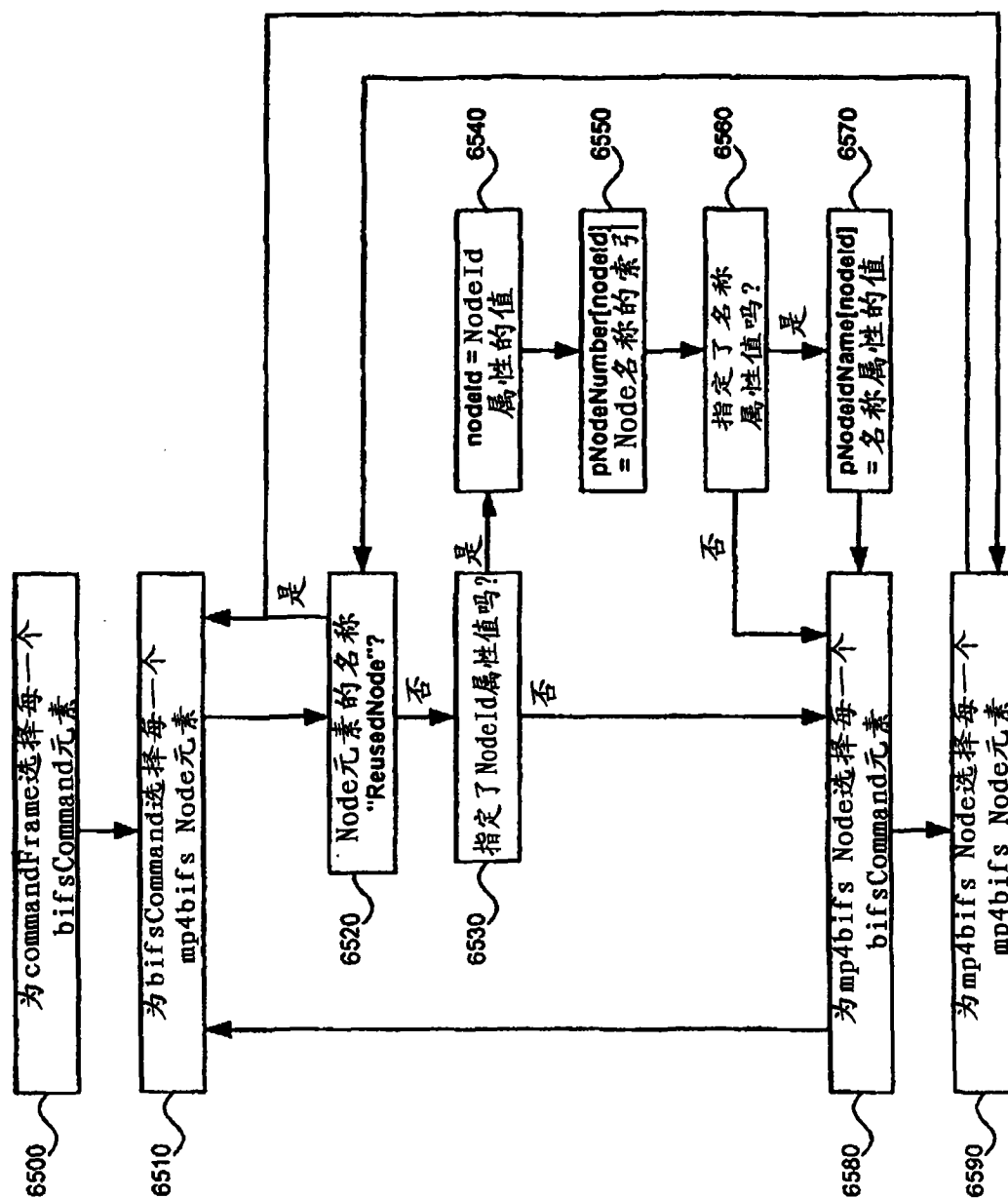


图 65