ASC Framing Decision List (ASC FDL)

Advanced Data Management Subcommittee

THE AMERICAN SOCIETY OF CINEMATOGRAPHERS

Specification v1.0

March 03, 2023

Table of Contents

1. Introduction

ASC FDLs are a set of instructions for how to view media in any application. The ASC FDL provides a mechanism to document framing decisions through all phases of a project's life cycle, from pre-visualization through post-production. The FDL can exist in the form of a sidecar JSON file, or embedded into another data structure like camera original files. Any time an application is rendering media to go to another department or person, an accompanying set of ASC FDL data should be created to inform how to view the newly generated content. This ASC FDL data can be applied to view the intended framing.

2. Scope

This document specifies format definitions and operations for the ASC Framing Decision List (ASC FDL), for the exchange and creation of framing data. This document also contains the information required to implement an "ASC FDL-compliant" software/hardware system.

The ASC FDL is intended for use in media production workflows and has been optimized to support the ability for a department or person to be able to easily recreate framing made by others, while limiting human error and the traditional operational burden. A common example is when on-set framing decisions have been made by an Image Author that then need to be conveyed to the vendor processing material for editing and review. Often, the processes used for re-creating that on-set framing decision are manual and prone to human error.

While ASC FDL is able to track framing decisions per shot, tracking per frame is currently out of scope.

3. Conformance Notation

Normative text is text that describes elements of the design that are indispensable or contains the conformance language keywords: "shall", "should", or "may". Informative text is text that is potentially helpful to the user, but not indispensable, and can be removed, changed, or added editorially without affecting interoperability. Informative text does not contain any conformance keywords.

All text in this document is, by default, normative, except: the Introduction, any section explicitly labeled as "Informative" or individual paragraphs that start with "Note:"

The keywords "shall" and "shall not" indicate requirements strictly to be followed in order to conform to the document and from which no deviation is permitted.

The keywords, "should" and "should not" indicate that, among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.

The keywords "may" and "need not" indicate courses of action permissible within the limits of the document. The keyword “reserved” indicates a provision that is not defined at this time, shall not be used, and may be defined in the future. The keyword “forbidden” indicates “reserved” and in addition indicates that the provision will never be defined in the future.

The following font/font color formatting is used throughout this document:

- JSON formatting has been used to show examples of ASC FDL code
- Purple is used for JSON elements, attributes and values, e.g. aspect_ratio

4. References

- JavaScript Object Notation (JSON)
- Unicode Transformation Format 8-bit (UTF-8)

5. Concepts and Semantics

5.1 ASC FDL

An ASC FDL is a self-contained file with an .fdl extension. The file is formatted utilizing JSON and does not require a specific directory structure. FDL data can also be written into camera files, or shared within other file types that support it.

5.2 Locating ASC FDL Files

Requiring specific folder structures, file names or how an application may locate an FDL is not in scope for this specification. It is by design that FDL files can be managed in any directory structure a user may wish, along with any filename the user wishes. It is up to each implementation how they would like the user to select any FDL to be used on any shot.

5.3 Applying ASC FDL Files

ASC FDLs are not intended to be a set of render instructions, but rather instructions for how to view media within an application. It is recommended that any time an application is applying an ASC FDL to media, the Canvas dimensions should be compared against the resolution of the source material. If these do not match, the user should be warned that they are trying to apply an FDL that does not match the Canvas dimensions resolution.

6. ASC FDL File Properties

6.1 Schema

An ASC FDL Manifest File is a JSON document.

6.2 Character Encoding

ASC FDL Manifests shall be encoded using UTF-8 character encoding.

7. Classes

Description:

ASC FDL files are organized in various sections, each containing its own set of attributes. The ASC FDL classes consist of: Header, Framing Intent, Context, Canvas, Framing Decision, Canvas Template.

7.1 Header

```
"uuid": "BCD142EB-3BAA-4EA8-ADD8-A46AE8DC4D97",
"version": {"major": 0, "minor": 1},
"fdl_creator": "ASC FDL Committee",
"default_framing_intent": "FDLSMP01",
```

7.1.1 UUID

Description:

The UUID field is a globally unique string that differentiates an individual ASC FDL file from any other. The format of the uuid must use the canonical textual representation. The 16 octets of a uuid are represented as 32 hexadecimal (base-16) digits, displayed in 5 groups separated by hyphens, in the form 8-4-4-4-12 for a total of 36 characters (32 alphanumeric characters and 4 hyphens). As an example:

afe122be-59d3-4360-ad69-33c10108fa7a

The uuid attribute is one of two fields used to determine which FDL should be applied to which shot: asc_fdl>uuid and framing_decision>id.

The ASC FDL uuid attribute will utilize the ISO/IEC 9834-8:2008 specification, enabling applications to produce 128-bit identifiers that are either guaranteed to be globally unique, or are globally unique with a high probability. For more information please see: ISO/IEC 9834-8:2008

As an example:

```
"uuid": "BCD142EB-3BAA-4EA8-ADD8-A46AE8DC4D97",
```

Required Field: Yes
Data Type: string

7.1.2 Version

Description:

The ASC FDL specification may be updated over a course of time and each updated version officially released will have a version number. All ASC FDL files should contain the implementer's ASC FDL version within the header of the ASC FDL file.

As an example:

```
"version": {"major": 0, "minor": 1},
```

Data Type: "major": integer, "minor": integer

Required Field: Yes

7.1.3 FDL Creator

Description:

This field can take a string indicating who created the FDL document. A user or implementation may choose to include the user or the software name/version that was used to create it. If any software receives an FDL and adds data to it, the fdl_creator field should represent the most recent modifying author when creating a new version of the FDL.

As an example:

```
"fdl_creator": "ASC FDL Committee",
```

Data Type: string

Required Field: No

Default Value: Omitted

7.1.4 `default_framing_intent`:

Description:

Having a specific framing intent flagged as default will allow implementations to know which one to automatically use, when multiple framing intents exist within a single FDL.

The default_framing_intent contains the Framing Intent id of the primary Framing Intent.

As an example:

```
"default_framing_intent": "FDLSMP01",
```

Data Type: string

Required Field: No

7.2 Framing Intents

```
"framing_intents": [{
      "label": "2.39-1 Framing",
      "id": "FDLSMP01",
      "aspect_ratio": {"width": 239, "height": 100},
      "protection": 0.05
    }
```

Description:

Creating a Framing Intent is the first key step to creating an FDL. It represents the intended aspect ratio, unbounded by the constraints of any camera or device. This is the region within which a cinematographer will compose content intended for the viewing audience. An FDL may contain multiple Framing Intents.

Child Elements:

7.2.1 label:

The label field in an FDL is a field to provide a human readable title for a Framing Intent.

Note: Using UTF-8 strings with non-ASCII characters may have compatibility issues on some software.

Note: An implementation may choose to show this field in their interface for allowing a user to choose which framing intent from the selected FDL they would like to apply.

As an example:

```
"label": "2.39-1 Framing",
```

Data Type: string

Required Field: No

7.2.2 id:

The Framing Intent id field is meant to provide a means of identification for a Framing Intent. This id is not universally unique, but no other Framing Intent id within a single FDL will use the same id. The Framing Intent id field shall be between 1-32 characters in length and be limited to the use of alphanumeric characters and underscores.

As an example:

```
"id": "FDLSMP01",
```

Data Type: string

Required Field: Yes

7.2.3 aspect_ratio:

The aspect_ratio field represents the image author's original intention. An aspect_ratio shall utilize width and height integers to provide an exact ratio for higher accuracy, non-integers (e.g. 1.78) are not allowed.

As an example:

```
"aspect_ratio": {"width": 239, "height": 100},
```

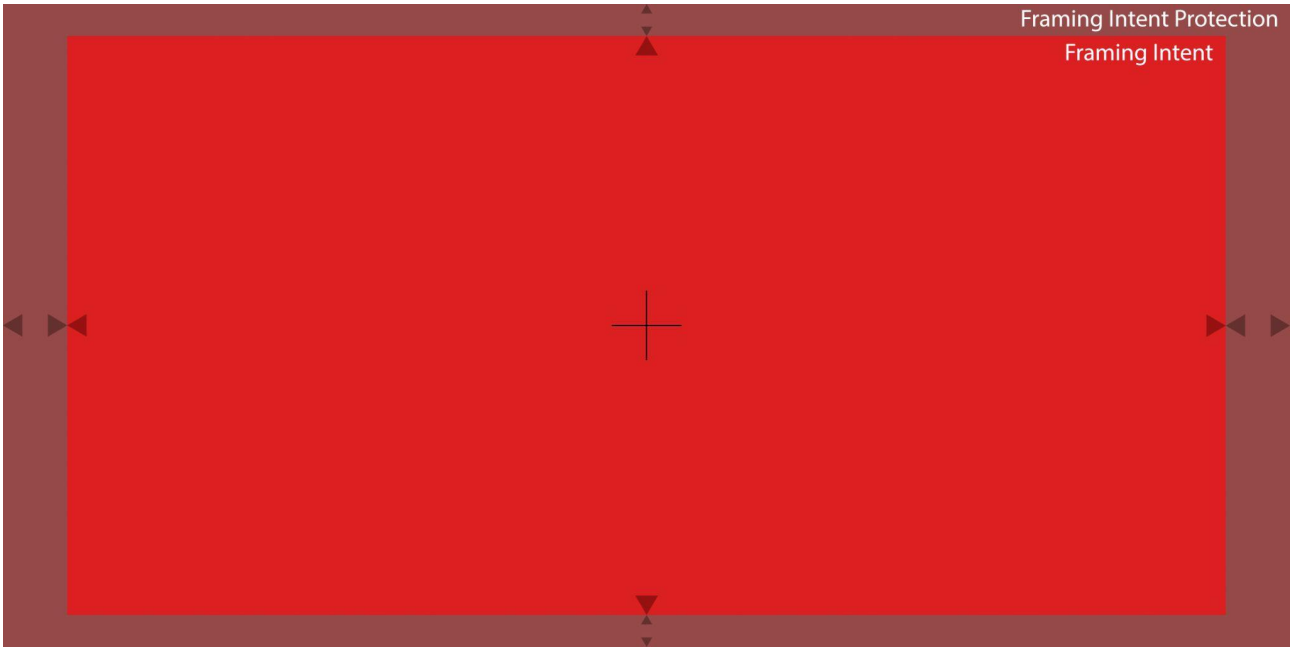
Data Type: "width": integer, "height": integer

Required Field: Yes

7.2.4 protection:

A framing intent may include an area for protection. This area is called Framing Intent Protection and matches the aspect ratio of the Framing Intent.

A protection value of 0 means there will be no Framing Intent Protection. A protection value of 0.05 results in a 5% protection area outside of the framing intent. In this case the Framing Intent Protection width would be 105% of the Framing Intent width and similarly for height.



The default value for the protection will be 0.

The protection aspect ratio will always match the aspect ratio of the associated Framing Intent.

Unless manually overridden by the user, implementations should default to center the protection to the associated Framing Decision.

As an example:
`"protection": 0.05`
Data Type: float (greater than or equal to 0)

Required Field: No

Default Value: 0

7.3 Contexts

```
"contexts": [{
    "label": "ArriLF OG",
    "context_creator": "ASC FDL Committee",
```

Description:

The Context class provides the ability for image authors to provide additional information on the origin of the ASC FDL. The field's purpose is for users to manage and organize their FDL data. As an example, an image author may choose to have framing data specific to a certain camera, editorial delivery, visual effects plates, etc, all separated by Contexts. Another production may choose to use the Context field to separate each camera manufacturer.

Child Elements:

7.3.1 `label`:

The label field is a user defined field available to categorize/manage FDL data. As an example: “ArriLF OG”.

Note: Using UTF-8 strings with non-ASCII characters may have compatibility issues on some software.

As an example:
`"label": "ArriLF OG",`
Data Type: string

Required Field: No

7.3.2 context_creator:

The context_creator attribute will be populated by the application that has generated the FDL. This field represents which user or implementation has generated this specific context within the ASC FDL. There could be different context_creator data in each context if multiple authors contributed to an ASC FDL. It is up to the implementation as to how it formats this attribute's values.

As an example:
`"context_creator": "ASC FDL Committee",`
Data Type: string

Required Field: No

7.4 Canvases

```
"canvases": [{
      "label": "Open Gate RAW",
      "id": "20220310",
      "source_canvas_id": "34256345",
      "dimensions": {"width": 4448, "height": 3096},
      "effective_dimensions": {"width": 4006, "height": 2788},
      "effective_anchor_point": {"x": 222, "y": 155},
      "photosite_dimensions": {"width": 4448, "height": 3096},
      "physical_dimensions": {"width": 36.7, "height": 25.54},
      "anamorphic_squeeze": 1.00,
```

Description:
The Canvas defines the active coordinate system of an application, file, or video stream. An application, file, or video stream could contain additional area outside of the defined Canvas, but applying an FDL to utilize that region would require a different Canvas. As an example, if a camera recorded a file, the file's recorded resolution will be utilized as the Canvas. If you then record a second file in a different resolution, you will have a new canvas. A Canvas can only be generated once the system creating it understands what the recorded/generated area is going to be.

Child Elements:

7.4.1 label:

The label field is a user defined field available to categorize/manage FDL Canvases. As an example: "Open gate RAW".

Note: Using UTF-8 strings with non-ASCII characters may have compatibility issues on some software.

As an example:
```
"label": "Open Gate RAW",
```
Data Type: string

Required Field: No

7.4.2 id:

The Canvas id field is meant to provide a means of identification for a Canvas. The id is unique to each Canvas inside of a given FDL, but may not be globally unique to other FDL files. The Canvas id field shall be between 1-32 characters in length and will allow for use of alphanumeric characters and underscores.

As an example:
```
"id": "20220310",
```

Data Type: string

Required Field: Yes

7.4.3 source_canvas_id:

ASC FDLs may be generated from original camera files, or derivatives from the original camera files. Therefore the source_canvas_id attribute has been created to allow a user to see the Canvas that was used when the original ASC FDL was created. As an example, a user may have an ASC FDL for the source camera file. They may then render plates to be delivered to a VFX vendor with a smaller resolution. Along with these plates, a new FDL may be delivered. The new ASC FDL's Canvas id will represent the new Canvas (VFX plates) that were generated. However the source_canvas_id would reference the initial Canvas id the new Canvas was generated from.

Any time an application creates a new FDL with a new Canvas, or appends an FDL with a new Canvas, it is expected that the original source Canvas be maintained. Therefore when a new FDL is created, both the new Canvas and the source Canvas should be present within the file.

If there is no prior knowledge of a past generation canvas, the source_canvas_id should be the same value as Canvas id.

As an example:
```
"source_canvas_id": "34256345",
```
Data Type: string

Required Field: Yes

7.4.4 dimensions:

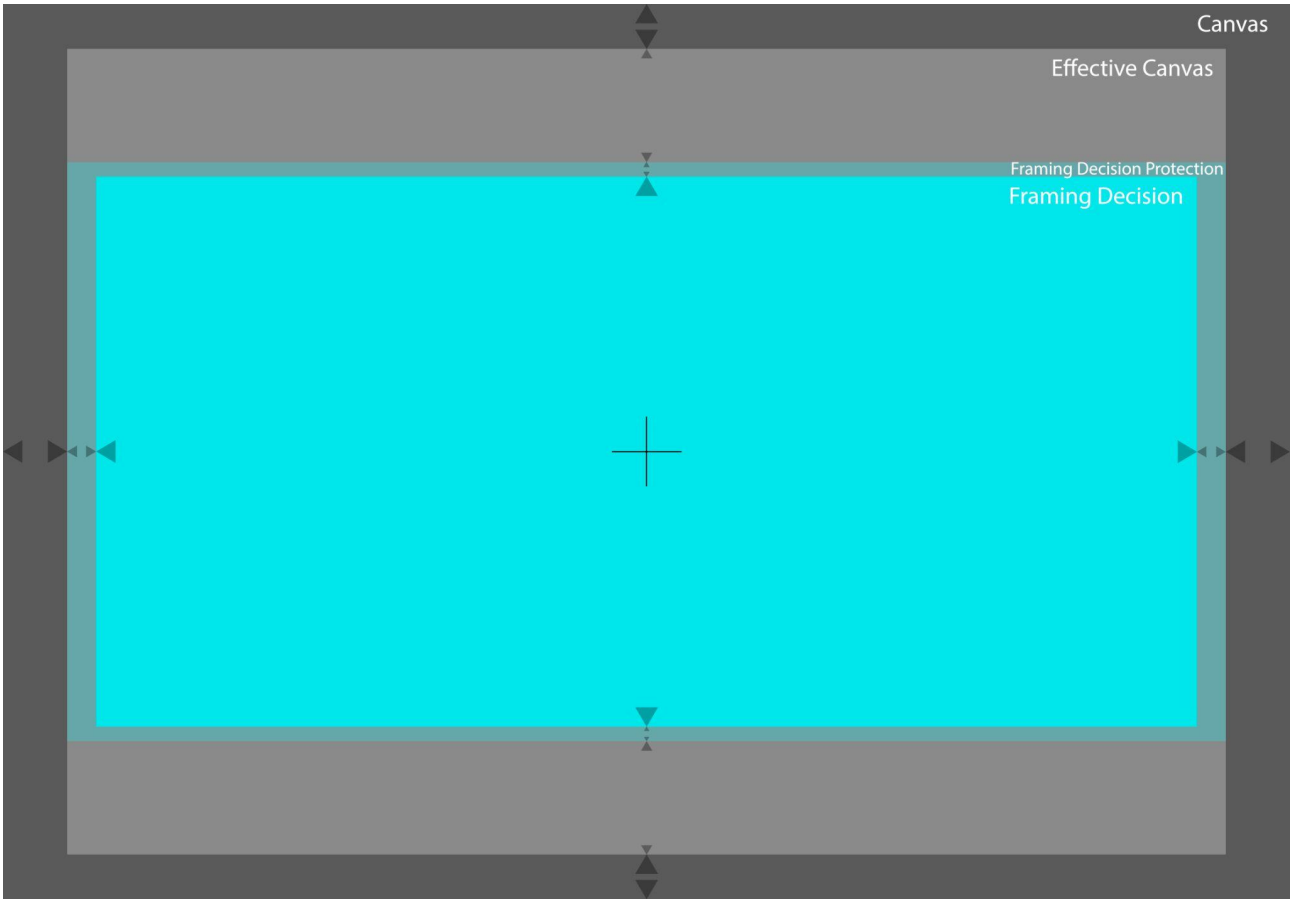
Any canvas within an ASC FDL will have a width and height defined. The dimensions field will be formatted as: "width": 4448, "height": 3096

As an example:
`"dimensions": {"width": 4448, "height": 3096},`
Data Type: "width": integer, "height": integer

Required Field: Yes

7.4.5 effective_dimensions:

A Canvas can be effectively constrained to prevent a Framing Intent and its Framing Intent Protection from being applied outside an intended area. This is called an Effective Canvas. As an example, when vignetting results from a lens that doesn't cover your camera's sensor, a user may choose to constrain the usable Canvas to the region not affected by vignetting. When the user applies the Framing Decision within, it will not have the vignetting in frame.



The effective_dimensions will define the width and height of this canvas constraint and will be written as:

"width": 4006, "height": 2788

As an example:
`"effective_dimensions": {"width": 4006, "height": 2788},`
Data Type: "width": integer, "height": integer

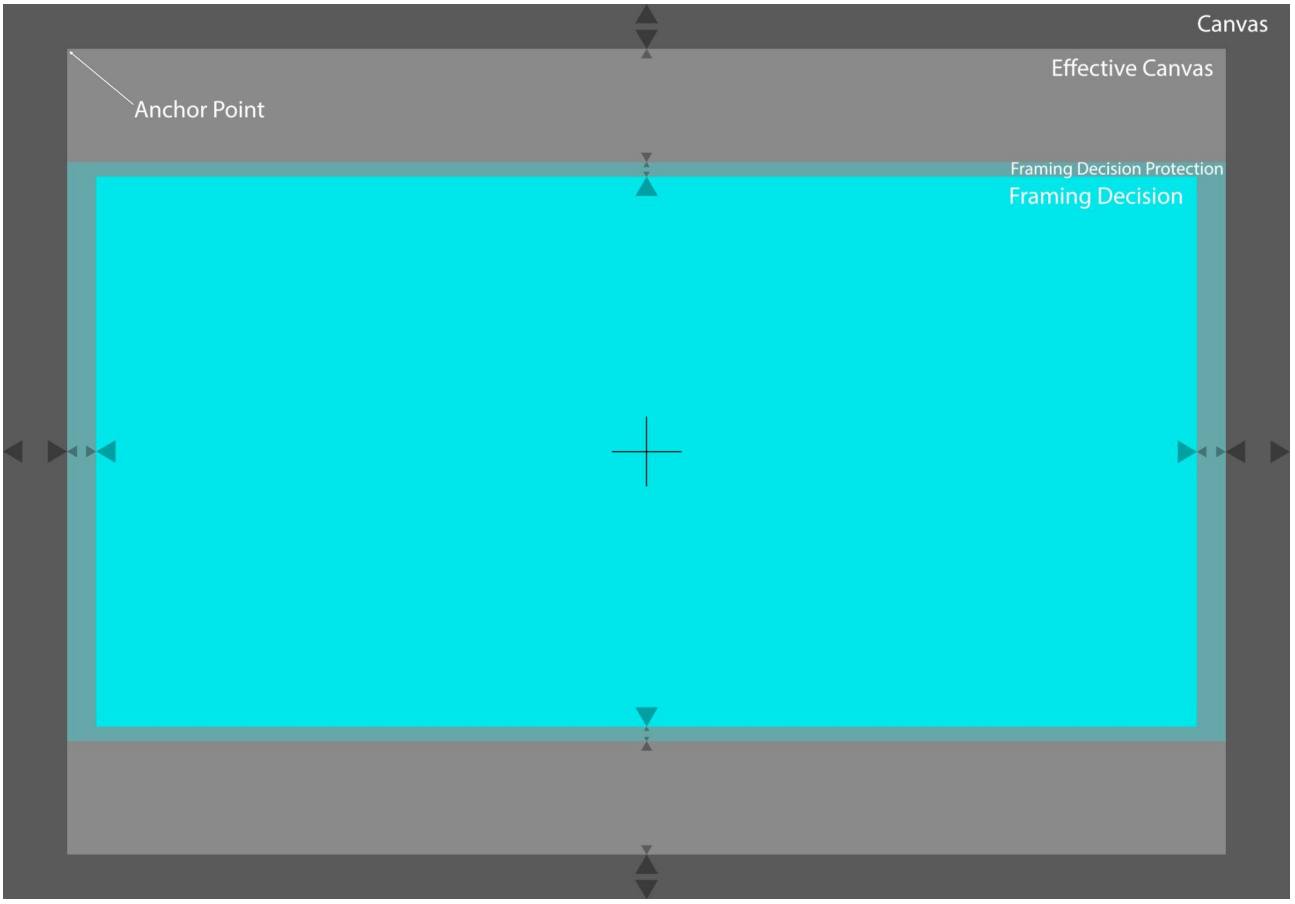
Units: Pixels

Required Field: No

7.4.6 `effective_anchor_point`:

If an Effective Canvas is to be placed within a Canvas, any implementation will require an understanding of the area to be used and where to position this area within the Canvas. The effective_anchor_point documents where the top left pixel of the Effective Canvas should be inside the Canvas. Similarly to the Framing Decision anchor_point, we're using float versus int values for these variables to avoid rounding issues when scaling.

The effective anchor point will be written as an example
"x": 222, "y": 155.1



As an example:

```
"effective_anchor_point": {"x": 222, "y": 155},
```

Data Type: "x": float, "y": float

Required Field: Yes if there is an effective_dimensions, otherwise No

Units: Pixels

7.4.7 photosite_dimensions:

We encourage camera manufacturers to provide this data when a camera has generated an ASC FDL Canvas. We do not require any implementation to generate photosite dimensions for non physical camera generated canvases. As an example, if a camera generated a Canvas it would ideally capture this attribute. However if an ASC FDL was generated with a new Canvas for a VFX Plate, this attribute would not be expected to be filled. Therefore, a source Canvas should have photosite_dimensions, but a subsequent child Canvas would likely not.

As an example;
Arri Alexa Mini - Recording Mode: 4K UHD

Sensor Active Image Area:
3200 x 1800 photosites
Recording File Image Content:
3840 x 2160 pixels

As an example:
`"photosite_dimensions": {"width": 4448, "height": 3096},`
Data Type: "width": integer, "height": integer

Required Field: No

Units: Photosites

Default Value: Omitted

7.4.8 physical_dimensions:

We encourage camera manufacturers to provide this data when a camera has generated an ASC FDL Canvas. We do not require any implementation to generate physical_dimensions for non physical camera generated canvases. As an example, if a camera generated a Canvas it would ideally capture this attribute. However if an ASC FDL was generated with a new Canvas for a VFX Plate, this attribute is not mandatory. Any implementation that is reading an FDL that originally contained physical_dimensions, could now generate new physical_dimension values.

Note: To ensure accuracy to one tenth of a micron users should provide 4 decimal places.

As an example:
`"physical_dimensions": {"width": 36.7, "height": 25.54},`
Data Type: "width": float, "height": float

Required Field: No

Unit Type: Millimeters

Default Value: Omitted

7.4.9 anamorphic_squeeze:

Any application reading an FDL will need to understand if the Canvas it is reading is squeezed or not. The anamorphic_squeeze attribute will match the image deformation numbering system that lens manufacturers use. As an example, 1.3 would mean the image is currently squeezed by a ratio of 1.3:1. Or 2.0 would indicate the image has been squeezed by a factor of 2:1. The squeeze is specifically a horizontal squeeze factor.

All applications reading an ASC FDL will apply the squeeze factor before any scaling, to ensure consistency between applications. This will be critical considering if you apply these in a different order you may get different results and we want to ensure consistency between any implementation.

Note: Users must ensure they provide enough decimal places. e.g. some lenses are 1.3 and some are 1.33, these are different values.

As an example:
`"anamorphic_squeeze": 1.00,`
Data Type: float (greater than 0)

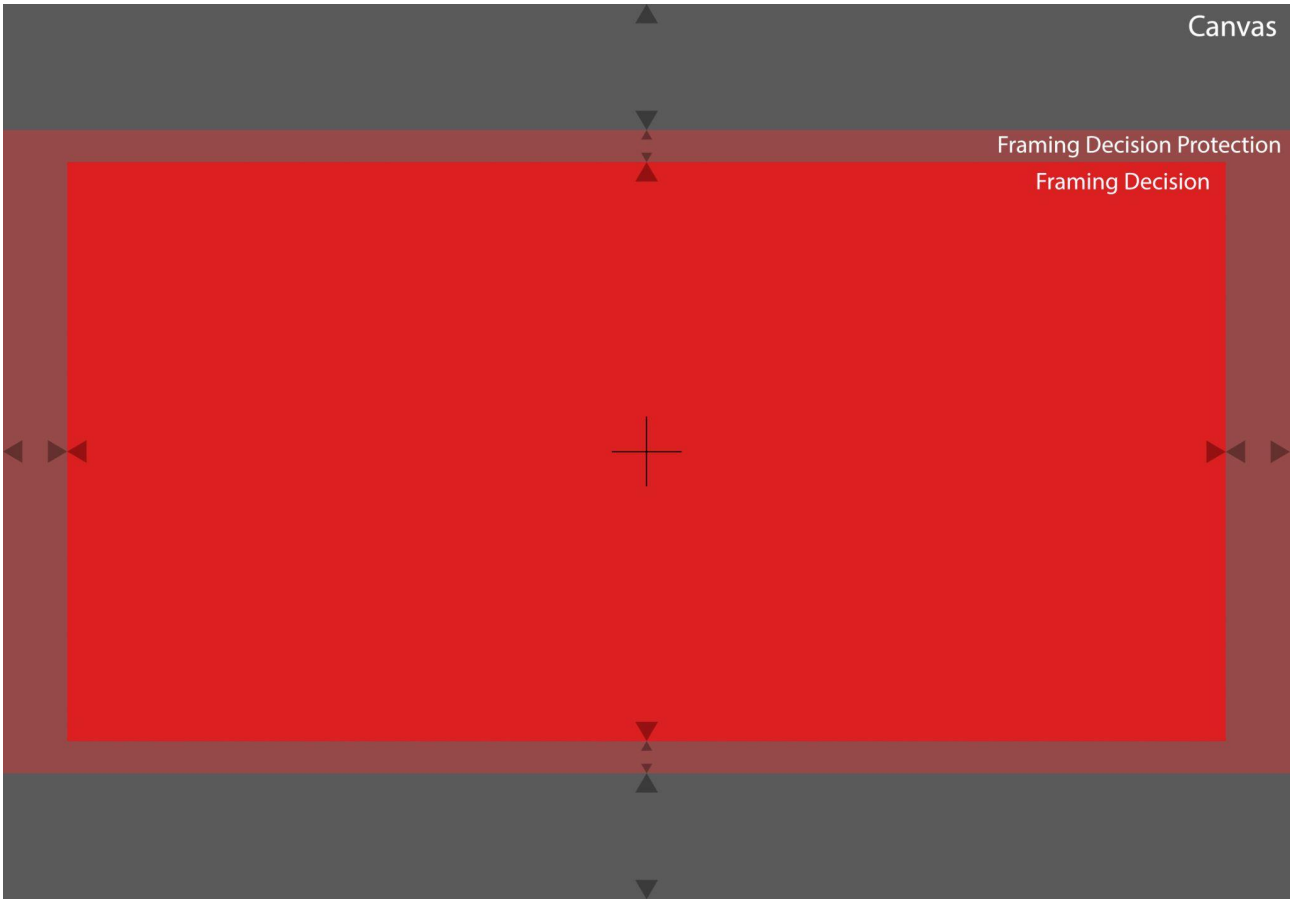
Required Field: No

Default Value: 1

7.5 Framing Decisions

```
"framing_decisions": [{
          "label": "2.39-1 Framing",
          "id": "20220310-FDLSMP01",
          "framing_intent_id": "FDLSMP01",
          "dimensions": {"width": 4004.1, "height": 2002},
          "anchor_point": {"x": 222.00, "y": 547.00},
          "protection_dimensions": {"width": 4448, "height": 2224},
          "protection_anchor_point": {"x": 0, "y": 436}
```

Description:
When the initial Framing Intent was created, it did not have any attributes that documented its position within a Canvas, nor anything that defined its actual size. It was just a defined aspect ratio. When a Framing Intent and its Framing Intent Protection are now applied to a Canvas, two decisions are produced: a Framing Decision and a Framing Decision Protection. The Framing Decision which is connected to a specific Canvas will have a defined set of coordinates, so any application reading an ASC FDL can understand where the intended frame resides.



Child Elements:

7.5.1 `label`:

The Framing Decision label is a user defined field available to categorize/manage FDL data. As an example: “2.39:1 Framing”.

Note: Using UTF-8 strings with non-ASCII characters may have compatibility issues on some software.

As an example:

```
"label": "2.39-1 Framing",
```

Data Type: string

Required Field: No

7.5.2 id:

Each Framing Decision will have its own id field that will be unique to the ASC FDL, but not universally unique among other ASC FDLs.

The formatting of the framing decision id will be:

[canvas>id] [-] [framing_intent>id]

As an example:
`"id": "20220310-FDLSMP01",`

Data Type: string

Required Field: Yes

7.5.3 framing_intent_id:

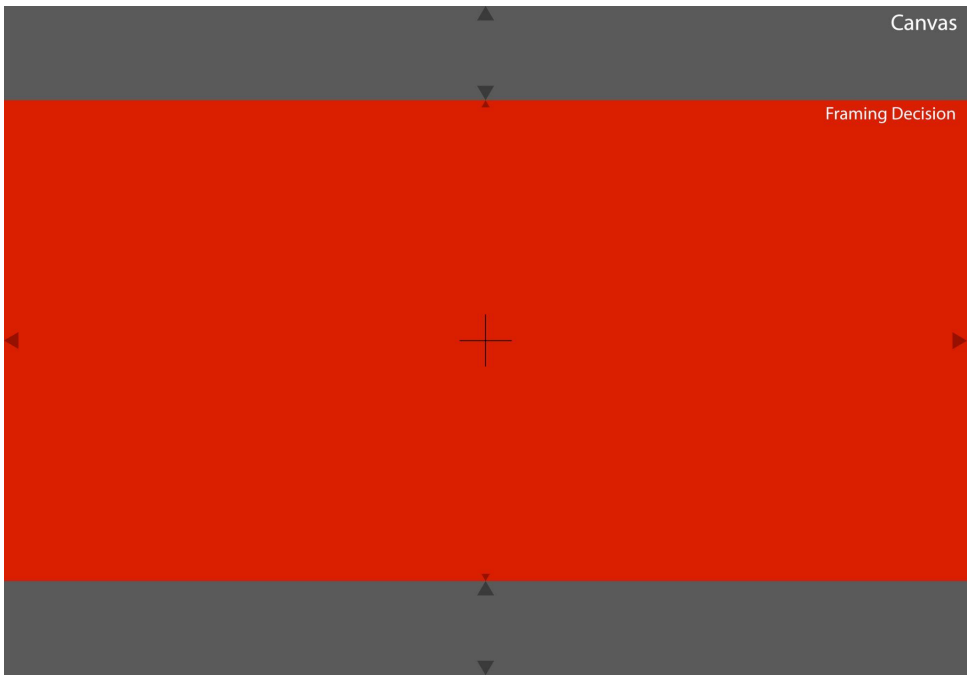
Including the framing_intent_id inside the framing_decision class is intended to allow for any implementation to infer which framing_intent this framing_decision is connected to.

As an example:
`"framing_intent_id": "FDLSMP01",`
Data Type: string

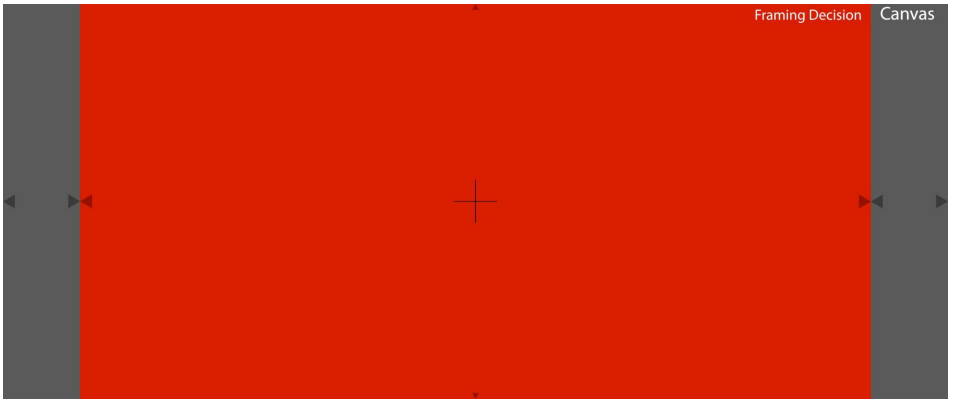
Required Field: Yes

7.5.4 dimensions:

The Framing Decision dimensions will specify the width and height of the Framing Decision now that it has been placed within a Canvas. When generating an ASC FDL, any implementation shall by default place the framing decision to fit within the Canvas, not cropping any of the resulting Framing Decision. As an example, if the Canvas had an aspect ratio of 1.43, and the Framing Intent was 2.0, the resulting Framing Decision would letterbox the Canvas:



Alternatively, if the Canvas had an aspect ratio of 2.39, it would be pillarboxed:



The dimensions shall be formatted as float number values, for higher scaling precision. When the float values need to become integers for display purposes in any implementation, we recommend the nearest integer value.

When a Canvas has been scaled to a new dimension, it is possible that the Framing Decision dimensions will not match the Framing Intent perfectly. When this new FDL is read by any implementation, the Framing Decision dimensions should take precedence.

As an example:

```
"dimensions": {"width": 4004.1, "height": 2002},
```

Data Type: "width": float, "height": float

Required Field: Yes

7.5.5 anchor_point:

If an application is given a dimension for the Framing Decision in pixels, it still needs to understand where to position this within the Canvas. The Framing Decision anchor_point specifies where the top left pixel of the Framing Decision is, in relation to the top left of the Canvas, or the Effective Canvas if one was used.

The anchor_point uses float versus int values to avoid scaling ambiguities and rounding issues when scaling.

As an example: "x": 222.00, "y": 547.00

The first value is the number of pixels horizontally from the left side of the Canvas, unless an Effective Canvas has been used. If an Effective Canvas has been used, the anchor_point would count from the left side of the Effective Canvas.

The second value uses the same process, but now for the y-axis.

As an example:
```
"anchor_point": {"x": 222.00, "y": 547.00},
```
Data Type: "x": float, "y": float

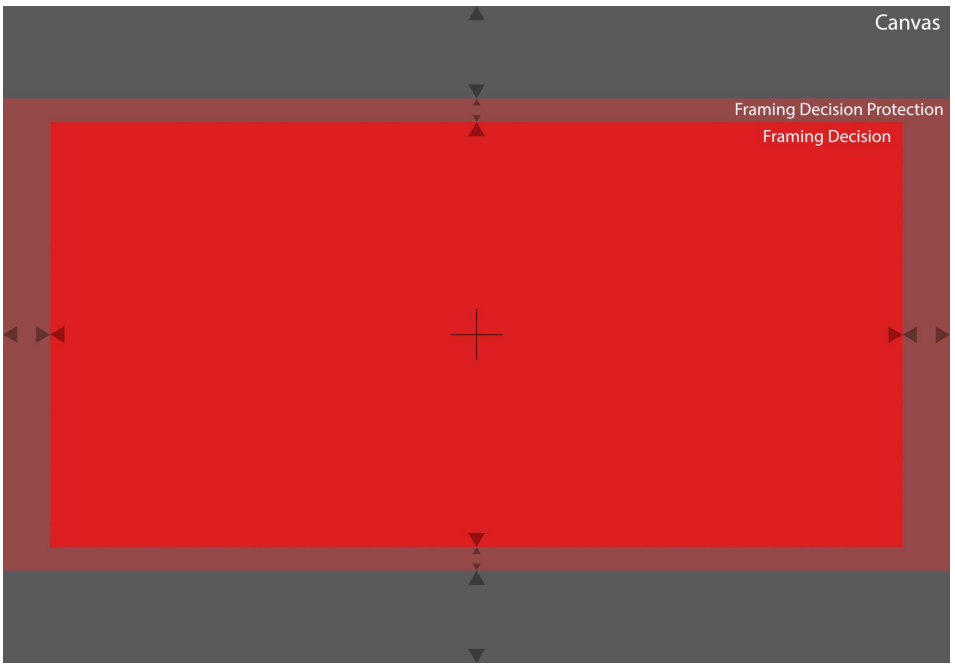
Required Field: Yes

Units: Pixels

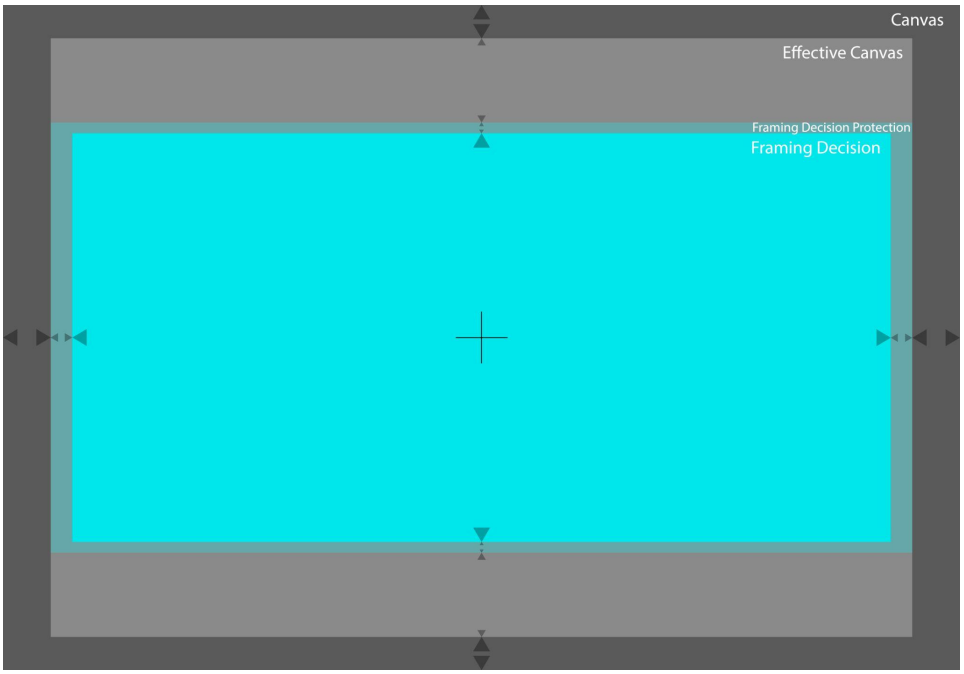
7.5.6 protection_dimensions:

Similarly to the Framing Intent resulting in a Framing Decision once placed within a Canvas, the Framing Intent Protection will result in a Framing Decision Protection once placed within a Canvas. This area may be utilized as a safety in post production to allow for slight reframing, stabilization and more.

If a Framing Decision had a protection, the protection would be fit into the Canvas:



If an Effective Canvas exists, the Framing Decision protection would be placed within it:



In the event that a user has chosen a Framing Decision dimensions that conflicts with an existing Framing Intent protection, it is permissible to override the Framing Intent protection to utilize the Framing Decision dimensions.

The Framing Decision protection_dimensions allows float number values for higher scaling precision.

The framing Framing Decision protection_dimensions can be omitted in the case where no protection is being added to the Framing Decision.

The formatting of this data will be: "width": float, "height": float

Note: When the float values need to become integers for display purposes in any implementation, we recommend the nearest integer value.

As an example:
`"protection_dimensions": {"width": 4448, "height": 2224},`
Data Type: "width": float, "height": float

Unit Type: Pixels

Required Field: Optional

7.5.7 protection_anchor_point:

If a protection_dimensions is utilized, any implementation will require an understanding of the area within the Canvas to be used. Even if an application is given a dimensions in pixels for this area, it still needs to understand where to position it within the Canvas. Therefore the Framing Decision protection_anchor_point specifies where the top left pixel of the Framing Decision protection is, in relation to the top left of the Canvas, or Effective Canvas if one was used.

The Framing Decision protection_anchor_point will use pixels as the unit type.

The protection_anchor_point originates from the top left of the Canvas.

The first value is the number of pixels horizontally from the left of the Canvas dimensions (or Effective Canvas dimensions if one was used) to the edge of the Framing Decision protection_dimensions (x-axis).

The second value is the number of pixels vertically from the top of the Canvas dimensions (or Effective Canvas dimensions if one was used) to the top of the Framing Decision protection_dimensions (y-axis).

The framing Framing Decision protection_anchor_point can be omitted in the case where no protection is being added to the Framing Decision.

As an example:
`"protection_anchor_point": {"x": 0, "y": 436}`
Data Type: "x": float, "y": float

Required Field: Optional

Unit Type: Pixels

7.6 Canvas Template

```
"canvas_templates": [{
      "label": "VFX Pull",
      "id": "VX220310",
      "target_dimensions": {"width": 3840, "height": 2160},
      "target_anamorphic_squeeze": 1.00,
      "fit_source": "framing_decision.dimensions",
      "fit_method": "width",
      "alignment_method_vertical": "center",
      "alignment_method_horizontal": "center",
      "preserve_from_source_canvas": "canvas.dimensions",
      "round": {"even": "even", "mode": "up"},
      "maximum_dimensions": {"width": 5000, "height": 3496},
      "pad_to_maximum": "true",
      "round": {"even": "whole", "mode": "up"}
```

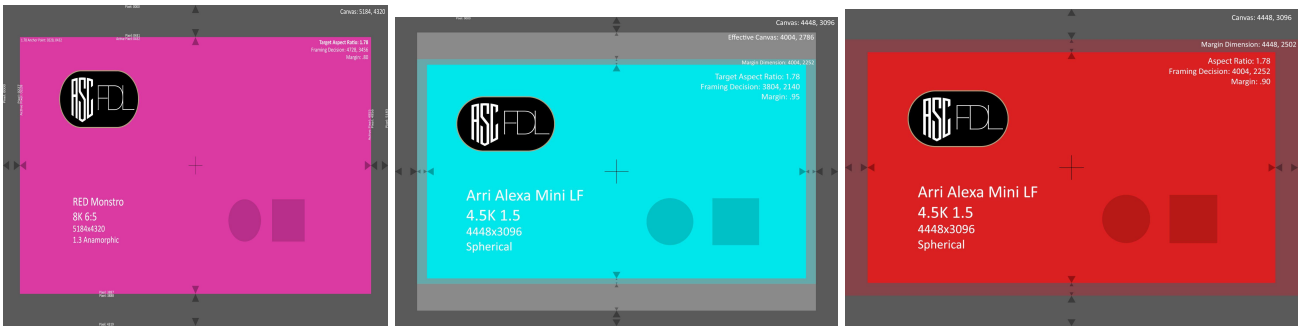
Description:

The Canvas Template is a class within an ASC FDL that provides the ability for users to normalize framing across all camera formats that may have been used during a production.

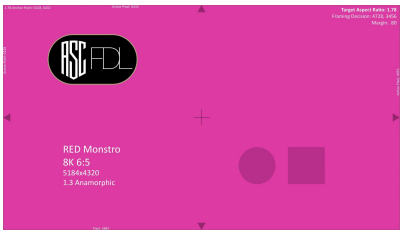
The Canvas Template provides a set of framing instructions that map a Canvas into a newly defined Canvas. As an example, if various cameras and resolutions were captured during a production, the VFX Supervisor or Picture Finishing Facility may want to ensure that all plates generated for VFX work are normalized into a common container before going to vendors. However they do not know all of the various camera formats that have been captured on set. The user could create a Canvas Template within an ASC FDL and provide instructions for any application to place the sources into this newly defined Canvas.

As an example:

Source 01 Source 02 Source 03



Result 01 Result 02 Result 03



Note: An ASC FDL file does not require a Canvas Template. Many workflows may never utilize this enhanced feature. However this functionality has been added to the specification for sophisticated jobs looking to normalize framing.

An ASC FDL with a Canvas Template may, or may not contain source FDL data. However, if an ASC FDL contains the source file's FDL data (Canvas, Framing Decision, etc) this data can be used for the implementation's decision making on normalizing framing.

For ASC FDL files that do not contain the source files Framing Decisions, implementations will need to look for ASC FDL data for each source file they intend to use the Canvas Template on.

7.6.1 Applying Canvas Templates

For a Canvas Template to be used by any application, the application would need to know which source FDL data (Canvas, Framing Decision) it should be utilizing. It is not within scope of this specification to mandate how an implementation will request a user to point the application to specific FDLs for this source data.

Child Elements:

7.6.2 `label`:

The label field is a user defined field available to categorize/manage FDL data. As an example: "VFX Pull".

Note: Using UTF-8 strings with non-ASCII characters may have compatibility issues on some software.

As an example:

```
"label": "VFX Pull",
```

Data Type: string

Required Field: No

7.6.3 `id`:

The Canvas Template id field is meant to provide a means of identification for a Canvas Template. This id is not universally unique, but no other Canvas Template id within a single FDL will use the same id. The Canvas Template id field shall be between 1-32 characters in length and will allow for use of alphanumeric characters and underscores.

The id data will be formatted as follows:
`"id": "value",`

As an example:
`"id": "VX220310",`

As an example:
`"id": "VX220310",`
Data Type: string

Required Field: Yes

7.6.4 `target_dimensions`:

For a Canvas Template to be utilized, the user will choose a specific set of dimensions. Using the fit_source attribute they can specify what specifically to fit into this target_dimensions. As an example, a user may choose a target dimension of: {"width": 3840, "height": 2160 } and a fit source of Framing Decision. That would mean the user wants to fit the framing decision into the target_dimensions of 3840 x 2160.

The target_dimensions data will be formatted as follows:

```
"target_dimensions": {"width": integer, "height": integer},
```

As an example:

```
"target_dimensions": {"width": 3840, "height": 2160},
```

For an ASC FDL to utilize a canvas template, this field must be populated. If it is left blank, any application reading the ASC FDL should flag this field as missing and the canvas template cannot be used.

As an example:
`"target_dimensions": {"width": 3840, "height": 2160},`
Data Type: "width": integer, "height": integer

Required Field: Yes

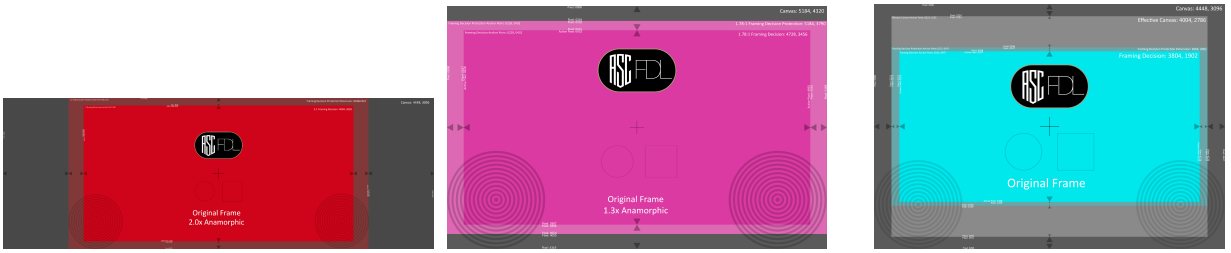
Unit Type: Pixels

7.6.5 `target_anamorphic_squeeze`:

The target_anamorphic_squeeze field represents the squeeze factor we are forcing on any incoming media when placed into this new Canvas. As an example, if 1.0 is written as the target_anamorphic_squeeze; that means the user wants everything to be normalized to a squeeze factor of 1.0. Therefore media that was originally captured squeezed at 2.0 would be horizontally stretched to 1.0. If other media was captured at 1.3, it would be horizontally stretched to become 1.0. If other media was shot spherically at 1.0, it would remain 1.0.

2.0:1 Source: 1.33:1 Source 1.0:1 source

Result: 1.0:1 Result: 1.0:1 Result: 1.0:1



If a value of 0 has been used within the target_anamorphic_squeeze field, all media should be left to their source squeeze factor defined in the source FDL's anamorphic_squeeze. This is similar to some applications using the terminology of Same As Source. Therefore, if an incoming acquisition was shot at 2.00 anamorphic, and the user set the target_anamorphic_squeeze to 0, it would remain 2.00.

The source's squeeze factor must be defined in order for the target_anamorphic_squeeze attribute to work. This is available as the anamorphic_squeeze in the source ASC FDL's Canvas.

Any application performing this process will need to apply the squeeze factor before any Fit operation (scaling). The order of operations will be: Desqueeze and then Scale, to ensure consistency between applications.

This field's formatting will follow the image deformation numbering system typically used by lens manufacturers. As an example, 1.3 or 2. The formatting will specifically be:

```
"target_anamorphic_squeeze": float
```

Note: Any user interface used to manually specify pixel aspects should support at least 5 decimal place precision, to ensure that unsqueezed images are likely to be correct to the nearest pixel.

As an example:
`"target_anamorphic_squeeze": 1.00,`
Data Type: float (greater than 0)

Required Field: Yes

7.6.6 `fit_source`:

After determining a target_dimensions, the user will need to choose which region to fit into the target_dimensions from the source files. Available options will be:

- framing_decision.dimensions
- framing_decision.protection_dimensions
- canvas.dimensions
- canvas.effective_dimensions

For the Canvas Template to successfully target specific regions to place into the target_dimensions, the incoming media will require associated ASC FDL data.

The fit source field will be formatted as follows:

```
"fit_source": "value",
```

As an example:
`"fit_source": "framing_decision.dimensions",`
Data Type: Enum

Required Field: Yes

Allowed Values:

- framing_decision.dimensions

- framing_decision.protection_dimensions
- canvas.dimensions
- canvas.effective_dimensions

7.6.7 `fit_method`:

The fit_method attribute specifies how to fit the fit_source selection into the target_dimensions.

Any implementation shall apply the squeeze factor before any Fit (scaling). Therefore the order of operations shall be: Desqueeze > Fit/Scale

The fit method field will be formatted as follows:

```
"fit_method": "value",
```

Available Attributes: Width, Height, Fit All
width
The selected fit_source attribute's width should be fit into the target_dimensions regardless of whether the height is cropped, or doesn't fill the target_dimensions height.

height
The selected fit_source attribute's height should be fit into the target dimension regardless of whether the width is cropped, or doesn't fill the target_dimensions width.

fit_all
The selected fit_source attribute should be fit into the target_dimensions, not cropping anything from either the height or width.

fill
The selected fit_source attribute should fill the entire target_dimensions, therefore potentially cropping the height or width of the selected fit_source.

As an example:
`"fit_method": "width",`
Data Type: Enum

Required Field: Yes

Allowed Values: width, height, fit_all, fill

7.6.8 `alignment_method_vertical`:

The alignment method allows users to choose to offset their frame vertically, or horizontally. The alignment_method_vertical, allows the ability to specifically alter the position of the source vertically.

The alignment method vertical field will be formatted as follows:

```
"alignment_method_vertical": "value",
```

As an example:

```
"alignment_method_vertical": "center",
```

As an example:
`"alignment_method_vertical": "center",`
Data Type: Enum

Required Field: No

Allowed Values: top, center, bottom

Default Value: center

7.6.9 `alignment_method_horizontal`:

The alignment method allows users to choose to offset their frame vertically, or horizontally. The alignment_method_horizontal, allows the ability to specifically alter the position of the source horizontally.

The alignment method horizontal field will be formatted as follows:

```
"alignment_method_horizontal": "value",
```

As an example:
`"alignment_method_horizontal": "center",`
Data Type: Enum

Required Field: No

Allowed Values: left, center, right

Default Value: center

7.6.10 `preserve_from_source_canvas`:

Once a region from the source has been chosen to fit into the target_dimensions, the user may choose to preserve a larger area outside of the chosen fit_source. Here is an example source image we'll be working with:

In the example above, the source Canvas was 4448 x 3096 and the framing decision within was 4004 x 2502. Using the Canvas Template, the user may choose to fit the Framing Decision into the Target Dimension of 3840 x 2160. The user then has the option to either preserve the area outside of this Framing Decision, or cut it off. Here is an example if the user had chosen framing_decision.dimensions for preserve_from_source_canvas:

Therefore nothing is maintained outside of the Framing Decision and the Canvas remains 3840 x 2160. If the user had chosen canvas.dimensions for preserve_from_source_canvas, the resulting image would have expanded outward within a new Canvas dimensions of 4262 x 2968.

This field will be formatted as follows:

```
"preserve_from_source_canvas": "value",
```

As an example:
`"preserve_from_source_canvas": "canvas.dimensions",`
Data Type: Enum

Required Field: No

Allowed Values:

- none
- framing_decision.dimensions
- framing_decision.protection_dimensions
- canvas.dimensions
- canvas.effective_dimensions

Default Value: none

7.6.11 `maximum_dimensions`:

The maximum_dimensions attribute will determine any newly generated Canvas' maximum dimensions. In the example used above for the preserve_from_source attribute, if a user had chosen to preserve the Canvas, but also set the maximum_dimensions to 4096 x 2160, the resulting Canvas would have been cropped (not scaled) resulting in a 4096 x 2160.

However, if the user had not defined `maximum_dimensions`, the resulting image would not have been cropped.

If the user as an example chose a maximum dimension of 5000 x 3496, the resulting image would still remain 4262 x 2968, as this attribute does not force any kind of scaling.

maximum_ dimensions will always take priority over the preserve_from_source results.

maximum_ dimensions does not have any requirements on the aspect ratio utilized.

The formatting for this attribute is as follows:

```
"maximum_dimensions": {"width": integer, "height": integer}
```

As an example:
`"maximum_dimensions": {"width": 5000, "height": 3496},`
Data Type: "width": integer, "height": integer

Required Field: No

Default Value: Omitted

7.6.12 `pad_to_maximum`:

As specified in section 8.6.12, the maximum_dimensions field only ensures that a Canvas is no larger than a specified dimension. Therefore, resulting Canvases from different inputs could vary in dimensions if they do not exceed the maximum_dimensions. The pad_to_maximum forces resulting Canvases to the specified maximum_dimensions. However it does not scale the image. Instead it adds pillarboxing, or letterboxing if needed.

As an example:

Here is our source, which has a Canvas of 4448 x 3096

The user has chosen a fit_method of framing_decision.dimensions and a dimensions of 3840 x 2160, and utilized a maximum_dimensions of 5000 x 3496. With the pad_to_maximum field set to false, this would have resulted in a Canvas of 4262 x 2968. i.e.; not forcing the Canvas to be the maximum_dimensions.

If the pad_to_maximum was set to true, this would have resulted in a new Canvas that is 5000 x 3496, with the resulting image both pillarboxed and letterboxed, considering the pad_to_maximum field does not force any scaling:

If the maximum_dimensions attribute is blank, the pad_to_maximum attribute will be disregarded.

Formatting for this attribute will be as follows:

```
"pad_to_maximum": value,
```

As an example:
`"pad_to_maximum": "true",`
Data Type: boolean

Required Field: No

Default Value: false

7.6.13 `round`:

Users may want to control whether a newly created Canvas' dimensions are allowed to contain odd number values. As an example, many users may choose to force any newly created canvas to be an even number of pixels. Different platforms handle rounding in different ways, so defining the rounding "rules" ensures consistency in scaling behavior between platforms. Rounding is one of the key variables that can be defined within a template to ensure consistent results.

The round field is only applicable to the final resulting Canvas dimensions and should not affect the fit_source or target_dimensions.

whole = to nearest integer
even = to nearest even-numbered integer

up = always round up
down = always round down

round = follow standard rounding rules (round up for values greater than or equal to +0.5, down for values less than +0.5)

In the case of resulting canvas_dimensions defined in a template (if pad_to_maximum is true) then rounding isn't applicable because that dimension must be filled.

If a resulting Canvas needs to have rounding applied, the Canvas should never be stretched, squished, or scaled. Instead padding, or cropping should occur.

The formatting for this field will be as follows:

```
"round": {"even": "value", "mode": "value"}
```

As an example:
`"round": {"even": "whole", "mode": "up"}`
Data Type: "even": Enum, "mode": Enum

Allowed Values for 1st Enum ('even'): whole, even

Allowed Values for 2nd Enum (‘mode’): up, down, round

Required Field: No

Default Value: "even": "even", "mode": "up"

8.0 Appendix

Appendix A: Example ASC FDL


```
{
  "uuid": "BCD142EB-3BAA-4EA8-ADD8-A46AE8DC4D97",
  "version": {"major": 0, "minor": 1},
  "fdl_creator": "ASC FDL Committee",
  "default_framing_intent": "FDLSMP01",
  "framing_intents": [{
      "label": "2-1 Framing",
      "id": "FDLSMP01",
      "aspect_ratio": {"width": 2, "height": 1},
      "protection": 0.1
    }
  ],
  "contexts": [{
    "label": "ArriLF",
    "context_creator": "ASC FDL Committee",
    "canvases": [{
      "label": "Open Gate RAW",
      "id": "20220310",
      "source_canvas_id": "20220310",
      "dimensions": {"width": 4448, "height": 3096},
      "effective_dimensions": {"width": 4448, "height": 3096},
      "effective_anchor_point": {"x": 0, "y": 0},
      "photosite_dimensions": {"width": 4448, "height": 3096},
      "physical_dimensions": {"width": 36.7, "height": 25.54},
      "anamorphic_squeeze": 1,
      "framing_decisions": [{
          "label": "2-1 Framing",
          "id": "20220310-FDLSMP01",
          "framing_intent_id": "FDLSMP01",
          "dimensions": {"width": 4004, "height": 2002},
```

```
          "anchor_point": {"x": 222, "y": 547},
          "protection_dimensions": {"width": 4448, "height": 2224},
          "protection_anchor_point": {"x": 0, "y": 436}
        }
      ]
    }]
  }]
}
```

Appendix B:
Example ASC FDL JSON File With 2 Framing Decisions

```
{
  "uuid": "DEAF6B84-6B8C-46DB-8CE3-DA5DAB8C9817",
  "version": {"major": 0, "minor": 1},
  "fdl_creator": "Jane Doe",
  "default_framing_intent": "29A901F1",
  "framing_intents": [{
      "label": "Hero 1.78",
      "id": "29A901F1",
      "aspect_ratio": {"width": 16, "height": 9},
      "protection": 0.05
    },
    {
      "label": "Hero 2-1",
      "id": "0302684B",
      "aspect_ratio": {"width": 2, "height": 1},
      "protection": 0.05
    }
  ],
  "contexts": [{
    "label": "ArriLF",
    "context_creator": "Arri LF",
    "canvases": [{
      "label": "Open Gate ARRIRAW",
      "id": "20210902",
      "source_canvas_id": "20210902",
      "dimensions": {"width": 4448, "height": 3096},
      "effective_dimensions": {"width": 4448, "height": 3096},
```

```
      "effective_anchor_point": {"x": 0, "y": 0},
      "photosite_dimensions": {"width": 4448, "height": 3096},
      "physical_dimensions": {"width": 36.7, "height": 25.54},
      "anamorphic_squeeze": 1,
      "framing_decisions": [{
          "label": "Hero 1.78",
          "id": "20210902-29A901F1",
          "framing_intent_id": "29A901F1",
          "dimensions": {"width": 4226, "height": 2376},
          "anchor_point": {"x": 111, "y": 360},
          "protection_dimensions": {"width": 4448, "height": 2508},
          "protection_anchor_point": {"x": 0, "y": 294}
        },
        {
          "label": "Hero 2-1",
          "id": "20210902-0302684B",
          "framing_intent_id": "0302684B",
          "dimensions": {"width": 4224, "height": 2112},
          "anchor_point": {"x": 112, "y": 492},
          "protection_dimensions": {"width": 4448, "height": 2224},
          "protection_anchor_point": {"x": 0, "y": 436}
        }
      ]
    }]
  }]
}
```

Appendix C:

Example ASC FDL JSON File With only the Canvas Template

```
{
  "uuid": "3E9F94EF-A910-470D-8EC4-B14E551AC6AB",
  "version": {"major": 0, "minor": 1},
  "fdl_creator": "The Camera",
  "default_framing_intent": "FDLSMP05",
  "framing_intents": [{
      "label": "2.39 Framing",
      "id": "FDLSMP05",
      "aspect_ratio": {"width": 2048, "height": 858},
      "protection": 0.1
    }
  ],
  "canvas_templates": [{
      "label": "VFX Pull",
```

```
      "id": "VX220310",
      "target_dimensions": {"width": 4096, "height": 1716},
      "target_anamorphic_squeeze": 1,
      "fit_source": "framing_decision.dimensions",
      "fit_method": "width",
      "alignment_method_vertical": "center",
      "alignment_method_horizontal": "center",
      "preserve_from_source_canvas": "canvas.dimensions",
      "round": {"even": "even", "mode": "up"}
  },
  {
      "label": "Editorial Dailies",
      "id": "ED220310",
      "target_dimensions": {"width": 1920, "height": 1080},
      "target_anamorphic_squeeze": 1.00,
      "fit_source": "framing_decision.dimensions",
      "fit_method": "width",
      "alignment_method_vertical": "center",
      "alignment_method_horizontal": "center",
      "preserve_from_source_canvas": "framing_decision.dimensions",
      "round": {"even": "even", "mode": "up"}
  }]
}
```

Appendix D:
Example ASC FDL JSON File With Effective Canvas

```
{
  "uuid": "5EDD03DC-4EFF-42BB-8085-DDECC3036982",
  "version": {"major": 0, "minor": 1},
  "fdl_creator": "ASC FDL Committee",
  "default_framing_intent": "FDLSMP04",
  "framing_intents": [{
      "label": "2-1 Framing",
      "id": "FDLSMP04",
      "aspect_ratio": {"width": 2, "height": 1},
      "protection": 0.05
    }
  ],
```

```
  "contexts": [{
    "label": "ArriLFV",
    "context_creator": "ASC FDL Committee",
    "canvases": [{
      "label": "Open Gate Vignette",
      "id": "20220311",
      "source_canvas_id": "20220311",
      "dimensions": {"width": 4448, "height": 3096},
      "effective_dimensions": {"width": 4004, "height": 2786},
      "effective_anchor_point": {"x": 222, "y": 155},
      "photosite_dimensions": {"width": 4448, "height": 3096},
      "physical_dimensions": {"width": 36.7, "height": 25.54},
      "anamorphic_squeeze": 1,
      "framing_decisions": [{
          "label": "2-1 Framing",
          "id": "20220311-FDLSMP04",
          "framing_intent_id": "FDLSMP04",
          "dimensions": {"width": 3804, "height": 1902},
          "anchor_point": {"x": 0, "y": 392},
          "protection_dimensions": {"width": 4004, "height": 2002},
          "protection_anchor_point": {"x": 100, "y": 442}
        }
      ]
    }]
  }]
}
```


Appendix E:
ASC FDL Represented in an ALE

A user may choose to communicate which ASC FDL should be applied to which shot through a metadata exchange file such as an ALE. There are two columns that should be provided in an ALE in order for any application to know which FDL to apply to any given shot:

- fdl-uuid
- fdl-framing-decision-id

The uuid field will be able to tell any application down stream which ASC FDL file to use for that particular shot. Whereas the framing decision id will tell the application which specific frame within the FDL to use.

Appendix F:
ASC FDL Data Placed within the header of a file:

A user may choose to communicate which ASC FDL should be applied to which shot by adding the necessary FDL info to the header of a rendered file. There are two key pieces of data that should be shared in order for any application to know which FDL to apply to any given shot:

- fdl-uuid
- fdl-framing-decision-id

The uuid field will be able to tell any application down stream which ASC FDL file to use for that particular shot. Whereas the framing decision id, will tell the application which specific frame within the FDL to use.

Appendix G:
ASC FDL Represented in an EDL

A user may choose to communicate which ASC FDL should be applied to which shot by adding the necessary FDL info to the comments section of an EDL. There are two key pieces of data that should be shared in order for any application to know which FDL to apply to any given shot:

- fdl-uuid
- fdl-framing-decision-id

The uuid field will be able to tell any application down stream which ASC FDL file to use for that particular shot. Whereas the framing decision id will tell the application which specific frame within the FDL to use.

The formatting of these values should be as follows:

* FDL-UUID: value
* FDL-FRAMING-DECISION-ID: value

TRADEMARK
ASC FDL™ is a trademark of American Society of Cinematographers.

DISCLAIMER
American Society of Cinematographers is not liable for any damages, including direct or consequential, from the use of ASC FDL specification outlined in this document.

LICENSING
This specification is made available under MIT License stated here - https://github.com/git/git-scm.com/blob/main/MIT-LICENSE.txt

NOTICE
For any further explanation of the contents of this document, or in case of any perceived inconsistency or ambiguity of interpretation, please contact American Society of Cinematographers at: ascfdl.feedback@gmail.com