Constraints (2014)

Change constraints to use WebIDL again.
Preserve baseline functionality.

Fix syntax, keep semantics (as much as possible)

Jan-Ivar Bruaroey - w3c - 3/26/14

8 slides + appendices

Dictionaries: Everything is optional

List preferences.

```
var constraints = {
  video: {
    width: { min: 640, max: 1280 },
    height: { min: 480, max: 768 },
    aspectRatio: 16/9,
    FrameRate: 60,
  }
};
nav.getUserMedia(constraints, success, fail);
```

UA picks best match.

Main feature: require

List names of individual constraints we require.

```
var constraints = {
  video: {
    require: ["width", "height"],
    width: { min: 640, max: 1280 },
    height: { min: 480, max: 768 },
    aspectRatio: 16/9,
    FrameRate: 60
  }
};
nav.getUserMedia(constraints, success, fail);
```

Same as mandatory

Infinity or die

UA sees unknown requirements:

```
var constraints = {
  video: {
    require: ["focus"],
    width: { min: 640, max: 1280 },
    height: { min: 480, max: 768 },
    aspectRatio: 16/9,
    focus: Infinity
  }
};
nav.getUserMedia(constraints, success, fail);
```

Fails (i.e. works!)

That's the main proposal

Two extensions add complexity.

Extension 1: prefer

List preferred order of constraints.

```
var constraints = {
  video: {
    require: ["width", "height"],
    prefer: ["aspectRatio", "frameRate"],
    width: { min: 640, max: 1280 },
    height: { min: 480, max: 768 },
    aspectRatio: 16/9,
    FrameRate: 60
  }
};
nav.getUserMedia(constraints, success, fail);
```

Same as optional (*)

Extension 2: ideal

Specify a target value within acceptable range.

```
var constraints = {
  video: {
    require: ["width", "height"],
    prefer: ["aspectRatio", "frameRate"],
    width: { min: 640, max: 1280, ideal: 1280 },
    height: { min: 480, max: 768, ideal: 768 },
    aspectRatio: 16/9,
    FrameRate: 60
  }
};
nav.getUserMedia(constraints, succ, fail);
```

E.g. I prefer higher resolutions

Benefits

<u>Users:</u>

Better defaults

More ideal expression

Simpler

<u>Implementers:</u>

Specific, not general

Standard WebIDL (types over prose)

Abstraction concept, not abstract interface

100% safe (pass by value)

Appendix 1: Comparison

Before

```
var ctr = { video: true, audio: true };

var constraints = {
  video: {
    mandatory: {
      width: { min: 640, max: 1280 },
      height: { min: 480, max: 768 }
    },
    optional: [
      { aspectRatio: 16/9 },
      { frameRate: 60 },
      { width: 1280 },
      { width: { min: 1024 } },
      { width: { min: 854 } },
      { width: { min: 800 } },
      { width: { min: 768 } },
      { height: { min: 720 } },
      { height: { min: 576 } }
    ]
  },
  audio: {
    mandatory: { sampleRate: 44000 },
    optional: [{ volume: 1.0 }]
  }
};

{{{{}{}}[{}{}{}{{}}{{}}{{}}{{}}{{}}]}{{}[{}]}}
```

After

```
var ctr = { video: true, audio: true };

var constraints = {
  video: {
    require: ["width", "height"],
    prefer: ["aspectRatio", "frameRate"],
    width: { min: 640, max: 1280, ideal: 1280 },
    height: { min: 480, max: 768, ideal: 768 },
    aspectRatio: 16/9,
    FrameRate: 60
  },
  audio: {
    require: ["sampleRate"],
    sampleRate: 44000,
    Volume: 1.0
  }
};

{{[][]{}{}}{[]}}
```

Appendix 2: Requirements (Jim B.)

	Requirement	Importance	Satisfied
1.	Clear distinction between mandatory and optional, namely all mandatory constraints must be satisfied, or it fails. Optional constraints may or may not be satisfied.	MUST	Yes
2.	Unknown/unsupported mandatory constraints must fail.	MUST	Yes
3.	Possible to implement back-off using optional constraints.	MUST	Mostly
4.	Possible to couple optional constraints, for example: I prefer an aspectRatio of 15/6 and width of 500. If I can't have both of those then give me aspectRatio of 4/3 and width of 400, etc. (DECEMBER 2013)	SHOULD	No
5.	The application must be notified if changing circumstances result in previously satisfied mandatory constraints becoming unsatisfied. (CONTROVERSIAL)	MUST	Yes

Appendix 3: WebIDL

```
dictionary MediaStreamConstraints {
    (boolean or MediaTrackConstraints) video = false;
    (boolean or MediaTrackConstraints) audio = false;
    DOMString peerIdentity;
};

dictionary Constraints { sequence<DOMString> require; sequence<DOMString> prefer; };
```

```
dictionary MediaTrackConstraints : Constraints {
    ConstrainLong width;
    ConstrainLong height;
    ConstrainDouble aspectRatio;
    ConstrainDouble frameRate;
    ConstrainVideoFacingMode facingMode;
    ConstrainDouble volume;
    ConstrainLong sampleRate;
    ConstrainLong sampleSize;
    boolean echoCancelation;
    ConstrainDOMString sourceId;

    // Extended through gUM-specific IANA registry.
};
```

```
dictionary ConstrainLongRange {
    long min;
    long max;
    long ideal;
};

dictionary ConstrainDoubleRange {
    double min;
    double max;
    double ideal;
};
```

```
typedef (long or ConstrainLongRange) ConstrainLong;
typedef (double or ConstrainDoubleRange) ConstrainDouble;
typedef (VideoFacingModeEnum or sequence<VideoFacingModeEnum>) ConstrainVideoFacingMode;
typedef (DOMString or sequence<DOMString>) ConstrainDOMString;
```

MediaStreamTrack

```
interface MediaStreamTrack : EventTarget {
    MediaTrackConstraints getCapabilities();
    MediaTrackConstraints getConstraints();
    MediaTrackConstraints getSettings();
    void applyConstraints(MediaTrackConstraints constraints,
                          VoidFunction successCallback,
                          ConstraintErrorCallback errorCallback);
             attribute EventHandler onoverconstrained;

    readonly attribute DOMString             kind;
    readonly attribute DOMString             id;
    readonly attribute DOMString             label;
             attribute boolean               enabled;
    readonly attribute boolean               muted;
             attribute EventHandler          onmute;
             attribute EventHandler          onunmute;
    readonly attribute boolean               _readonly;
    readonly attribute boolean               remote;
    readonly attribute MediaStreamTrackState readyState;
             attribute EventHandler          onstarted;
             attribute EventHandler          onended;
    MediaStreamTrack clone();
    void             stop();
};
```

Tip:

```
  var nativeHz =
m.getCapabilities().
  frameRate.ideal;
```