

display.js - code that is responsible for displaying local and remote media

This module contains code related to displaying local and remote media. Code is separated in two closures - one for local media and another one for remote

Display local media

1. Wrapper function

`initLocalDisplay()` [code](#)

Define top level closure for local media code

```
const initLocalDisplay = function(localDisplayElement){  
  ...  
}
```

2. Local variables

[code](#)

Define local variables

```
const localDisplayDiv = localDisplayElement;  
const localDisplays = {};
```

3. Remove local display

`removeLocalDisplay()` [code](#)

Define function that will remove local display once track is ended

```
const removeLocalDisplay = function(id) {  
  delete localDisplays[id];  
  $('#' + id).remove();  
  reassembleLocalLayout();  
}
```

4. Find video element without audio track

`getAudioContainer()` [code](#)


Define function that will find video element without audio track.

```
const getAudioContainer = function() {
  for (const [key, value] of Object.entries(localDisplays)) {
    let video = value.getElementsByTagName("video");
    if (video && video[0]) {
      let audioStateButton = value.getElementsByTagName("button");
      let audioTracks = video[0].srcObject.getAudioTracks();
      if (!audioTracks || audioTracks.length === 0) {
        return {
          id: value.id,
          video: video[0],
          audioStateDisplay: audioStateButton[0]
        };
      }
    }
  }
};
```

5. Add track to local display

`add()` [code](#)

Define add function which will add new track to local display

 **add**



Check if new track is an audio track. If so try to find video element without audio track attached. In case video element was found attach new audio track to it and return.

[code](#)

```
if (stream.getAudioTracks().length > 0) {
  let videoElement = getAudioContainer();
  if (videoElement) {
    let track = stream.getAudioTracks()[0];
    videoElement.video.srcObject.addTrack(track);
    videoElement.audioStateDisplay.innerHTML = "Audio state: " +
stream.getAudioTracks()[0].enabled;
    track.addEventListener("ended", function() {
      videoElement.video.srcObject.removeTrack(track);
      videoElement.audioStateDisplay.innerHTML = "Audio state: " +
false;

      //check video element has no tracks left
      for (const [key, vTrack] of
Object.entries(videoElement.video.srcObject.getTracks())) {
        if (vTrack.readyState !== "ended") {
          return;
        }
      }
    });
  }
}
```

```

        removeLocalDisplay(videoElement.id);
    });
    return;
}
}

```

Create new display container

code

```

const coreDisplay = document.createElement('div');
coreDisplay.setAttribute("style", "width:200px; height:auto; border: solid;
border-width: 1px");
coreDisplay.id = stream.id;

```

Create and add name container to display container

code

```

const streamNameDisplay = document.createElement("div");
streamNameDisplay.innerHTML = "Name: " + name;
streamNameDisplay.setAttribute("style", "width:auto; height:30px");
coreDisplay.appendChild(streamNameDisplay);

```

Create and add audio state display. Subscribe to `click` event to handle mute/unmute.

code

```

const audioStateDisplay = document.createElement("button");
audioStateDisplay.setAttribute("style", "width:auto; height:30px");
audioStateDisplay.innerHTML = "Audio state: " +
(stream.getAudioTracks().length > 0 ? stream.getAudioTracks()[0].enabled :
false);
audioStateDisplay.addEventListener('click', function(){
    if (stream.getAudioTracks().length > 0) {
        stream.getAudioTracks()[0].enabled = !(stream.getAudioTracks()
[0].enabled);
        audioStateDisplay.innerHTML = "Audio state: " +
stream.getAudioTracks()[0].enabled;
    }
});
coreDisplay.appendChild(audioStateDisplay);

```

Create stream display which will hold video element

code

```

const streamDisplay = document.createElement('div');
streamDisplay.id = id;
streamDisplay.setAttribute("style", "width:auto; height:auto");
coreDisplay.appendChild(streamDisplay);

```

Create video element and add it to stream display

code

```
const video = document.createElement("video");
streamDisplay.appendChild(video);
video.srcObject = stream;
video.muted = true;
video.onloadedmetadata = function (e) {
  video.play();
};
```

Subscribe to track's `ended` event. Once track is ended check if video element has any active tracks. If not - remove display

code

```
stream.getTracks().forEach(function(track){
  track.addEventListener("ended", function() {
    video.srcObject.removeTrack(track);
    //check video element has no tracks left
    for (const [key, vTrack] of
Object.entries(video.srcObject.getTracks())) {
      if (vTrack.readyState !== "ended") {
        return;
      }
    }
    removeLocalDisplay(id);
  });
});
```

Subscribe to `resize` event to keep video inside the container

code

```
video.addEventListener('resize', function (event) {
  streamNameDisplay.innerHTML = "Name: " + name + " " + video.videoWidth +
"x" + video.videoHeight;
  resizeVideo(event.target);
});
```

Save display, reassemble local displays and return newly created one

code

```
localDisplays[id] = coreDisplay;
reassembleLocalLayout();
return coreDisplay;
```

6. Refresh local display grid

`reassembleLocalLayout()` [code](#)

Define helper which will recalculate local displays grid and reassemble local displays.

```
const reassembleLocalLayout = function() {
  let gridWidth = gridSize(Object.keys(localDisplays).length).x;
  let container = document.createElement('div');
  let row;
  let rowI = 1;
  let colI = 0;
  for (const [key, value] of Object.entries(localDisplays)) {
    if (row) {
      if (colI >= gridWidth) {
        row = createRow(container);
        rowI++;
        colI = 0;
      }
    } else {
      row = createRow(container);
    }
    $("##" + key).detach();
    let col = createCol(row);
    col.appendChild(value);
    colI++;
  }
  $(localDisplayDiv).empty();
  localDisplayDiv.appendChild(container);
}
```

7. Export add function for main code

[code](#)

```
return {
  add: add
}
```

Display remote media

1. Wrapper function

`initRemoteDisplay()` [code](#)

Define top level closure for remote media code

```
const initRemoteDisplay = function(room, mainDiv, peerConnection) {
  ...
}
```

2. Local variables

[code](#)

Define local variables

```
const constants = SFU.constants;
const remoteParticipants = {};
```

3. Subscribe to room events

[code](#)

Subscribe to related room events



Room events



SFU_ROOM_EVENT.ADD_TRACKS

Find participant. If not found create a new one

[code](#)

```
let participant = remoteParticipants[e.info.nickName];
if (!participant) {
  participant = {};
  participant.nickName = e.info.nickName;
  participant.tracks = [];
  participant.displays = [];
  remoteParticipants[participant.nickName] = participant;
}
```

Add new tracks to the participant

[code](#)

```
participant.tracks.push.apply(participant.tracks, e.info.info);
```

Create display for every video track that is being added

[code](#)

```
for (const pTrack of e.info.info) {
  let createDisplay = true;
  for (let i = 0; i < participant.displays.length; i++) {
    let display = participant.displays[i];
    if (pTrack.type === "VIDEO") {
      if (display.hasVideo()) {
        continue;
      }
    }
  }
}
```

```

        }
        display.videoMid = pTrack.mid;
        display.setTrackInfo(pTrack);
        createDisplay = false;
        break;
    } else if (pTrack.type === "AUDIO") {
        if (display.hasAudio()) {
            continue;
        }
        display.audioMid = pTrack.mid;
        createDisplay = false;
        break;
    }
}
if (!createDisplay) {
    continue;
}
let display = createRemoteDisplay(participant.nickName,
participant.nickName, mainDiv);
participant.displays.push(display);
if (pTrack.type === "VIDEO") {
    display.videoMid = pTrack.mid;
    display.setTrackInfo(pTrack);
} else if (pTrack.type === "AUDIO") {
    display.audioMid = pTrack.mid;
}
}
}

```

SFU_ROOM_EVENT.REMOVE_TRACKS

Find remote participant. If not found return

code

```

const participant = remoteParticipants[e.info.nickName];
if (!participant) {
    return;
}

```

Walk through tracks

code

```

for (const rTrack of e.info.info) {
    ...
}

```

Find and remove participant's track that has the same mid as track that is being removed

code

```

for (let i = 0; i < participant.tracks.length; i++) {
    if (rTrack.mid === participant.tracks[i].mid) {
        participant.tracks.splice(i, 1);
    }
}

```

```

        break;
    }
}

```

Find display that corresponds to track and remove track from the display. If display has no active tracks remove display as well.

code

```

for (let i = 0; i < participant.displays.length; i++) {
    let found = false;
    const display = participant.displays[i];
    if (display.mids.audio.includes(rTrack.mid)) {
        //remove from mids array
        display.mids.audio.splice(display.mids.audio.indexOf(rTrack.mid), 1);
        //stop track and remove stream
        display.audioStreams[rTrack.mid].getAudioTracks()[0].stop();
        delete display.audioStreams[rTrack.mid];
        //remove audio element
        display.display.removeChild(display.audioElements[rTrack.mid]);
        delete display.audioElements[rTrack.mid];
        found = true;
    } else if (display.mids.video === rTrack.mid) {
        display.mids.video = undefined;
        display.mediaStream.getVideoTracks()[0].stop();
        found = true;
    }
    if (display.mids.audio.length === 0 && display.mids.video === undefined)
    {
        const video = display.display.getElementsByTagName("video")[0]
        video.pause();
        video.srcObject = null;
        display.display.remove();
        participant.displays.splice(i, 1);
    }
    if (found) {
        break;
    }
}

```

SFU_ROOM_EVENT.LEFT

Find and remove participant

code

```

let participant = remoteParticipants[e.name];
if (!participant) {
    return;
}
participant.displays.forEach(function(display){
    display.dispose();
})
delete remoteParticipants[e.name];

```


SFU_ROOM_EVENT.TRACK_QUALITY_STATE

Find participant. Return if not found

[code](#)

```
console.log("Received track quality state");
const participant = remoteParticipants[e.info.nickName];
if (!participant) {
  return;
}
```

Walk through tracks

[code](#)

```
for (const rTrack of e.info.tracks) {
  ...
}
```

Find corresponding display and update quality state


[code](#)

```
const mid = rTrack.mid;
for (let i = 0; i < participant.displays.length; i++) {
  const display = participant.displays[i];
  if (display.videoMid === mid) {
    display.updateQualityInfo(rTrack.quality);
    break;
  }
}
```

4. Create remote display

`createRemoteDisplay()` [code](#)

Helper function which will create display based on stream and track info.

 **createRemoteDisplay**



Remove remote display

`dispose()` [code](#)

```
dispose: function() {
  cell.remove();
}
```

Hide remote display

`hide()` [code](#)

```
hide: function(value) {
  if (value) {
    cell.style.display = "none";
  } else {
    cell.style.display = "block";
  }
}
```

Create remote display audio element

`setAudio()` [code](#)

```
setAudio: function(stream) {
  if (audio) {
    audio.remove();
  }
  if (!stream) {
    audio = null;
    this.audioMid = undefined;
    return;
  }
  audio = document.createElement("audio");
  audio.controls = "controls";
  cell.appendChild(audio);
  audio.srcObject = stream;
  audio.play();
}
```

Create remote display video element

`setVideo()` [code](#)

```
setVideo: function(stream) {
  if (video) {
    video.remove();
  }

  if (stream == null) {
    video = null;
    this.videoMid = undefined;
    qualityDivs.forEach(function(div) {
      div.remove();
    });
    qualityDivs = [];
    tidDivs.forEach(function(div) {
      div.remove();
    });
    tidDivs = [];
    return;
  }
}
```

```

    video = document.createElement("video");
    streamDisplay.appendChild(video);
    video.srcObject = stream;
    video.onloadedmetadata = function (e) {
        video.play();
    };
    video.addEventListener("resize", function (event) {
        streamNameDisplay.innerHTML = "Name: " + name + " " +
        video.videoWidth + "x" + video.videoHeight;
        resizeVideo(event.target);
    });
}

```

Display track quality info

`setTrackInfo()` [code](#)

```

setTrackInfo: function(trackInfo) {
    if (trackInfo && trackInfo.quality) {
        for (let i = 0; i < trackInfo.quality.length; i++) {
            const qualityDiv = document.createElement("button");
            qualityDivs.push(qualityDiv);
            qualityDiv.innerText = trackInfo.quality[i];
            qualityDiv.setAttribute("style", "display:inline-block; border:
solid; border-width: 1px");
            qualityDiv.style.color = "red";
            qualityDiv.addEventListener('click', function(){
                console.log("Clicked on quality " + trackInfo.quality[i] + "
trackId " + trackInfo.id);
                if (qualityDiv.style.color === "red") {
                    return;
                }
                for (let c = 0; c < qualityDivs.length; c++) {
                    if (qualityDivs[c].style.color !== "red") {
                        qualityDivs[c].style.color = "gray";
                    }
                }
                qualityDiv.style.color = "blue";
                room.changeQuality(trackInfo.id, trackInfo.quality[i]);
            });
            qualityDisplay.appendChild(qualityDiv);
        }
        for (let i = 0; i < 3; i++) {
            const tidDiv = document.createElement("button");
            tidDivs.push(tidDiv);
            tidDiv.innerText = "TID"+i;
            tidDiv.setAttribute("style", "display:inline-block; border:
solid; border-width: 1px");
            tidDiv.style.color = "gray";
            tidDiv.addEventListener('click', function(){
                console.log("Clicked on TID " + i + " trackId " +
trackInfo.id);
                for (let c = 0; c < tidDivs.length; c++) {
                    tidDivs[c].style.color = "gray";
                }
                tidDiv.style.color = "blue";
                room.changeQuality(trackInfo.id, null, i);
            });
            tidDivs.push(tidDiv);
        }
    }
}

```

```

    });
    tidDisplay.appendChild(tidDiv);
  }
}

```

Update quality state

`updateQualityInfo()` [code](#)

```

updateQualityInfo: function(videoQuality) {
  for (const qualityInfo of videoQuality) {
    for (const qualityDiv of qualityDivs) {
      if (qualityDiv.innerText === qualityInfo.quality){
        if (qualityInfo.available === true) {
          qualityDiv.style.color = "gray";
        } else {
          qualityDiv.style.color = "red";
        }
        break;
      }
    }
  }
}

```

5. Work with peer connection

[code](#)

Subscribe to `PeerConnection.ontrack` event.

```

peerConnection.ontrack = ({transceiver}) => {
  let rParticipant;
  console.log("Attach remote track " + transceiver.receiver.track.id + "
kind " + transceiver.receiver.track.kind + " mid " + transceiver.mid);
  for (const [nickName, participant] of Object.entries(remoteParticipants))
  {
    for (const pTrack of participant.tracks) {
      console.log("Participant " + participant.nickName + " track " +
pTrack.id + " mid " + pTrack.mid);
      if (pTrack.mid === transceiver.mid) {
        rParticipant = participant;
        break;
      }
    }
    if (rParticipant) {
      break;
    }
  }
  if (rParticipant) {
    for (const display of rParticipant.displays) {
      if (transceiver.receiver.track.kind === "video") {
        if (display.videoMid === transceiver.mid) {
          let stream = new MediaStream();

```

```

        stream.addTrack(transceiver.receiver.track);
        display.setVideo(stream);
        break;
    }
    } else if (transceiver.receiver.track.kind === "audio") {
        if (display.audioMid === transceiver.mid) {
            let stream = new MediaStream();
            stream.addTrack(transceiver.receiver.track);
            display.setAudio(stream);
            break;
        }
    }
    } else {
        console.warn("Failed to find participant for track " +
transceiver.receiver.track.id);
    }
}

```

Find participant based on track's mid

code

```

let rParticipant;
console.log("Attach remote track " + transceiver.receiver.track.id + " kind "
+ transceiver.receiver.track.kind + " mid " + transceiver.mid);
for (const [nickName, participant] of Object.entries(remoteParticipants)) {
    for (const pTrack of participant.tracks) {
        console.log("Participant " + participant.nickName + " track " +
pTrack.id + " mid " + pTrack.mid);
        if (pTrack.mid === transceiver.mid) {
            rParticipant = participant;
            break;
        }
    }
    if (rParticipant) {
        break;
    }
}

```

Find corresponding display among participant's displays and add track.

code

```

for (const display of rParticipant.displays) {
    if (transceiver.receiver.track.kind === "video") {
        if (display.videoMid === transceiver.mid) {
            let stream = new MediaStream();
            stream.addTrack(transceiver.receiver.track);
            display.setVideo(stream);
            break;
        }
    }
    } else if (transceiver.receiver.track.kind === "audio") {
        if (display.audioMid === transceiver.mid) {
            let stream = new MediaStream();

```

```
        stream.addTrack(transceiver.receiver.track);  
        display.setAudio(stream);  
        break;  
    }  
}  
}
```