INTERNATIONAL STANDARD

ISO/IEC 14496-20

Second edition
2008-12-01

AMENDMENT 2
2010-03-01

Information technology — Coding of audio-visual objects —

Part 20: Lightweight Application Scene Representation (LASeR) and Simple Aggregation Format (SAF)

AMENDMENT 2: Technology for scene adaptation

Technologies de l'information — Codage des objets audiovisuels —

Partie 20: Représentation de scène d'application allégée (LASeR) et format d'agrégation simple (SAF)

AMENDEMENT 2: Technologies pour adaptation de scène

Reference number
ISO/IEC 14496-20:2008/Amd.2:2010(E)

© ISO/IEC 2010

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2010

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

 © ISO/IEC 2010 – All rights reserved

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Amendment 2 to ISO/IEC 14496-20:2008 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

© ISO/IEC 2010 – All rights reserved

Information technology — Coding of audio-visual objects —

Part 20:
Lightweight Application Scene Representation (LASeR) and Simple Aggregation Format (SAF)

AMENDMENT 2: Technology for scene adaptation

In Table 3, add the following new LASeR events:

Event name	Namespace	Description	Bubble	Canc.	IDL
"DisplaySizeChanged(A)"	Urn:mpeg:mpeg4:laser:200x	Occurs when the terminal screen or viewport size has been changed to the specified 'A' value. 'A' is expressed in Inches.			DisplaySizeChanged Event
"MemoryStatus(*a,b,c*)"	Urn:mpeg:mpeg4:laser:200x	Occurs when the terminal memory occupancy changes more than the each specified parameter.			MemoryStatusEvent

The parameters of **MemoryStatus** event are:

- *a*: number of graphics points. If the number of graphic points released/allocated by the terminal is strictly less than *a*, the event is not triggered.
- *b*: number of Unicode characters. If the number of Unicode characters released/allocated by the terminal is strictly less than *b*, the event is not triggered.
- *c*: size, in kilo-bytes, of the composition buffers (as described in 6.11.3). If the size released/allocated by the terminal is strictly less than *c*, the event is not triggered.

In 6.5.6, add the new attribute `delta` *to ExternalValueEvent:*

```
interface ExternalValueEvent : LASeREvent {
       readonly attribute float absoluteValue;
       readonly attribute boolean computableAsFraction;
       readonly attribute float fraction;
       readonly attribute signed long delta;
};
```

Attributes

- **absoluteValue**: This value represent the status of a resource of any kind, e.g. the remaining battery time.
- **computableAsFraction:** This value indicates whether a fraction can be computed from the absoluteValue.
- **fraction**: This value shall be between 0 and 1 inclusively and represent the status of the resource, e.g. the fraction of remaining battery time over operation time when fully charged.
- **delta**: The delta field carries the difference of absoluteValue between this occurrence of the event and the previous occurrence.

© ISO/IEC 2010 – All rights reserved

After 6.5.6.2, add the following new subclauses:

6.5.6.3 DisplaySizeChanged Event

```
interface DisplaySizeChangedEvent : LASeR Event {
              readonly attribute SVGLength diagonal;
              readonly attribute SVGLength screenWidth;
              readonly attribute SVGLength screenHeight;
};
```

No defined constants

Attributes

- **diagonal**: This value indicates the new screen or viewport diagonal size. The diagonal value must be consistent with the screenWidth and screenHeight values. The value is expressed in inches.
- **screenWidth**: This value indicates the new viewport width. The value is expressed in pixels.
- **screenHeight**: This value indicates the new viewport height. The value is expressed in pixels.

6.5.6.4 MemoryStatus Event

```
interface MemoryStatusEvent : LASeR Event {
       readonly attribute integer numberOfPoints;
       readonly attribute Integer numberOfCharacters;
       readonly attribute Integer compositionMemoySize;
};
```

No defined constants

Attributes

- **numberOfPoints**: number of points allocated/freed by the terminal.
- **numberOfCharacters**: number of Unicode characters allocated/freed by the terminal.
- **compositionMemoySize**: composition memory size of images allocated/freed by the terminal.

In 6.6.2.2, add the following new attributes and tables:

- ***adaptationType****:* the LASeRHeader can signal a set of adaptation constraints in the form of type and value pairs. These type and value pairs can then be referenced in an **AdpativeUpdateGroup** update to indicate that the given group of updates shall only be applied if the given adaptation constraints match the LASeR engine characteristics (e.g. required display size). The list of adaptation types is given in Table AMD2.1.
- ***adaptationTypeValue****:* indicates the value of the required LASeR engine characteristic described in the adaptation constraint type.
- ***adaptationConstraints****:* this attribute represents a list of indexes in the set of adaptation constraints required to process the scene segment.

© ISO/IEC 2010 – All rights reserved

Table AMD2.1 — adaptationType values

adaptationType	Description
0x01	Indicates the minimum viewport size required to process the associate LASeR update. The value describes the viewport diagonal, expressed in inches.
0x02	minimum memory size expressed in the same unit as min memory attribute.
0x03	adaptation filter as defined in 6.13
0x04	implicit Boolean (eg no value coded) evaluating to TRUE if the terminal is capable of mixing an additional audio object
0x05	Boolean evaluating to TRUE if the terminal is capable of composing an additional image object. One parameter is given to determine whether overlaying or regular composition is desired
0x06	Boolean evaluating to TRUE if the terminal is capable of composing an additional video object. One parameter (Boolean, one bit) is given to determine whether overlaying or regular composition is desired
0x07	Evaluates to TRUE if the indicated Interaction method is supported. Refer to Table AMD2.2 for a list interaction method types
0x08 – 0xFF	Reserved

Table AMD2.2 — InteractionMethodType values

InteractionMethodType	Description
0x00	None.
0x01	Stylus
0x02	Mouse
0x03	Full keyboard
0x04	Keypad (cellphone or similar)
0x05-0x80	ISO Reserved
0x81-xFE	User Reserved
0xFF	ISO Reserved

After 6.7.16, add the following new subclause:

6.7.17 AdaptiveUpdateGroup

6.7.17.1 Semantics

The ***AdaptiveUpdateGroup*** update is a wrapper of LASeR updates with an indication of the associated required adaptation criteria. It can be used for adaptation purposes.

6.7.17.2 Attributes

`adaptationConstraints`: this attribute represents a list of indexes in the set of adaptation constraints defined in the LASeRHeader required to process the associated group of updates.

© ISO/IEC 2010 – All rights reserved

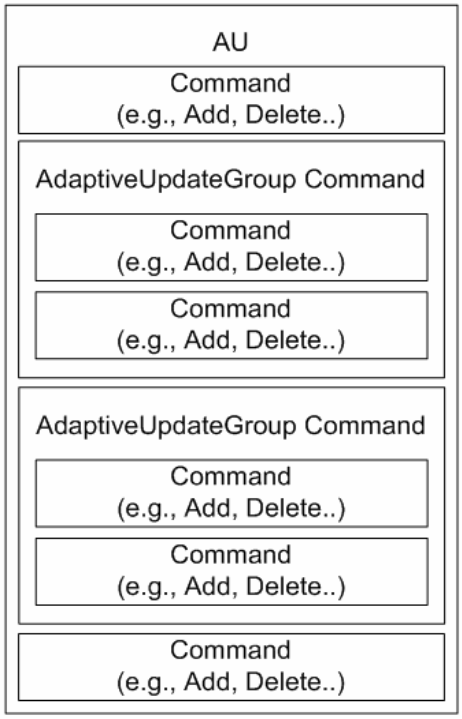

Figure AMD2.1 — LASeR updates in one LASeR Access Unit (AU).

Example of Scene Segment Level and Command Level Switching:

```
<LASeRHeader ... adaptationConstraints= " ... ">
   <AdaptiveSceneInfo>
      <constraint adaptationType="0x01" adaptationTypeValue=" ... "/>
      <constraint adaptationType=" ... " adaptionTypeValue=" ... "/>
      <constraint adaptationType=" ... " adaptionTypeValue=" ... "/>
   </AdaptiveSceneInfo>
</LASeRHeader>

<!-- a AU start -->
... commands(e.g., Add, Delete,...) ...

<AdaptiveUpdateGroup adaptationConstraints=" ... ">
... commands ...
</AdaptiveUpdateGroup>

<AdaptiveUpdateGroup adaptationConstraints=" ... ">
... commands ...
</AdaptiveUpdateGroup>

... commands ...

<!-- a AU End -->
```


© ISO/IEC 2010 – All rights reserved

At the end of 6.8.3, add the following new paragraph:

The xlink:href attribute of the a element may contain a special URL such as “tel:06778899”, thus using the rfc3966 definition in order to signal a “click to call” element. Other special URLs may be used to send messages to external systems for which the LASeR engine works as user interface.

After 6.8.59, add the following new subclause:

6.8.60 LASeR parsingSwitch

6.8.60.1 Semantics

The ***parsingSwitch*** element is a parsing construct which signals scene tree alternatives with respect to memory requirements. A LASeR engine loads ***parsingSwitch*** subtrees in the scene tree based on its memory occupancy. The **minMemory** attribute may be used on direct children elements of the ***parsingSwitch*** to indicate the foreseen memory requirements.

The children of the ***parsingSwitch*** element are loaded according to the **mode** attribute.

6.8.60.2 attributes

- **mode**: this attribute selects how children of the ***parsingSwitch*** element are loaded by the LASeR engine. It can take the following values
    - ”descending”: the children of the ***parsingSwitch*** element are listed in decreasing order with respect to their memory requirements and only the first subtree that fits in the available memory is loaded.
    - “ascending”: the children of the ***parsingSwitch*** element are listed in increasing order with respect to their memory requirements and only the last subtree that fits in the available memory is loaded.
    - “incremental”: the children of the ***parsingSwitch*** element are listed in increasing order with respect to their memory requirements. Subtrees are loaded in listed order and loading stops at the first subtree that doesn’t fit in the available memory.

© ISO/IEC 2010 – All rights reserved

Figure AMD2.2 illustrates the different modes of the ***parsingSwitch*** element. Numbers indicate the subtree order in the element and box sizes represent the memory footprint of each subtree.

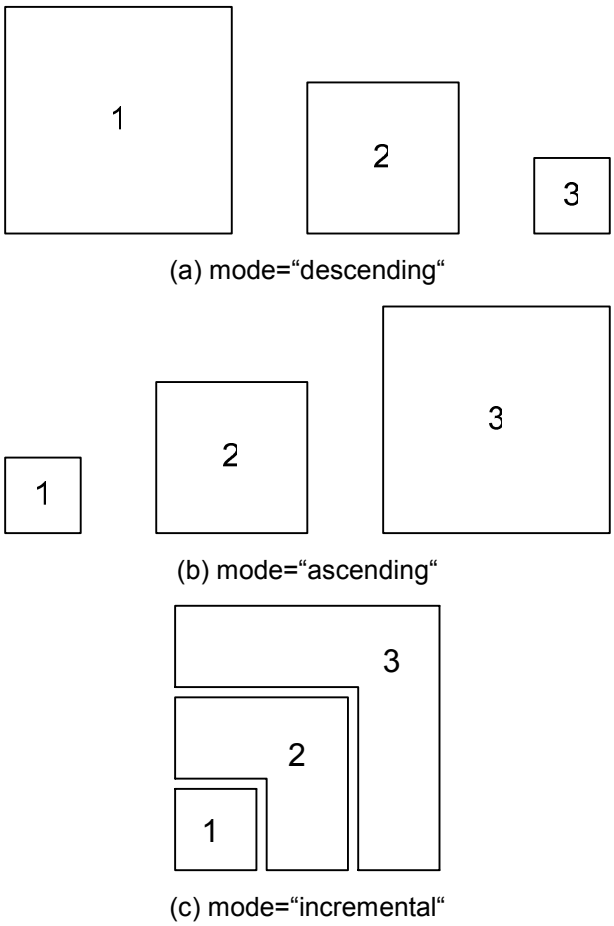

Figure AMD2.2 — illustration of *parsingSwitch* mode attribute

After 6.10, add the following new subclauses:

6.11 Attributes for static adaptation

Attributes for static adaptation are similar to "conditional processing attributes" of SVG, except that they are evaluated only when the element is loaded or whenever deemed necessary by the LASeR Engine implementation. Elements for which static adaptation attributes are evaluated to false, are still part of the DOM. An evaluation result of 'false' is equivalent to setting the display presentation attribute to none.

The static adaptation attributes cannot be animated or updated.

6.11.1 Container element attributes extension

The static adaptation attributes defined in this Subclause are added to the following elements: ***a, g, svg, switch, lsr:conditional, lsr:rectclip, lsr:selector, lsr:simplelayout***.

© ISO/IEC 2010 – All rights reserved

6.11.2 Display Size adaptation

The **minDisplaySize** attribute specifies the minimum display size required to display the element to which it applies. The display size refers to the size of the viewport (e.g., document window ...). The dimension of the display size is expressed in inches and represents the length of the viewport diagonal.

6.11.3 Memory adaptation

The **minMemory** attribute specifies a set of parameters which may be used by the LASeR engine to determine if it has enough available memory to display the subtree. The parameters are:

- The total number of points in the subtree.
- The number of Unicode characters in the subtree.
- The size, in kilo-bytes, of the composition buffers needed to load all the images of the subtree. The size of the composition buffer for an image is computed as the product of the width, the height and the number of bytes needed to represent a pixel value in the associated color format.
- The size, in kilo-bytes, of the composition buffers needed to process all the videos in the subtree. The size of the composition buffer for a video is computed in the same manner as for images.
- The sum of maximum sample rates of all the audio objects in the subtree.

Example:

```
<g minMemory=”1534 254 102 300 20”>
  ... some images, text, graphics ...
</g>
```

6.11.4 Resolution adaptation

The **minResolution** attribute specifies the minimum resolution required to display the element. The given resolution is in dpi (dots per inch). Note that a square pixel aspect ratio is assumed.

6.11.5 Font Resolution adaptation

The **minSize** attribute specifies the minimum physical font size required to render the element it applies to. The attribute can be specified on text elements and containing elements listed in 6.11.1. The physical size is expressed in points (pt).

The **minSizeBehaviour** attribute controls whether the element should be displayed or not, according to the following values:

- “freeze”: When the physical size of the font is less than **minSize**, local coordinate system scaling of the element (text or container) is adjusted so that the physical font size matches **minSize**.
- “remove”: When the physical size of the font is less than **minSize**, the element is not displayed.

Example 1:

```
<lsr :NewScene>
 <svg xmlns="http://www.w3.org/2000/svg"
     width="100%" height="100%" viewBox="0 0 400 400">
  <g id=”Adaptation”>
   <text x="200" y="200" font-size="20" minSize=”9pt” minSizeBehaviour=”freeze”>
LASeR PDAM2 </text>
  </g>
 </svg>
</lsr :NewScene>
```

© ISO/IEC 2010 – All rights reserved

Example 2:

```
<lsr :NewScene>
 <svg xmlns="http://www.w3.org/2000/svg"
     width="100%" height="100%" viewBox="0 0 400 400">
  <g id=”Adaptation” font-size="20" minSize=”9pt” minSizeBehaviour=”freeze”>
   <Rectangle x=”150” y=”150” width=”100” height=”100”/>
   <text x="200" y="200" > LASeR PDAM2 </text>
  </g>
 </svg>
</lsr :NewScene>
```

6.12 Attribute for best effort adaptation

- **Priority**: this attribute specifies an order of importance for the scene element on which it is set. This value is expressed as an integer ranging from 0 to 255. A high value represents a more important scene element than a low value. A LASeR engine may use this attribute to achieve better performance by stopping the processing of elements with low priorities, as if they had conditional processing attributes evaluated to false. In this case, all elements with the same priority shall be skipped together. The default priority value is 0 on non-container elements and the default value on container elements is the maximum of the values of this attribute on its children.

Example:

```
<lsr :NewScene>
   <svg xmlns="http://www.w3.org/2000/svg" .../>
      <text id=”title” x="18" y="42" font-size="26" font-family="Vernada" font-
weight="bold" fill="gray"> LASeR Adaptation </text>
     <switch>
         <g id=”group01” minDisplaySize=”3”>
            <image   xml:id="img01"   xlink:href="priority.jpg"   x="19"   y="47"
width="268" height="335" priority=”10”/>
            <g id="video">
               <rect   id=”rect”   x="27"   y="200"   width="250"   height="170"
fill="green" visibility="hidden">
                  <set     id=”set”     attributeName="visibility"     to="visible"
begin="3s"/>
               </rect>
               <video    id="video01"    xlink:href="video.mp4"    x="0"    y="210"
width="300" height="150" begin="4s" dur="12s" priority=”20”/>
           </g>
          </g>
        <g id=”group02” minDisplaySize=”10”>
           ...
        </g>
     </switch>
  </svg>
</lsr :NewScene>
```

In the above example, static adaptation and best effort adaptation attributes are used together in a scene. Static adaptation is processed first. According to the result of the evaluation, the ‘g’ element with ‘id=”group01”’ is being displayed. However, the terminal performances are not sufficient to render the scene. The LASeR engine then decides which element is rendered or not based on the priority value. The following table shows the priority value of each element in the scene. The LASeR engine skips elements in increasing order of priority. In the above example, according to terminal performances, the elements “rect”, “title” and “set” will be skipped first (priority=0), then the element “img01” (priority=10) and finally the element “video01” (priority=20).

© ISO/IEC 2010 – All rights reserved

scene element					priority value
root					
	text (id=”title”)				0
	g (id=”group01”)				20
		image (id="img01")			10
		g (id="video")			20
			rect (id=”rect”)		0
				Set (id=”set”)	0
			video (id="video01")		20

6.13 Adaptation Filter

In order to query tools defined in existing adaptation frameworks such as MPEG-21, LASeR provides declarative and programmatic means to access environment data usefull for fine-graine adaptation.

lsr:constraintFilter = "PredicateExpr"

This attribute specifies an XPath Predicate Expression which must evaluate to true for the element to be processed as per SVG ‘conditional processing attribute’. The refresh of this processing is implementation-specific. The context node for the Predicate expression is the current element. This attribute is applicable to all elements. It is not inheritable, not animatable and not updatable.

The following set of additional adaptation functions must be supported in the Xpath grammar:

Function: *number* **lsr_context_number**(context_keyword)

Function: *string* **lsr_context_string**(context_keyword)

Both **lsr_context_number** and **lsr_context_number** functions can be used to query the string or numerical value of a context keyword.

In 12.2.1, replace `class attr_any` *with:*

```
class attr_any {
  do {
    uint(extensionIDBits) extensionID;
    vluimsbf5 len; //length in bits
    if (extensionID == 2) { //LASeR AMD1
      bit(1) hasRequiredFonts;
      if (hasRequiredFonts) {
       attr_custom_byteAlignedString requiredFonts;
      }
      bit(1) hasX;
      if (hasX) {
        attr_custom_coordinate x;
      }
      bit(1) hasY;
      if (hasY) {
        attr_custom_coordinate y;
      }
    }
    else if (extensionID == 3) { //LASeR AMD2
      bit(1) hasMinDisplaySize;
      if (hasMinDisplaySize) {
       attr_display_size minDisplaySize;
      }
      bit(1) hasMinMemory;
      if (hasMinMemory) {
       attr_memory minMemory;
      }
      bit(1) hasPriority;
      if (hasPriority) {
       attr_priority priority;
      }
      bit(1) hasMinResolution;
      if (hasMinResolution) {
       attr_min_resolution minResolution;
      }
      bit(1) hasMinSize;
      if (hasMinSize) {
```

© ISO/IEC 2010 – All rights reserved

```
        attr_min_size minSize;
       }
       bit(1) hasMinSizeBehaviour;
       if (hasMinSizeBehaviour) {
        attr_min_size_behaviour minSizeBehaviour;
       }
       bit(1) hasConstraintFilter;
       if (hasConstraintFilter) {
        attr_constraint_filter constraintFilter;
       }
     } else {
       bit[len] toSkip;
     }
     bit(1) hasNextExtension;
   } while (hasNextExtension);
}
```

and replace `class element_any` *with:*

```
class element_any {
  uint(extensionIDBits) extensionID;
  vluimsbf5 len; //length in bits
  if (extensionID == 2) { //LASeR AMD1
    const vluimsbf5 reserved = 87; // or "const bit(10) reserved = 599;"
    vluimsbf5 occ2;
    for(int t=0;t<occ2+1;t++) {
      bit(2) ch5;
      switch(ch5){
        case 0:
          SVGAmd1Extension SVGAmd1Extension;
          break;
        case 2:
          LASeRAmd1Extension LASeRAmd1Extension;
          break;
      }
    }
  } else if (extensionID == 3) { //LASeR AMD2
    const vluimsbf5 reserved = 87; // or "const bit(10) reserved = 599;"
    vluimsbf5 occ3;
    for(int t=0;t<occ3+1;t++) {
      bit(2) ch6;
      switch(ch6){
        case 2:
          LASeRAmd2Extension LASeRAmd2Extension;
          break;
        default:
          reserved;
          }
      }
    }
  } else {
    bit[len] toSkip;
  }
}
```

and replace `class update_any` *with:*

```
class update_any {
  uint(extensionIDBits) extensionID;
  vluimsbf5 len; //length in bits
  if (extensionID == 2) { //LASeR AMD1
    const vluimsbf5 reserved = 87; // or "const bit(10) reserved = 599;"
    vluimsbf5 occ2;
    for(int t=0;t<occ2+1;t++) {
      const bit(2) reserved = 1;
      LASeRAmd1Command LASeRAmd1Command;
    }
  } else if (extensionID == 3) { //LASeR AMD2
    const vluimsbf5 reserved = 87; // or "const bit(10) reserved = 599;"
    vluimsbf5 occ2;
    for(int t=0;t<occ2+1;t++) {
      const bit(2) reserved = 1;
      LASeRAmd2Command LASeRAmd2Command;
    }
  } else {
    bit[len] toSkip;
  }
}
```


© ISO/IEC 2010 – All rights reserved

and replace `class extendedInitialisation` *with:*

```
class extendedInitialisation {
  // stringIDTable: external string ID references
  vluimsbf5 countG;
  for (int i = 0; i < countG; i++) {
    vluimsbf5 binaryIdForThisStringID[[i]];
    attr_custom_byteAlignedString stringID[[i]];
  }
  bit(1) hasExtension;
  if (hasExtension) {
    vluimsbf5 len;
    // globalStreamTable: external global streamID references
    vluimsbf5 countGS;
    for (int i = 0; i < count; i++) {
      vluimsbf5 localStreamIdForThisGlobal[[i]];
      byteAlignedStringClass globalName[[i]];
    }
    //adaptiveSceneIndicator
    static int constraintIndex = -1;
    static int constraintIndexBits = 0;
    bit(1) hasConstraintIndicator;
    if (hasConstraintIndicator) {
      vluimsbf5 list;
      for(int t=0; t<list;t++) {
        uint(8) constraintType;
        constraintIndex = constraintIndex+1;
        //cf table XXX
        switch (constraintType) {
        case 0x01:
          attr_display_size minDisplaySize;
          constraints[constraintIndex] = minDisplaySize;
          break;
        case 0x02:
          attr_memory_size minMemorySize;
          constraints[constraintIndex] = minMemorySize;
          break;
        case 0x03:
          attr_constraint_filter constraintFilter;
          constraints[constraintIndex] = constraintFilter;
          break;
        case 0x04:
          attr_audio_mix_constraint audioMixConstraint;
          constraints[constraintIndex] = audioMixConstraint;
          break;
        case 0x05:
          attr_image_compose_constraint imageComposeConstraint;
          constraints[constraintIndex] = imageComposeConstraint;
          break;
        case 0x06:
          attr_video_compose_constraint videoComposeConstraint;
          constraints[constraintIndex] = videoComposeConstraint;
          break;
        case 0x07:
          attr_input_method_constraint inputMethodConstraint;
          constraints[constraintIndex] = inputMethodConstraint;
          break;
        default:
       vluimsbf5 nb_bits;
       bit[nb_bits] any_constraint;
          break;
        }
      }
      constraintIndexBits = log2sup(constraintIndex);
    }
    bit[len2] remainingData;
    // len2 is defined implicitly as: len - sizeof(globalStreamTable) - sizeof(adaptiveSceneIndicator)
  }
}
```

© ISO/IEC 2010 – All rights reserved

and add the following classes:

```
class LASeRAmd2Extension {
  vluimsbf5 occ1;
  for(int t=0;t<occ1+1;t++) {
    bit(1) ch1;
    switch(ch1){
      case 0:
        element_parsingSwitch parsingSwitch;
        break;
    }
  }
}
class parsingSwitch_object_content {
 bit(1) opt_group;
 if (opt_group) {
  privateAttributeContainer privateAttributes;
 }
 bit(1) opt_group1;
 if (opt_group1) {
  vluimsbf5 occ1;
  for(int t=0;t<occ1;t++) {
   vluimsbf5 encoding-length; //size of encoded child0[[t]] in bytes
   attr_custom_align encoding-align-before;
   elements child0[[t]];
   attr_custom_align encoding-align-after;
  }
 }
}
class element_parsingSwitch {
  bit(1) has_id;
  if (has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if (has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
  bit(1) has_parsing_mode;
  if (has_parsing_mode) {
     attr_parsing_mode mode;
  }
  parsingSwitch_object_content child0;
}
class LASeRAmd2Command {
  vluimsbf5 occ3;
  for(int t=0;t<occ3+1;t++) {
    bit(1) ch5;
    switch(ch5){
    case 0:
      update_AdaptiveUpdateGroup adaptiveUpdate;
      break;
    }
  }
}
class update_AdaptiveUpdateGroup {
  vluimsbf5 list;
  for(int t=0; t<list;t++) {
   uint(contraintIndexBits) constraintIndex;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
  vluimsbf5 encoding-length; // size of encoded updates in bytes
  attr_custom_align encoding-align-before;
  updateListType_content updates;
  attr_custom_align encoding-align-after;
}
```


© ISO/IEC 2010 – All rights reserved

In 12.2.2, add the following classes:

```
class attr_display_size {
   vluimsbf5 minDisplaySize; //value in inches
}
class attr_memory {
  vluimsbf5 nb_points;
  vluimsbf5 nb_chars;
  vluimsbf5 image_size;
  vluimsbf5 video_size;
  vluimsbf5 audio_rates;
}
class attr_priority {
  uint(8) priority;
}
class attr_constraint_filter {
   attr_custom_byteAlignedString constraintFilter;
}
class attr_min_resolution {
   vluimsbf5 min_resolution; //in dpi
}
class attr_min_size {
   vluimsbf5 min_size;  //in points
}
class attr_min_size_behaviour {
  uint(1) flag; //0: none (default), 1: value
  if (flag) {
    uint(2) min_size_behaviour;   //0: freeze, 1: remove
  }
}
class attr_parsing_mode {
   bit(2) mode;    //0: descending, 1: ascending, 2: incremental
}
class attr_audio_mix_constraint {
}
class attr_image_compose_constraint {
 bit(1) check_overlay;
}
class attr_video_compose_constraint {
 bit(1) check_overlay;
}
class attr_input_method_constraint {
 vluimsbf5 input_method_type;
}
```

In the LASeRML directory of electronic attachment of ISO/IEC 14496-20, add the following enclosed attachments:

laser3.xsd

laser-datatypes3.xsd

svg3.xsd

© ISO/IEC 2010 – All rights reserved

ICS 35.040

Price based on 13 pages

© ISO/IEC 2010 – All rights reserved