Third Party Integration Service (TPIS)

This document describes how to integrate **Connect Me** within another web application.

- Events
- Actions
- IFrame
- Capabilities & Ready
- Logging in
- Calls
  - `newSession`
  - `sessionStatusUpdate`
- SIP Connectivity
- Start a call
- Answer a call
- Hangup a call
- Reject a call
- Open conversation page
- Complete Example

Connect Me is expected to be running within an <iframe> in your web application.

Connect Me interacts with its environment using the Window.postMessage() API. There are 2 kinds of messages exchanged on the window: - Outgoing events, or "events" - Incoming actions, or "actions"

Events

Events are messages posted on the window by Connect Me. These result in events being emitted by the "window" with the following data:

```
// event.data example
{
    event: "eventName",
    someArgument: { ... },
    someOtherArgument: "blabla"
}
```

Actions

Actions are messages posted on the window by your application and destined to Connect Me. These are destined to control Connect Me.

```
// event.data example
{
    action: "actionName",
    someArgument: { ... },
    someOtherArgument: "blabla"
}
```

IFrame

The first step is to include the *iframe* in your application. The `src` attribute must contain URL of Connect Me. Ensure that you include `#tpis=v1` at the end of the URL, to ensure the TPIS interface is enabled.

```
<!DOCTYPE html>
<html>
  <head>
        <title>Your web application</title>
  </head>
  <body>
      <iframe
        id="connect-me"
        src="https://xyz.connect.fuzer.net#tpis=v1"
        allow="camera *; microphone *; fullscreen *;"
        style="width: 800px; height: 600px;">
      </iframe>
      <script>
        // Javascript will go here.
      </script>
  </body>
</html>
```

> Your application must be running under the **https://** protocol to avoid compatibility issues.
>
> Your domain must be added to the list of accepted origins, otherwise Connect Me will ignore the actions posted. This is an action to be executed by the provider of Connect Me.

Once Connect Me is loaded it will emit a `capabilitiesRequest` message.

Capabilities & Ready

Your application should respond with the list of capabilities supported by your application. According to the provided capabilities Connect Me will either support some additional features (like sharing chat messages) or delegate some of its features to your application (like generating the ringing tone)

Some features of Connect Me can be delegated to your application:

- Generating the ringer tone - `nativeRinger` - This is geared for integrations with mobile phones, where the phone should use the system ringer, insted of the ringer provided by Connect Me.
- Switching from headphone to speakerphone - `speaker` - This is geared for integrations for mobile phones, when enabled it allows Connect Me to display a speaker button in the call session window, which allows the user to switch between the headphone to speaker. If enabled your application should react to this event and switch the audio accordingly.
- Sharing chat messages - `share` - This indicates if the platform supports sharing content.

You would answer with the following message:

```
{
  capabilities: {
    capA: true,
    capB: true
  }
}
```

Lets add the code to listen to messages sent by Connect Me. It is mandatory to reply to the capabilities request, otherwse the TPIS interface will be suspended. In our example we won't be activating any capability.

```
<script>
    const iframe = document.getElementById('connect-me')
    window.addEventListener('message', ({ data }) => {
      if (data === 'capabilitiesRequest') {
        post({ capabilities: {} })
      }
    })
    function post (data) {
      iframe.contentWindow.postMessage(data, '*')
    }
</script>
```

In your browser console you should see a message: `GOT-CAPABILITIES`. Connect Me is now ready to interact with your application. It will send back a `ready` event.

Logging in

This step can be ignored if you use display the Connect Me interface in your application, in that case your users can use the login interface from Connect Me to login.

The `login` action can be used to log into Connect Me. This can be useful if Connect Me is not visible within your application. It can also be useful if your application already knows the user credentials, avoiding entering credentials again for the end user.

```
{
  action: 'login',
  method: 'basic',
  credentials : { username: 'user@example.com', password: 'yourPassword' }
}
```

We adapted our previous code to send the `login` action once we receive the ready event.

```
...
<script>
    // This snipped assumes that the capabilitiesRequest was already answered.
    const iframe = document.getElementById('connect-me')
    window.addEventListener('message', ({ data }) => {
      const { event, ...args } = data
      switch (event) {
        case 'ready': return ready()
      }
    })
    function ready () {
      post({
        action: 'login',
        method: 'basic',
        credentials : { username: 'user@example.com', password: 'yourPassword' }
      })
    }
    function post (data) {
      iframe.contentWindow.postMessage(data, '*')
    }
</script>
...
```

Calls

Two events are emitted by Connect Me for managing calls.

`newSession`

A newSession event is emitted when a new call is handled by Connect Me. It can be an incoming or outoing call.

```
{
  id: 0,
  type: "SipSession",
  incoming: true,
  status: "calling",
  remoteIdentity: {
    displayName: "John Doe",
    uri: {
      user: "+3227887900"
    }
  }
}
```

- `id`: identifier for the call
- `type`: Can either be `SipSession` or `UepChannelSession`. When Connect Me's softphone feature is enabled `SipSession` sessions will be available, this kind of session is created for all calls answered/dialed with Connect Me's softphone. `UepChannelSession` are always available, and are created for all calls that are not handled by Connect Me itself (FMU / Polycom managed by UEP). Take care that on incoming calls until you answer the call you might have a SipSession and UepChannelSession for the same call.
- `incoming`: when `true` indicates that its an incoming call, otherwise its an outgoing call.
- `status`: current status for the call. Can be: `calling`, `ringing`
- `remoteIdentity`: provides information about the other party. On outgoing calls the displayName will be `undefined`.

`sessionStatusUpdate`

A `sessionStatusUpdate` event is emitted when a call changes state. Example:

```
{
  id: 4,
  status: "confirmed"
}
```

- `id`: refers to the identifier of the call (see `newSession` event)
- `status`: provides information about the status of a call. Can be: `calling`, `ringing`, `confirmed` (answered), `terminated`.

The following example extends the base example to listen for session events:

```
...
<script>
  // This snipped assumes that the capabilitiesRequest was already answered.
  const iframe = document.getElementById('connect-me')
  window.addEventListener('message', ({ data }) => {
```

```
    const { event, ...args } = data
    switch (event) {
      case 'newSession': return onSession(args)
      case 'sessionStatusUpdate': return onSessionUpdate(args)
    }
  })
  function onSession ({ id, type, status, incoming, remoteIdentity, ...data }) {
    console.log(`>>TPIS Call ${id} was detected`)
    console.log(`>>TPIS Call ${id} is incoming call:`, incoming)
    console.log(`>>TPIS Call ${id} is outgoing call:`, !incoming)
    console.log(`>>TPIS Call ${id} type ${type}`) // SipSession / UepChannelSession
    if (incoming) {
      const name = remoteIdentity.displayName
      const number = remoteIdentity.uri.user
      console.log(`>>TPIS Call ${id} is from: ${name} <${number}>`)
    }
  }
  function onSessionUpdate ({ id, status, ...data }) {
    switch (status) {
      case 'confirmed': return console.log(`>>TPIS Call ${id} was answered`)
      case 'terminated': return console.log(`>>TPIS Call ${id} was terminated`)
    }
  }
</script>
...
```

SIP Connectivity

In order to use the softphone, the application should wait for SIP Connectivity. A `sipRegistered` event is emitted by Connect Me, when a SIP user agent is started:

```
{
  event: "sipRegisted"
}
```

Start a call

To dial a number you must use the `dial` action. Example:

```
{
  action: "dial",
  number: "8001"
}
```

- `number`: number that will be composed.

```
<script>
    // This snipped assumes that the capabilitiesRequest was already answered.
    const iframe = document.getElementById('connect-me')
    function post (data) {
      iframe.contentWindow.postMessage(data, '*')
    }
    post({ action: "dial", number: "8001" })
</script>
```

Answer a call

To hangup a call that was not answered you need to use the `answerSession` action.

```
{
  action: "answerSession",
  id: 1
}
```

- `id`: refers to the identifier of the call (see `newSession` event)

```
<script>
  // This snipped assumes that the capabilitiesRequest was already answered.
  const iframe = document.getElementById('connect-me')
  function post (data) {
    iframe.contentWindow.postMessage(data, '*')
  }
  post({ action: "answerSession", id: 1 })
</script>
```

Hangup a call

To hangup a call that was answered you need to use the `terminateSession` action.

```
{
  action: "terminateSession",
  id: 1
}
```

- `id`: refers to the identifier of the call (see `newSession` event)

```
<script>
  // This snipped assumes that the capabilitiesRequest was already answered.
  const iframe = document.getElementById('connect-me')
```

```
  function post (data) {
    iframe.contentWindow.postMessage(data, '*')
  }
  post({ action: "terminateSession", id: 1 })
</script>
```

Reject a call

To hangup a call that was not answered you need to use the `rejectSession` action.

```
{
  action: "rejectSession",
  id: 1
}
```

- `id`: refers to the identifier of the call (see `newSession` event)

```
<script>
  // This snipped assumes that the capabilitiesRequest was already answered.
  const iframe = document.getElementById('connect-me')
  function post (data) {
    iframe.contentWindow.postMessage(data, '*')
  }
  post({ action: "rejectSession", id: 1 })
</script>
```

Open conversation page

To open a conversation page:

```
{
  action: "conversation",
  uri: "ID00000099@127.0.0.1"
}
```

- `uri`: identifier of the conversation.

```
<script>
    // This snipped assumes that the capabilitiesRequest was already answered.
    const iframe = document.getElementById('connect-me')
    function post (data) {
      iframe.contentWindow.postMessage(data, '*')
    }
    post({ action: "conversation", uri: "ID00000099@127.0.0.1" })
</script>
```

Complete Example

```html
<!DOCTYPE html>
<html>
  <head>
        <title>Your web application</title>
  </head>
  <body>
      <iframe
        id="connect-me"
        src="https://xyz.connect.fuzer.net#tpis=v1"
        allow="camera *; microphone *; fullscreen *;"
        style="width: 800px; height: 600px;">
      </iframe>
      <script>
          const iframe = document.getElementById('connect-me')
          window.addEventListener('message', ({ data }) => {
            if (data === 'capabilitiesRequest') {
              return post({ capabilities: {} })
            }
            const { event, ...args } = data
            switch (event) {
              case 'ready': return onReady(args)
              case 'login': return onLogin(args)
              case 'newSession': return onSession(args)
              case 'sessionStatusUpdate': return onSessionUpdate(args)
              default: console.warn('>>TPIS Unkown event:', event)
            }
          })
          function onReady () {
            post({
              action: 'login',
              method: 'basic',
              credentials : { username: 'user@example.org', password: 'your-password' }
            })
          }
          function onLogin () {
            console.log('>>TPIS Congratulations, you are logged in.')

            // We can now dial a number
            // post({ action: "dial", number: "8001" })

            // Or open a conversaion:
            // post({ action: "conversation", uri: "ID00000099@127.0.0.1" })
          }
```

```
        function onSession ({ id, type, status, incoming, remoteIdentity, ...data }) {
          console.log(`>>TPIS Call ${id} was detected`)
          console.log(`>>TPIS Call ${id} is incoming call:`, incoming)
          console.log(`>>TPIS Call ${id} is outgoing call:`, !incoming)
          console.log(`>>TPIS Call ${id} type ${type}`) // SipSession / UepChannelSession
          if (incoming) {
            const name = remoteIdentity.displayName
            const number = remoteIdentity.uri.user
            console.log(`>>TPIS Call ${id} is from: ${name} <${number}>`)
          }
        }
        function onSessionUpdate ({ id, status }) {
          switch (status) {
            case 'confirmed': return console.log(`>>TPIS Call ${id} was answered`)
            case 'terminated': return console.log(`>>TPIS Call ${id} was terminated`)
          }
        }
        function post (data) {
          iframe.contentWindow.postMessage(data, '*')
        }
      </script>
  </body>
</html>
```