Second Screen WG/CG TPAC 2019 - Day 1

Peter Thatcher (pthatcher@google.com)
Mark Foltz (mfoltz@google.com)
Fukuoka September 2019

Day 1 - Outline

Introductions, Agenda Bashing

Overview & Update on Group Work

Open Screen Protocol Spec Changes

OSP 1.0 Spec: Extensibility, Security, Remote Playback

<Lunch>

OSP 1.0 Spec: Remote Playback cont'd, Streaming

OSP V2 Feature Proposals

SSWG/Media & Entertainment IG Joint Session

SSWG/CG Overview & Update

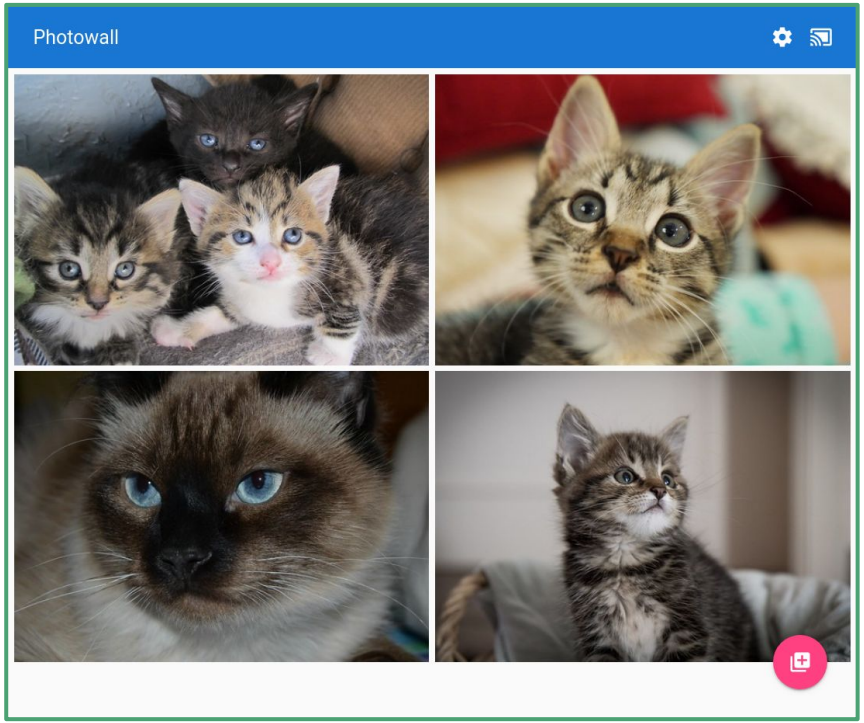
Second Screen CG History

- Nov 2013: Initial charter
- Nov 2013 - Dec 2014: Incubation of Presentation API
- Dec 2014: Presentation API transitioned to Second Screen Working Group
- Sept 2016: CG rechartered to focus on interoperability
- 2016-2017: Requirements, protocol alternatives, benchmarking plan
- Jan-Feb 2018: SSWG rechartered. Phone conference, work plan
- May 2018: Berlin F2F
- October 2018: TPAC 2018
- April 2019: 1.0 draft spec released
- May 2019: Berlin F2F, many refinements to 1.0, added remoting/streaming
- Sept 2019: Here we are :-)

Presentation API

1. **Controlling** page (in a browser) requests presentation of a URL on a **receiver** device (on a connected display).
2. Browser lists displays compatible with the URL; the user selects one to start the presentation.
3. Controlling and receiving pages each receive a **presentation connection.**
4. The connection can be used to exchange messages between the two pages.
5. Either side may close the connection or terminate the presentation.

Presentation API - In practice

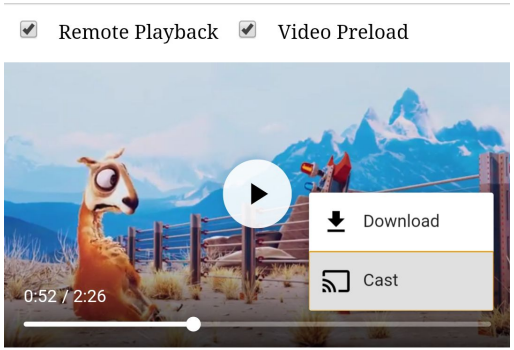


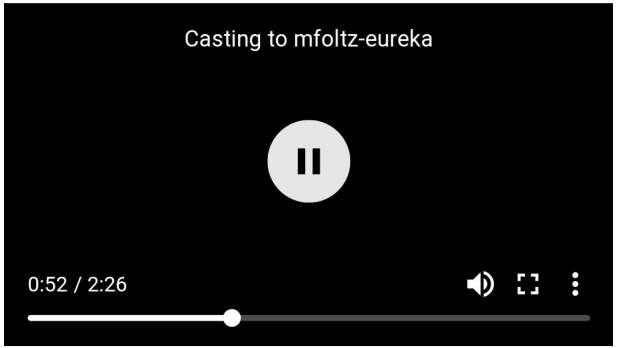
https://googlechromelabs.github.io/presentation-api-samples/photowall/

Remote Playback API

<audio> or <video> element can:

1. Watch for available remote displays
2. Request remote playback by calling `video.remote.prompt()`
3. Media state is synchronized with the remote playback device





Second Screen Community Group Scope

Address **interoperability** through **protocol incubation**

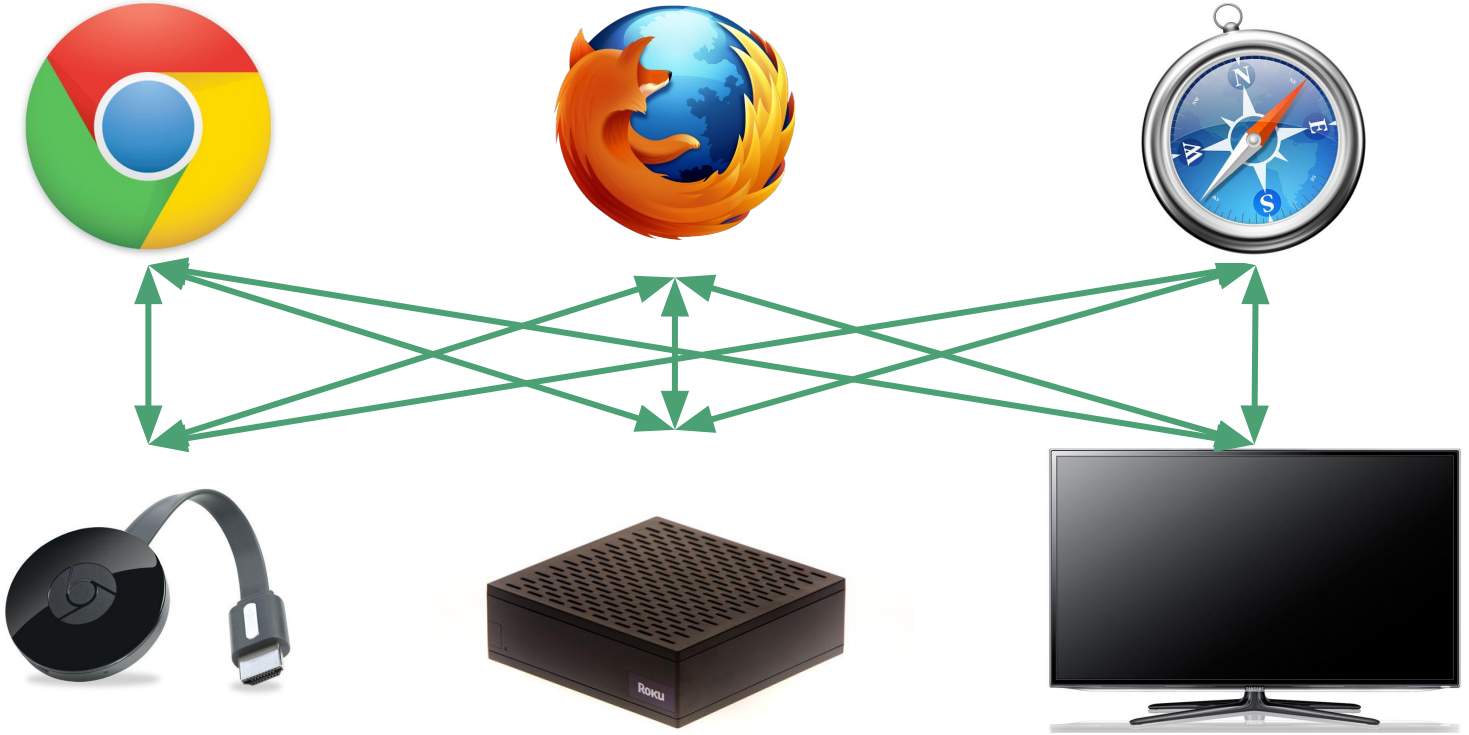
Controllers and receivers (or remote playback devices) on **same LAN**

Presentation API: 2-UA mode (**“flinging” URL**)

Remote Playback API: Remote playback of a **src=“...”** <audio> or <video>

Consider future use cases including **streaming** and **cross-LAN** connections

Open Screen Protocol



Functional Requirements

1. Discovery of receivers and controllers on a shared LAN
2. Implement Presentation API
    a. Determine display compatibility with a URL
    b. Creating a presentation and connection given a URL
    c. Reconnecting to a presentation
    d. Closing a connection
    e. Terminating a presentation
3. Reliable, in-order message exchange
4. Authentication and confidentiality
5. Implement Remote Playback API for <audio> and <video> src=

Non-functional Aspects

Usability

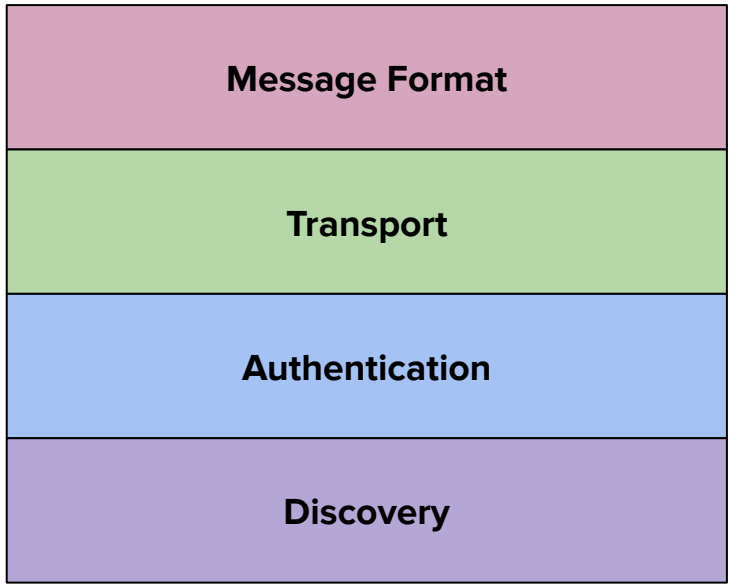
Preserving privacy and security

Resource efficiency (battery, memory)

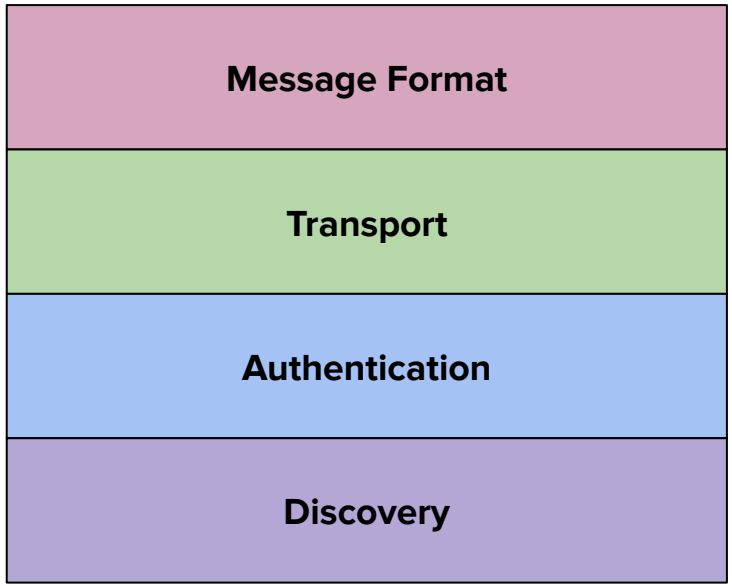
Implementation complexity on constrained devices

Extensibility and upgradeability

Open Screen Protocol - Stack



Open Screen Protocol - Current Approach



Message Format	**CBOR (RFC 7049)**
Transport	**QUIC (Draft RFC)**
Authentication	**TLS 1.3 (RFC 8446) with SPAKE2**
Discovery	**mDNS (RFC 6762) / DNS-SD (RFC 6763)**

Open Screen Protocol - Life of an Agent

1. Agent discovers other agents with mDNS
2. Agent connects with QUIC and and TLS
3. Agent can query remote agent name, capabilities, status
4. To do anything else, agents authenticate with a pairing code
5. Authenticated agents can do presentation, remote playback, streaming
6. Agents can disconnect and reconnect through QUIC without losing sessions

What has been accomplished

- Requirements analysis, research into alternatives
- Decision to pursue the current protocol stack
- Multiple authentication approaches investigated:
  - Challenge/Response w/ HKDF
  - J-PAKE
  - Current spec uses SPAKE2
- V1 spec document has reached completion as a 1.0 draft
- Open Screen Protocol Library implements part of 1.0 spec

Major work items remaining to complete 1.0 spec

- Pairing code (PSK) encoding into numeric value or QR code
- Finish defining TLS 1.3 application profile
- How to add and remove values to/from enums with backwards compatibility
- Refinements to remote playback protocol
- Refinements to streaming and remoting protocols
- Define suspend/resume behavior for network protocols
- Support for HDR media (depending on discussion in Media WG)

Major work items remaining for wide review

- Finish TAG Explainer
- Resolve and merge PRs for "v1-spec" issues
- Document on custom schemes (like cast:, hbbtv:) finished

Implementation: OSP Library

Landed!

Full mDNS support

Full QUIC support

Platform abstraction layer

CBOR support via CDDL codegen

Presentation API support

Demos in C++ and Go

Future Work

SPAKE2 implementation

Capabilities, agent-info

Remote Playback protocol

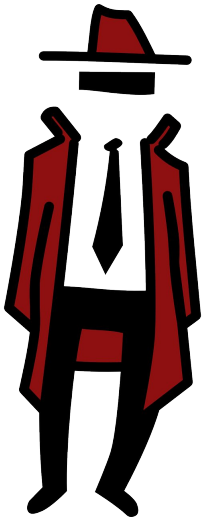
Streaming/remoting protocols

Open Screen Protocol "v2" Proposals

- **Support for GenericCue/DataCue during Remote Playback**
- **Multi-device timing/media synchronization**
- **Attestation**
- **Data frames use cases**
- **Backup/alternate discovery**
- Wake-on-LAN scenarios
- LAN traversal (ICE) support
- "Pings"
- Other v2 features

Review of OSP changes since Berlin

First some names!



Open Screen Protocol Agent	Software / device that implements OSP
Advertising Agent	mDNS responder / QUIC server
Listening Agent	mDNS listener / QUIC client
PSK Presenter	Shows the PSK (pairing code) during auth
PSK Consumer	Receives the PSK from the user

Advertiser/Listener independent from Presenter/Consumer!

More names!

Controller	Presentation Controller / Remote Playback user agent
Receiver	Presentation Receiver / Remote Playback Device
Media Sender	Sends audio/video frames
Media Receiver	Receives audio/video frames

Controller/Receiver independent from Media Sender/Receiver

All are independent of Advertiser/Listener and PSK Presenter/Consumer

Changes to QUIC, CBOR, & TLS

- QUIC connection IDs are zero-length.
- No length prefixing for CBOR messages.
- Use CDDL comments to note message type keys in the spec.
- Use QUIC varints to encode message type keys.
- remote-playback-state uses a one-byte message type key.
- For TLS, require EC certs, and ignore most extensions.
- Clear rules for when a new type key is needed and when a type can be extended without a new type key

Changes to Agents & Capabilities

- Agents now advertise "capabilities" based on what messages they understand.
- Standard capabilities use numbers 1-1000.
- Added receives-audio and receives-video capabilities.
- Agents advertise their preferred locale in agent-info.
- If an agent changes its metadata, they can send an agent-info-event.
- Agents keep a counter to create unique IDs for protocol messages.
- If they reset the counter (e.g. on reboot) they need to update their state-token.
- Agents can advertise extended (non-standardized) capabilities.
- Extended capabilities and messages should be registered in GitHub.

Changes to Remote Playback & Presentation

- Added changed-text-tracks to allow the controller to update text tracks.
- Added add-text-tracks to allow the controller to add text tracks.
- Added algorithm for when to send remote-playback-state messages from receiver to controller.
- Remote Playback ID is a now GUID (to allow reconnection in the future).
- presentation-connection-close-request and -response replaced by -event.
- presentation-change-event added so controllers can count connections.

Changes to Authentication

- Agents use auth-capabilities to decide who inputs the pairing code (PSK).
- Agents exchange psk-min-bits-of-entropy to decide length of PSK.
- Agents advertise a token through mDNS to validate incoming auth requests.
- Added UX guidance for displaying and inputting the PSK.
- Agents must never display a truncated display (friendly) name from mDNS.
- Agents use SPAKE2 to verify the PSK.
- PSK must be the same if QR code or numeric PIN

Changes to Streaming

- Kept data frames
- Added session negotiation (offer/request for encodings)
- Added stats reporting
- Added support for media playback remoting
- Added support for supports-rotation

Other spec changes

- All the terminology changes discussed earlier :-)
- Links between protocol capabilities and API conformance classes
- Can now autolink protocol messages like `agent-info` to definitions

OSP 1.0 Issues to Discuss

Remaining OSP 1.0 Things to Review / Discuss

1. **Security & Authentication Things**
    a. Review the details of TLS 1.3 usage (Issues #130, #135)
    b. Review the security UI guidelines for name display and PSK exchange (#118)
    c. Review the auth-initiation-token (#185)
    d. Review SPAKE2
2. **Remote Playback Things**
    a. Review the remote playback update algorithm (#158)
    b. How to exchange capabilities for HDR rendering (#194)
    c. Do we need any changes for multiple controllers? (#149)
3. **Streaming Things**
    a. Review sessions, stats
    b. Review media remoting
    c. Bidirectional streaming / stream requests (#176)
4. **Extensions** (#175)

Security & Authentication

Discussion: TLS 1.3 & Extensions

PR #212 describes how agents can use TLS:

	Mandatory	Optional
Ciphers	AES-128-GCM / SHA-256	AES-256-GCM / SHA-384 CHACHA20 / SHA-256
Signature Algorithms	secp256r1 / SHA-256	secp384r1 / SHA-384 secp521r1 / SHA-512
Extensions	signature_algorithms supported_groups key_share server_name	All others are ignored.

TLS 1.3: Which ciphers?

- No public benchmarks with the same hardware across all three ciphers.
- ARMv8 is the first generation with AES-NI available (hardware acceleration)
- CHACHA20 is generally very fast for chips without AES-NI

	ARMv7 32-bit	ARMv8a 64-bit	Intel / AMD
AES-128-GCM	???	???	???
AES-256-GCM	???	???	???
CHACHA20	???	???	???

TLS 1.3: Which ciphers?

PROPOSED ACTION: Run a benchmark test (openssl -speed) and use it to fill in this table.

	ARMv7 32-bit	ARMv8a 64-bit	Intel / AMD
AES-128-GCM	???	???	???
AES-256-GCM	???	???	???
CHACHA20	???	???	???

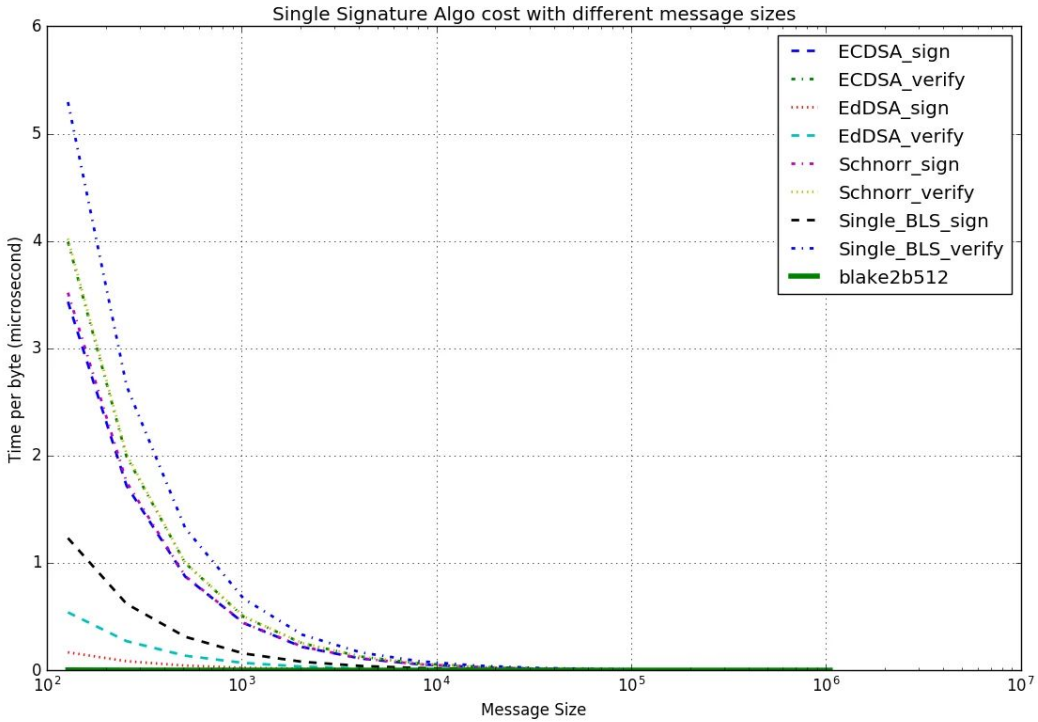
- **Ensure that there are efficient options for hardware with & without AES-NI.**
- **Allow battery powered devices to prefer CHACHA20 if they don't have AES-NI.**

TLS 1.3: Which signature algorithms?

- Currently secp256r1 is mandatory, other ECDSA are recommended
- EdDSA is not allowed, but has some advantages
- No public benchmarks with the same hardware across all algorithms :-(

	"Bits"	ARMv7 32-bit	ARMv8a 64-bit	Intel / AMD
ECDSA secp256r1	128	???	???	???
ECDSA secp384r1	192	???	???	???
ECDSA secp521r1	256	???	???	???
EdDSA 25519	128	???	???	???
EdDSA 448	224	???	???	???

TLS 1.3: Which signature algorithms?



Source

TLS 1.3: Which signature algorithms?

PROPOSED ACTION: Run a benchmark test (openssl -speed) and use it to fill in this table.

	"Bits"	ARMv7 32-bit	ARMv8a 64-bit	Intel / AMD
ECDSA secp256r1	128	???	???	???
ECDSA secp384r1	192	???	???	???
ECDSA secp521r1	256	???	???	???
EdDSA 25519	128	???	???	???
EdDSA 448	224	???	???	???

Note: benchmark both signing and verification.

TLS 1.3: Do we need session resumption?

According to Victor Vasiliev: "If you can get rid of session resumption, get rid of session resumption".

It requires secure storage.

It's only good for 0-RTT data, which we don't need.

Q: Do we have consensus to eliminate session resumption?

TLS 1.3: Do we need the Cookie extension?

HelloRetryRequest is sent by the TLS server when it couldn't generate keys from the ClientHello.

Cookies allow the server to send a hash of the original ClientHello which is replayed with the HelloRetryRequest.

But normally this shouldn't happen since we will mandate a compatible set of cryptographic parameters. Cookies will just add complexity to the client.

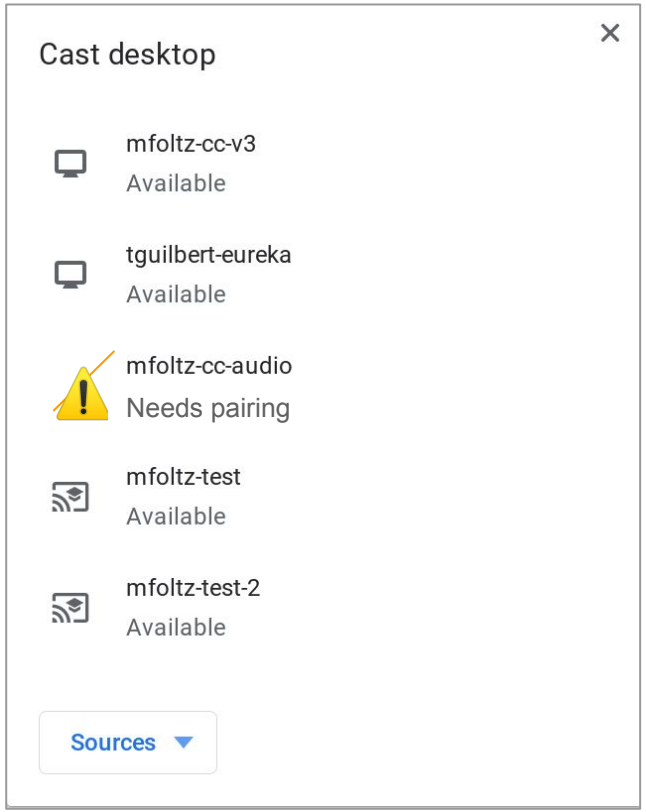
Q: Do we have consensus to not require the Cookie extension?

Authentication and User Interface Guidelines

We don't want to mandate UI; PR #197 and PR #202 added guidelines.

1. Render information that hasn't been verified (pre-auth) differently.
2. If the agent needs to be re-authenticated ("suspicious") then display it differently.
3. Make the PSK display and input hard to spoof.
4. Make the user take action to input the PSK.
5. Meet accessibility guidelines when showing & inputting the PSK.

Authentication and User Interface Guidelines



Note: This is a concept. Final version will look very different.

Authentication and User Interface Guidelines

1. Render information that hasn't been verified (pre-auth) differently.
2. If the agent needs to be re-authenticated ("suspicious") then display it differently.
3. Make the PSK display and input hard to spoof.
4. Make the user take action to input the PSK.
5. Meet accessibility guidelines when showing & inputting the PSK.

Q: Are these sufficient based on what we know now?

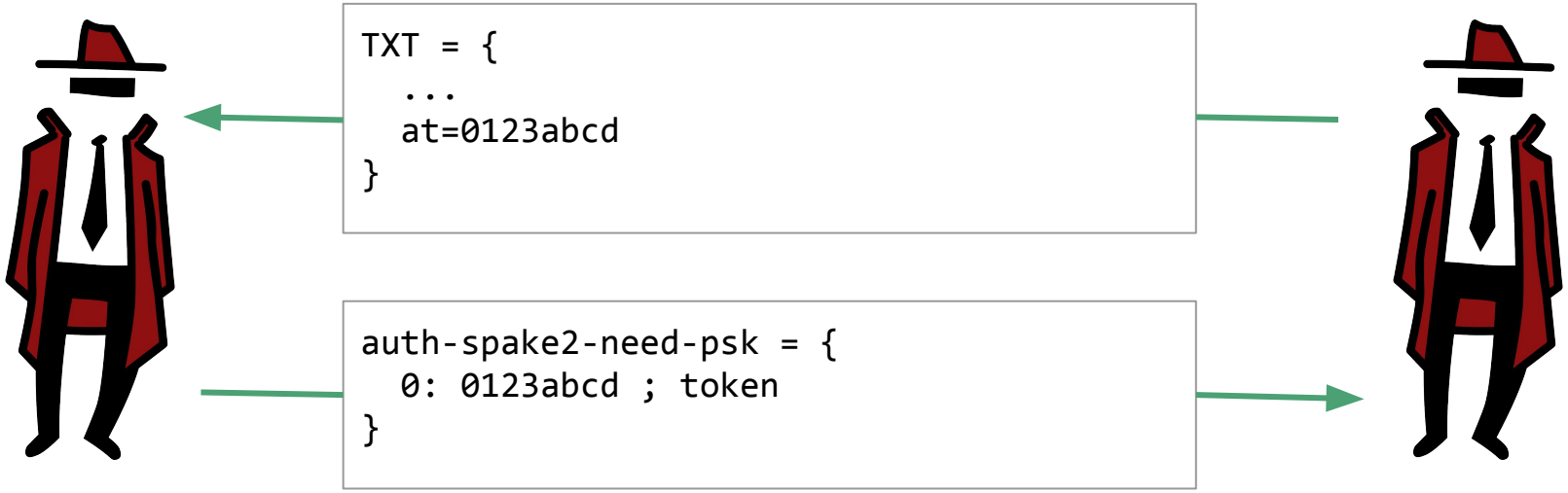
Authentication: auth-initiation-token

What if anyone could send `auth-spake2-need-psk` to your agent? Then a pairing code would pop up. That's annoying!

We added a short, random token advertised through mDNS. This token has to be provided to request authentication (PR #182, PR #189).

Authentication: auth-initiation-token

We use the "at" field in mDNS.



Q: Do we agree this works to prevent misuse of authentication?

Authentication: What PAKE to use?

Current spec uses SPAKE2. (PR #178)

- Challenge/response (proposal #2) requires a memory-hard HKDF (hash) function
  - Exceeds memory requirements for target devices (> 128MB)
- J-PAKE (proposal #1) requires more complex messages, and is not implemented in BoringSSL/OpenSSL.
- SPAKE2 was recommended by Google experts & fits requirements
- However, standardization of SPAKE2 is not complete (but neither is J-PAKE)
- By way, we have a PR to make important properties more explicit

Do we have consensus to move forward with SPAKE2?

Remote Playback Protocol

Remote Playback: Done since Berlin

- Added/refined Remote playback update algorithm
- Table for defaults/required added
- Remoting PR landed (remote playback via streaming)
  - Should we review the message structure of "streaming session attached to remote playback?" We never had consensus on that.

```
remote-playback-start-request = {
  ...
  ? 6: {streaming-session-start-request-params} ; remoting
}
remote-playback-start-response = {
  ...
  ? 2: {streaming-session-start-response-params} ; remoting
}
```

Remote Playback: Not Done since Berlin

- Minor things to do
  - Add extended mime types to remote playback (HTMLSourceElement.type) and Add CSS media query to remote playback (HTMLSourceElement.media)
    - PR for discussion: should these go in the availability request as well? Would support be based on these attributes?
  - Use MediaCapabilities/CSS colorspace values
    - PR for discussion: is that the right reference?
- Issue #146: Recommended HTTP headers: any idea what these should be?
- Issue #194: Capabilities for HDR rendering and Display (in 2 slides)
- Issue #149: Multiple controllers of remote playback (next slide)

Remote Playback multiple controllers

- Would require some API changes
  - Something like RemotePlayback.reconnect.
  - Could also overload RemotePlayback.prompt() but that seems confusing and different than Presentation API
- Questions
  - Should it require the same URL like the Presentation API does?
  - If the not and the URLs differ, should it push over the new one?

HDR

Related MediaCapabilities issues:

- w3c/media-capabilities#118
  - ```
    enum HdrCapability {
      “HDR10”,
      “HDR10Plus”,
      “DolbyVision”,
      “HLG”,
    }
    ```
  - Or more complex things. There's a lively discussion!
- w3c/media-capabilities#119
  - The above plus width + height, which we already cover

Question: Should we do something now or wait until this settles?
    ```

Streaming

Streaming: Done since Berlin

- Merged big streaming PR that finished session start stats
- Remoting PR landed (remote playback via streaming)
- Re-added data frames synced with audio and video
- Add video rotation capability

Streaming: Note Done since Berlin

- Per-codec max resolution (limited by sender)
  (Related to something on next slides, so let's go there....)

Streaming new issues

- Issue #223: Codec switching when remoting (next slide)
- Issue #176: Bidirectional streaming / stream request (in 2 slides)

Remoting changing codec

- Problem: currently the session is started like this:
  - Sender: "I can send you codec A or B"
  - Receiver: "I would like codec B"
- But if the source stream switches to codec A, the sender has to transcode
- Might be better as:
  - Receiver (via capabilities): "I can receive A w/profile X, A w/ profile Y, B up to resolution Z, or C"
  - Sender: "I can send you logical stream M"
  - Receiver: "I would like logical stream M"
- Now the sender chooses which codec at any time, rather than the receiver.
- However, we need more complex capabilities
- Alternative: every time the codec switches, the sender sends a message to the receiver asking it to pick again (yuck)

Bidirectional streaming / stream request

If I want to receive media from you (like a TV pulling up a video doorbell feed), what do I do?

We could add a "please stream me media" message which simply causes the sender to send a `streaming-session-start-request` message (`streaming-session-want-to-receive?`).

For bidirectional streaming, we could do either of:

A. Start two unidirectional sessions
B. Attach a `streaming-session-want-to-receive` to a `streaming-session-start-request`.

Extensions and Capabilities

Capabilities & Extensions

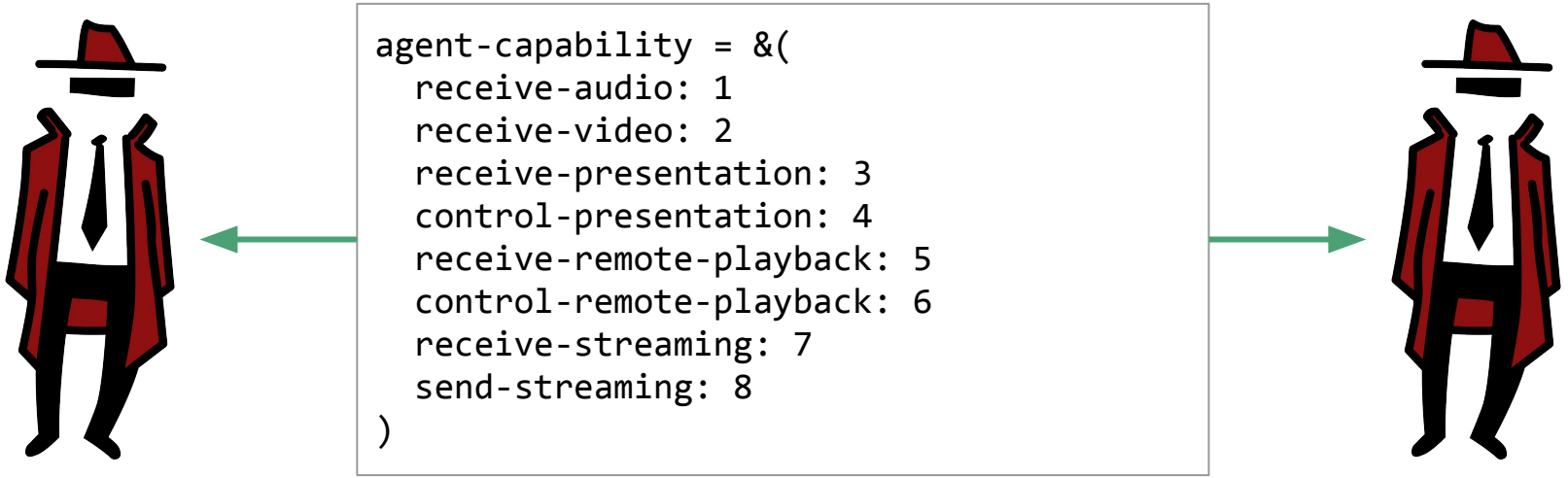
Agents discover what each other can do through capabilities.

```
agent-info = {
  0: text ; display-name
  1: text ; model-name
  2: [* agent-capability] ; capabilities
  3: text ; state-token
  4: [* text] ; locales
}
```

Capabilities & Extensions

We defined some standard capabilities, which map onto messages in the spec.

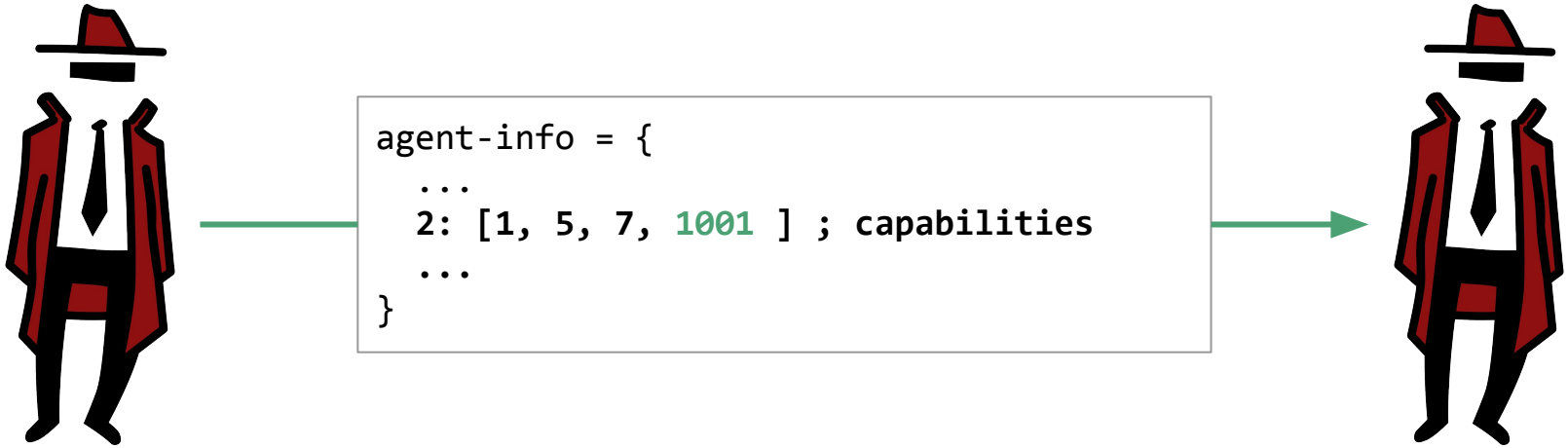
Agents can only send messages the other will understand.



(Still discussing meaning of receive-audio and receive-video; Issue #200)

Capabilities & Extensions

PR #183 adds a way for agents to add "extended" capabilities with IDs >= 1000



Extended capabilities can add new messages and fields to existing messages.

This allows vendor specific protocols to be supported (like device setup).

Capabilities & Extensions

We added a public registry all capability IDs to avoid conflicts.

Open Screen Protocol Capabilities

Id	Name	Description	Message Type IDs
1	`receive-audio`	Audio Receiver	22
2	`receive-video`	Video Receiver	23
3	`receive-presentation`	Presentation API Receiver	14,16,104,106,109,113
4	`control-presentation`	Presentation API Controller	15,16,103,105,107,108,1
5	`receive-remote-playback`	Remote Playback Receiver	17,115,117,119
6	`control-remote-playback`	Remote Playback Controller	18,20,21,114,116,118
7	`receive-streaming`	Streaming Receiver	24
8	`send-streaming`	Streaming Sender	

Capabilities & Extensions

If you want to register an extension send a PR. (Eventually we'll use IANA.)

Id	Name	Organization	Description	Message Type IDs
1000	`frobinate-xyzzy`	FrobozzCo	Adds xyzzy capability	49-51, 8193-8199

Q: Do we have consensus that this is a good model for extensions?

Open Screen Protocol 1.0 wrap-up

State of the Repository: 17 "v1-spec" issues

Remote Playback Protocol	**6**
Streaming Protocol	**3**
Security	**5**
Other	**2**

(Plus issues identified here at TPAC)

Propose merging PRs for all issues except HDR, then fixing TODOs, then closing meta issue and calling OSP 1.0 done!
(And scrubbing old/obsolete issues that are not "v2".)

Open Screen Protocol V2 Features

Open Screen Protocol V2 Features

- **Support for DataCue**
- **Attestation**
- **Data Frames use cases**
- **Alternative Discovery**
- **Multi-device timing (to be discussed in joint session)**

Support for GenericCue / DataCue

We have AudioFrame, VideoFrame, and DataFrame all synced. How about first-class support for TextFrame?

Would be like a DataFrame but with a payload which would match the form of DataCue/TextTrackCue. ie either:

A. .data of byte
B. .value of {
   key: String
   data: String | Number | Array | ArrayBuffer | Object
   locale: String
  }

Attestation

Attestation

Attestation is how an agent finds out information about another agent, attested by a trusted party. What are interesting things to attest?

- Manufacturer and model name (to show in the UI)
- Serial number (to avoid counterfeit devices)
- OS/software version
- Compliance with certain standards (i.e., HDCP)
- Audio, video, or other capabilities

Attestation

In general attestation is done through certificates.

The agent wanting to attest hands over a certificate signed by the trusted party.

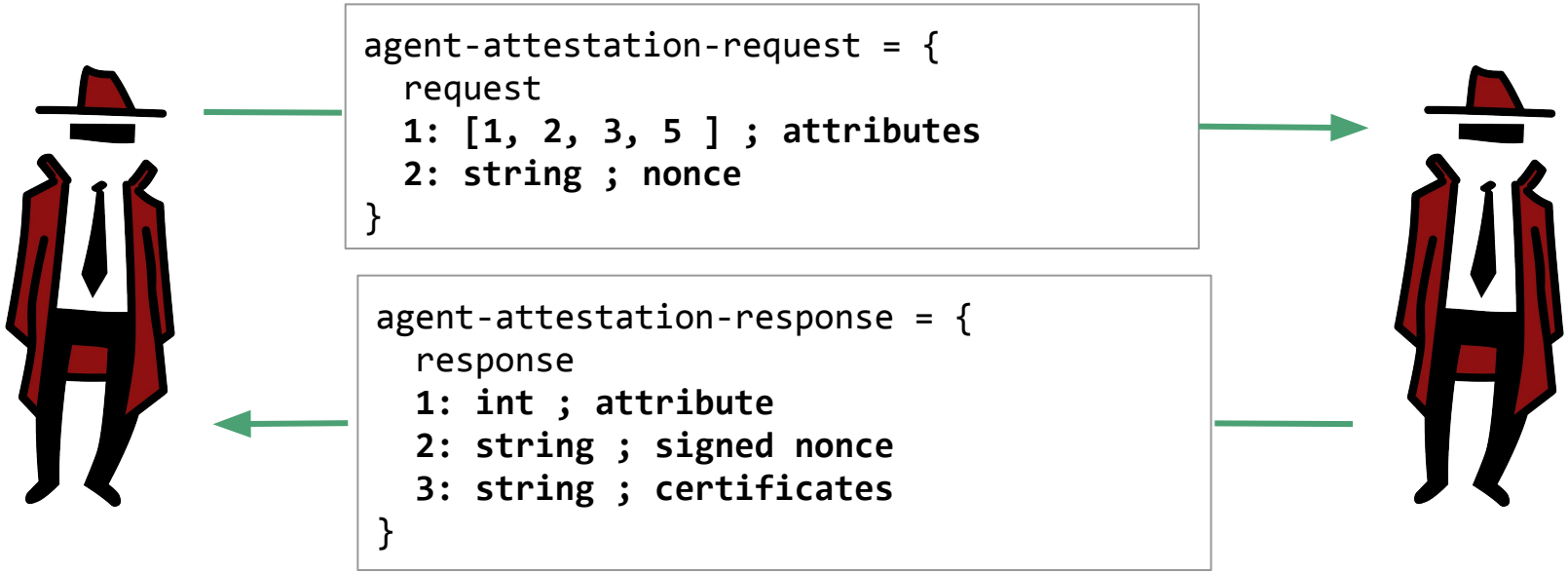
The agent requesting authentication inspects the certificate and verifies the signature (or chain of signatures).

Note that these certificates can be baked into the device, generated on demand, or fetched from a server.

These certificates are **not related** to the agent certs for transport auth.

One model for attestation

An agent can ask another agent for attestable attributes.



The agent responds by signing the request and providing the certificate(s).

Attestation

There is a lot to figure out here:

How is this done currently? Some precedents with EME, WebAuthN.

How do we bind attestation to devices using hardware backed certificates?

Do we want to link this to OSP authentication? (Maybe skip pairing codes.)

Do we expose this to applications? That has fingerprinting and privacy implications.

PROPOSED ACTION: Start a companion note separately with use cases, requirements, and draft framework.

Alternative Discovery

What if mDNS doesn't work?

I could have my WiFi turned off.

It could be a managed network (separate networks, client isolation).

It could be a display in a public place, hotels, or a friend's house.

We've discussed ICE (RFC 8445) as a solution for connectivity. How do we get it started?

To connect to an agent you need:

1. A way to trigger ICE on the other agent.
2. A way to exchange ICE candidates.
3. A way to get the other agent's auth-initiation-token.

Alternative Discovery Proposal

Define an Open Screen Beacon format.

The beacon should be hard to guess; maybe it's a one-time token.

Beacon could be obtained through BTLE, NFC or a QR code.

The beacon should include the hostname of a service we can use for signaling.

The agent who wants to connect should pass the beacon to this service with some candidates. The service will communicate with the other agent and relay candidates back to the original agent.

Once ICE is connected, OSP can proceed as usual.

Alternative Discovery Proposal

Define a beacon format.

Q: Does this sound like a good direction for enabling alternative discovery?

PROPOSED ACTION: Write this up outside of the community group repository along with an explainer.

Media & Entertainment IG Joint Session

Second Screen WG/CG TPAC 2019 - Day 2

Peter Thatcher (pthatcher@google.com)
Mark Foltz (mfoltz@google.com)
Fukuoka September 2019

Day 2 - Outline

Agenda Bashing

Remaining Day 1 Topics

New API Features

OSP Wide Review, TAG Explainer

SSWG Rechartering

Remote Playback API

Remote Playback "disconnected state"

GitHub

Remote buffer state for Remote Playback + MSE

Remote Playback + MSE

- Receiver buffer may be too small
  ⇒ The sender UA can limit the transmission to the receiver
- Bitrate may be higher than network bandwidth
  ⇒ HTMLMediaElement.buffered and readyState can be used
- Alternatively: new API
  ⇒ Pull request on GitHub

Code example

```
const video = document.querySelector('#my-video');
video.src = window.URL.createObjectURL(mediaSource);
video.remote.addEventListener('remotingstatechanged', onRemotingStateChanged);

function onRemotingStateChanged() {
  switch (video.remote.remotingBufferState) {
    case 'insufficient-data':
      lowerResolution();
      break;
    case 'too-much-data':
      pauseBufferingSegments();
      break;
  }
}
```

Web IDL

```
partial interface RemotePlayback {
  readonly attribute RemotingBufferState remotingBufferState;
  attribute EventHandler onremotingbufferstatechanged;
};

enum RemotingBufferState {
  "insufficent-data",
  "enough-data",
  "too-much-data",
  "not-remoting"
};
```

Proposal: One prompt for Presentation and Remote Playback APIs

Issue on GitHub

State of the current APIs

Two separate methods to start sessions:

- `PresentationRequest.start()`
- `RemotePlayback.prompt()`

Each shows a potentially different list of receiver devices to choose from, so user may need to open two different device selection dialogs to find a device

Example code

```
const presentation =
    new PresentationRequest('https://example.com/myvideo.html');
const remote = document.querySelector('#my-video').remote;
const device = await navigator.secondScreen.prompt(presentation, remote);

if ((device.supportsPresentation && myPagePrefersPresentation()) ||
    !device.supportsRemotePlayback) {
  const connection = await device.startPresentation();  // Doesn't prompt
} else {
  device.startRemotePlayback();  // Doesn't prompt
}
```

Web IDL

```
interface SecondScreen {
  Promise<SecondScreenDevice> prompt(PresentationRequest presentationRequest,
                                     RemotePlayback remotePlayback);
};

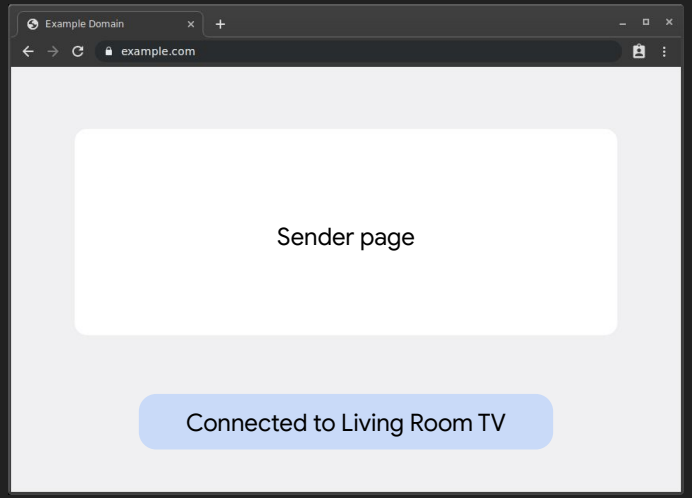
interface SecondScreenDevice {
  readonly attribute boolean supportsPresentation;
  readonly attribute boolean supportsRemotePlayback;

  Promise<PresentationConnection> startPresentation();
  Promise<void> startRemotePlayback();
};
```

Proposal: Presentation receiver friendly name

PRs on GitHub: controller side, receiver side

Example code




```
const request = new PresentationRequest('https://example.com/receiver.html');
const connection = request.start();
connection.addEventHandler('connect', () => {
  document.querySelector('#status').innerText = `Connected to ${connection.receiverName}`;
});
```

Web IDL

```
// Controlling user agent:
partial interface PresentationConnection {
  readonly attribute USVString receiverName;
};

// Receiving user agent:
partial interface PresentationReceiver {
  readonly attribute USVString friendlyName;
};
```

Streaming API

Maybe we don't need one

We could just support this:

```
const element = ...;  // Some HtmlMediaElement
element.srcObject = mediaStream;
element.remote.start();
```

With remoting, that's the same as streaming a MediaStream. Would a different streaming API provide any advantages?

OSP 1.0 Wide Review

Open Screen Protocol Wide Review

TAG "Explainer"

Homework: please review PR so we can publish with the 1.0 spec.

Open Screen Protocol Wide Review

Who should be asked to review?

TAG

WebAppSec

PING

Accessibility (WAI?)

SSWG Rechartering

Recharter Draft

- Draft
- Diff of material changes
- Added to scope
  - Presentation of part of an HTML document
  - Remote Playback features for OSP
  - Presentation/Remote Playback integrations
- Out of scope
  - Network protocols (?)
  - Codecs
  - Input methods

Rechartering: Decision making

- Draft
- Option 1: Extend charter for 2 years with no protocols, to allow implementations of OSP to be finished.
- Option 2: Extend charter for 1 year to see if we can finish implementations, then decide on protocols.
- Option 3: Extend charter for 6 months to gather feedback on standardizing protocols, then recharter again.

Design Time

Remote Playback Capabilities

```
partial interface RemotePlayback {
  Promise<RemotingCapabilitiesInfo> remotingInfo(
          RemotingConfiguration configuration)
}

dictionary RemotingConfiguration : MediaConfiguration {
};

dictionary RemotingCapabilitiesInfo : MediaCapabilitiesInfo {
};
```

Includes MediaCapabilities

```
dictionary VideoConfiguration {
  required DOMString contentType;
  required unsigned long width;
  required unsigned long height;
  required unsigned long long bitrate;
  required DOMString framerate;
  boolean hasAlphaChannel;
};
dictionary AudioConfiguration {
  required DOMString contentType;
  DOMString channels;
  unsigned long long bitrate;
  unsigned long samplerate;
  boolean spatialRendering;
}
```

```
dictionary MediaCapabilitiesInfo {
  required boolean supported;
  required boolean smooth;
  required boolean powerEfficient;
}
```

Where is the codec profile? Embedded in the contentType :(.

Example

```
let remote = ...;
let info = await remote.remotingInfo({
  video : {
    contentType : 'vp8',
    width : 640,
    height : 480,
    bitrate : 10000,
    framerate : '30'
  }
});
if (info.supported) {
  ...
}
```

OSP CDDL

```
receive-video-capability = {
  // Already there!
  0: format ; codec
  ? 1: video-resolution ; max-resolution
  ? 2: ratio ; max-frames-per-second
  ? 3: uint ; max-pixels-per-second
  // New
  ? 10: max-bits-per-second
}
```

Explicit remoting signal

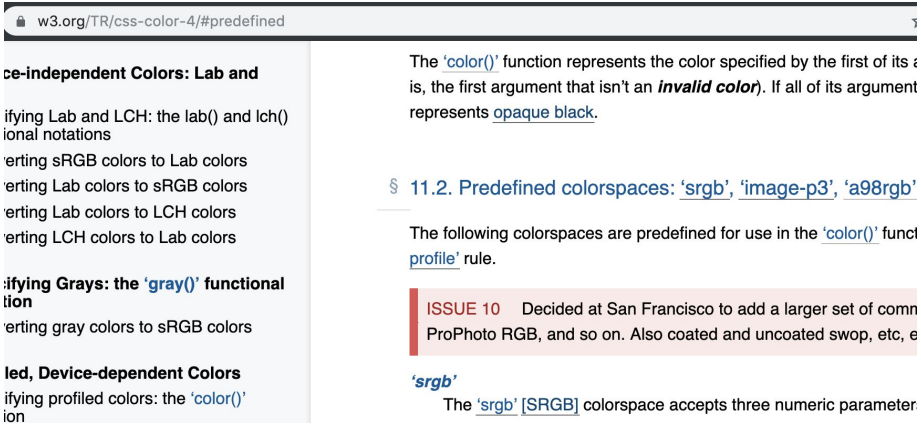
```
remote-playback-controls = {
  ...
  ? 1: text ; source-url / remoting
}

remote-playback-start-request = {
  ...
  2: [* remote-playback-source] / remoting ; sources
}

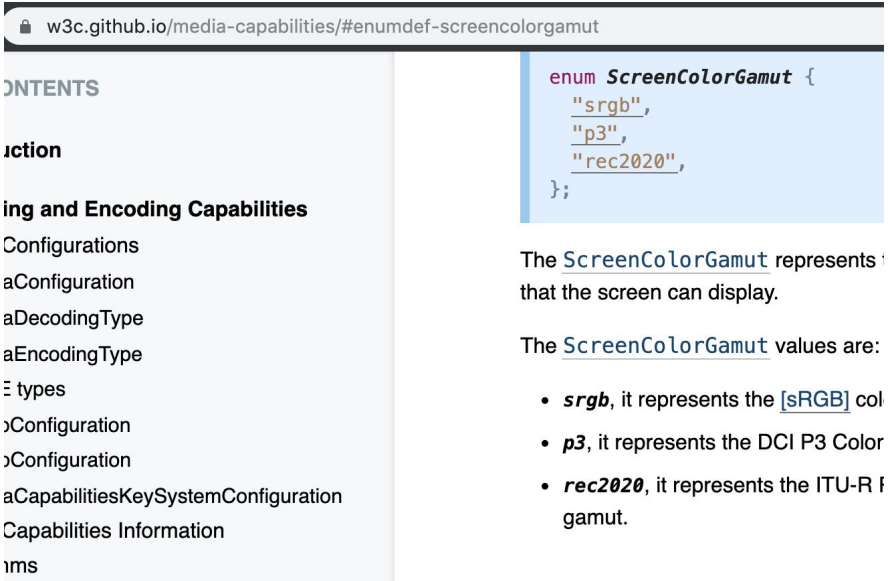
remote-playback-state = {
   ...
  ? 2: text ; source-url / remoting
}

remoting = 0
```

Color spaces



Which one?



Remote playback reconnect

```
let remote = ...;
let token = ...;
await remote.prompt({allowReconnectWithToken: token});
... get disconnected or transfer to another machine ...
await remote.reconnect(token);
```

Remote playback reconnect IDL

```
partial interface RemotePlayback {
  Promise<void> start(RemotePlaybackStartParameters params);
  Promise<void> reconnect(DOMString token);
}

dictionary RemotePlaybackStartParameters {
  DOMString allowReconnectWithToken;
}
```

DataCue in normal Remote Playback (not streaming)

```
text-track-cue = {
  ...
  3: text ; text
  4: data ; data
}

cue-data = {
  1: text ; key
  2: text / bytes / float ; value
  3: text ; locale
}
```

DataCue in Remote Playback with streaming

```
text-frame = {
  0: uint; encoding-id
  ? 1: uint ; sequence-number
  ? 2: uint ; start-time
  ? 3: uint ; duration
  4: cue-data; cue-data  // Part different from data-cue
  ? 5: media-time ; sync-time
}
```

End of TPAC 2019 slides