RTCIceCandidate.usernameFragment

Syntax
Usage notes
Example
Specifications
Browser compatibility

Related Topics
WebRTC API
RTCIceCandidate
▼ Constructor
RTCIceCandidate()
▼ Properties
address
candidate
component
foundation
port
priority
protocol
relatedAddress
relatedPort
sdpMid
sdpMLineIndex
usernameFragment
▼ Methods
toJSON()
▼ Related pages for WebRTC
MediaDevices.getUserMedia()
Navigator.mediaDevices
RTCCertificate

The read-only **`usernameFragment`** property on the `RTCIceCandidate` interface is a string indicating the username fragment ("ufrag") that uniquely identifies a single ICE interaction session.

This value is specified when creating the `RTCIceCandidate` by setting the corresponding `usernameFragment` value in the `RTCIceCandidateInit` object when creating a new candidate with `new RTCIceCandidate()`. Note that 24 bits of the username fragment are required to be randomized by the browser. See Randomization below for details.

If you instead call `RTCIceCandidate()` with a string parameter containing the `candidate` m-line text, the value of `usernameFragment` is extracted from the m-line.

Syntax

```
var ufrag = RTCIceCandidate.usernameFragment;
```

Value

A `DOMString` containing the username fragment (usually referred to in shorthand as "ufrag" or "ice-ufrag") that, along with the ICE password ("ice-pwd"), uniquely identifies a single ongoing ICE interaction, including for any communication with the STUN server. The string may be up to 256 characters long, and has no default value.

Randomization

At least 24 bits of the text in the `ufrag` are required to be randomly selected by the ICE layer at the beginning of the ICE session. The specifics for which bits are random and what the remainder of the `ufrag` text are are left up to the browser implementation to decide. For example, a browser might choose to always use a 24-character `ufrag` in which bit 4 of each character is randomly selected between 0 and 1. Another example: it might take a user-defined string and append three 8-bit random bytes to the end. Or perhaps every character is entirely random.

RTCDTMFSender
RTCDTMFToneChangeEvent
RTCDataChannel
RTCDataChannelEvent
RTCDtlsTransport
RTCErrorEvent
RTCIceTransport
RTCPeerConnection
RTCPeerConnectionIceErrorEvent
RTCPeerConnectionIceEvent
RTCRtpReceiver
RTCRtpSender
RTCRtpTransceiver
RTCSctpTransport
RTCSessionDescription
RTCStatsEvent
RTCStatsReport
RTCTrackEvent

Usage notes

ICE uses the `usernameFragment` and password to ensure message integrity. This avoids crosstalk among multiple ongoing ICE sessions, but, more importantly, helps secure ICE transactions (and all of WebRTC by extension) against attacks that might try to inject themselves into an ICE exchange.

> **Note:** There is no API to obtain the ICE password, for what should be fairly obvious security reasons.

The `usernameFragment` and password both change every time an ICE restart occurs.

Example

Although the WebRTC infrastructure will filter out obsolete candidates for you after an ICE restart, you can do it yourself if you're trying to absolutely minimize the number of messages going back and forth.

To do so, you can compare the value of `usernameFragment` to the current `usernameFragment` being used for the connection after receiving the candidate from the signaling server and before caling `addIceCandidate()` to add it to the set of possible candidates.

When the web app receives a message from the signaling server that includes a candidate to be added to the `RTCPeerConnection`, you can (and generally *should*) simply call `addIceCandidate()`. There's not typically a need to manually worry about filtering the candidates.

However, let's imagine that we do need to minimize traffic. The function below, `ssNewCandidate()`, is called when a message, `signalMsg`, arrives from the signaling server that contains an ICE candidate to be added to the `RTCPeerConnection`. To avoid including candidates obsoleted by an ICE restart, we can use code like this:

```
const ssNewCandidate = signalMsg => {
  let candidate = new RTCIceCandidate(signalMsg.candidate);
  let receivers = pc.getReceivers();

  receivers.forEach(receiver => {
    let parameters = receiver.transport.getParameters();

    if (parameters.usernameFragment === candidate.usernameFragment) {
      return;
    }
  });
```

```
12
13    pc.addIceCandidate(candidate)
14      .catch(reportError);
15  }
```

This walks through the list of the `RTCRtpReceiver` objects being used to receive ICE data, and looks to see if the `usernameFragment` indicated in the candidate matches any of them. If it does, `ssNewCandidate()` aborts. Otherwise, after checking every receiver, it adds the new candidate to the connection.

Specifications

Specification	Status	Comment
WebRTC 1.0: Real-time Communication Between Browsers The definition of 'RTCIceCandidate.usernameFragment' in that specification.	CR Candidate Recommendation	Initial definition.

Browser compatibility

Update compatibility data on GitHub

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Android webview	Chrome for Android	Firefox for Android	Opera for Android	Safari on iOS	Samsung Internet
`usernameFragment`	74	≤79	67	No	?	?	74	74	67	?	?	11.0

What are we missing?

Full support

No support

Compatibility unknown

Last modified: May 7, 2019, by MDN contributors