Possible Constraint Syntax Compromise

Constraints to make (almost) everyone happy

Problem

We can’t seem to agree.
We’re stuck.

So, I asked around to lots of people.
And found the real source of pain/conflict is...

The “required” hack

```
getUserMedia({“video”: {
  // If not supported, you get an error
  “required”: ["width", "depth", “facingMode”],
  “width”: {"min": 320, "max":1920},
  “height”: {"min":240, "max":1080},
  “depth”: ...
}}, ...);
```

The “required” hack…. why again?

WebIDL + “must fail” == need a hack

Short story: WebIDL hides keys unknown to the browser, which means the browser doesn’t know about them, which means it can’t reject them. For example, future constraint “`depth`”.

The options available

1. Toss WebIDL
2. Add a hack (“required”: [...])
3. Let JS check what’s supported (relax “must fail” a bit)

#1, no one really wants.

#2 is how we got where we are.

So how about door #3?

Before

```
getUserMedia({“video”: {
  // If "depth" not supported, error
  “required”: ["depth"],
  “width”: {"min": 320, ...},
  “height”: {"min": 240, ...},
  “depth”: ...
}}, ...);
```

1. Remove `required: [“depth”]`

After

```
if(!MST.getSupportedConstraints()["depth"]) {
  // Treat like an error.
}
getUserMedia({“video”: {
  “width”: {"min": 320, ...},
  “height”: {"min": 240, ...},
  “depth”: {..., "required": true}
}}, ...);
```

2. Add `MST.getSupportedConstraints.`
3. Move “required” into each constraint.

(Oh, and “ideal” and “advanced” are orthogonal)

Pros

- No more hack (prettier!)
- We can (hopefully) keep progressing.
- We don't have to deal with `required: ["depth"]` when `"depth":` isn't present.

Cons

- If the JS forgets to check, the required unsupported constraints are ignored. The JS does not get an error callback.

Consensus Call

Everyone happy?

Building on a Baseline Consensus™

Now that we have a Baseline Consensus™ we're all happy with, let's see if we can do one step better...

(and if we fail, we fall back to the Baseline Consensus™)

We can get rid of “required: true”

Before

```
getUserMedia({“video”: {
  // Ask for 640x480, get back 320x240, Huh?
  // Oh, you forgot the “do what I said” flag!
  “width”: 640,
  “height”: 480
}}, ...);

getUserMedia({“video”: {
  “facingMode”: {“value”: “environment”,
                 “required”: true},
}}, ...);
```

1. Remove “required” from each constraint

After

```
getUserMedia({“video”: {
  // Get 640x480 or fail.  Makes sense!
  “width”: 640,
  “height”: 480
}}, ...);


getUserMedia({“video”: {
  “facingMode”: “environment”,
}}, ...);
```

2. Make them all required.

Whither optional constraints?

“advanced” (in draft already)

```
getUserMedia({“video”: {
  “advanced”: [{
    “width”: 640,
    “height”: 480,
  }]
}}, ...);
```

“ideal” (proposed already)

```
getUserMedia({“video”: {
  “width”: {“ideal”: 640},
  “height”: {“ideal”: 480}
}}, ...);
```

We can do both…

ideal defined in terms of advanced

```
getUserMedia({“video”: {
  “framerate”: {“min”: 5, “ideal”: 15, “max”: 30}
}}, ...);
```

```
getUserMedia({“video”: {
  “framerate”: {“min”: 5, “max”: 30}
  “advanced”: [
    {“framerate”: {“min”: 15, “max”: 15}},
    {“framerate”: {“min”: 14, “max”: 16}},
    {“framerate”: {“min”: 13, “max”: 17}},
    // …
    {“framerate”: {“min”: 5, “max”: 40}}
  ]
}}, ...);
```

Mostly deterministic and easy to understand.
Still have to figure what to do with multiple “ideal”s. How do they mix?

Altogether now

```
if(!supports(“aspectRatio”, “facingMode”)) {
  // Blow up.
}
getUserMedia({“video”: {
  “width”: {"min": 320, “ideal”: 1280, “max”: 1920},
  “height”: {"min": 240, “ideal”: 720, “max”: 1080},
  “aspectRatio”: 1.77777,  // 16.9
  “framerate”: {“min”: 5, “ideal”: 60},
  “facingMode”: “environment”,
  “advanced”: [
    {“width”: 1280, “height”: 720, “framerate”: 60},
    {“width”:  640, “height”: 360, “framerate”: 60},
    {“width”: 1920, “height”: 1080, “framerate”: 30},
    {“width”: 1280, “height”: 720, “framerate”: 30},
    // ...
    {“width”:  320, “height”: 240},
  ]
}}, ...);
```

```
function supports() {
  var supported = MST.getSupportedConstants();
  for(var i = 0; i < arguments.length; i++) {
    if(!supported[arguments[i]]) {
      return false;
    }
  }
  return true;
}
```