



TangoFX Sessions for Developers

TangoFX Sessions is a customizable solution where developers can create and add their own widget over the underlying WebRTC communication mechanism . It can support extensive set of user activity such as video chat , message , play games , collaborate on code , draw something together etc . It can go as wide as your imagination .

This manual describes the process of creating widgets to host over existing TFX platform . Note that this manual contains examples and guidelines and users are requested to follow the specific guidelines like directory structure and composition to avoid confusion and complexity.

Contents

[Prerequisites](#)

[How does TFX Sessions work ?](#)

[How to make widgets using TFX API ?](#)

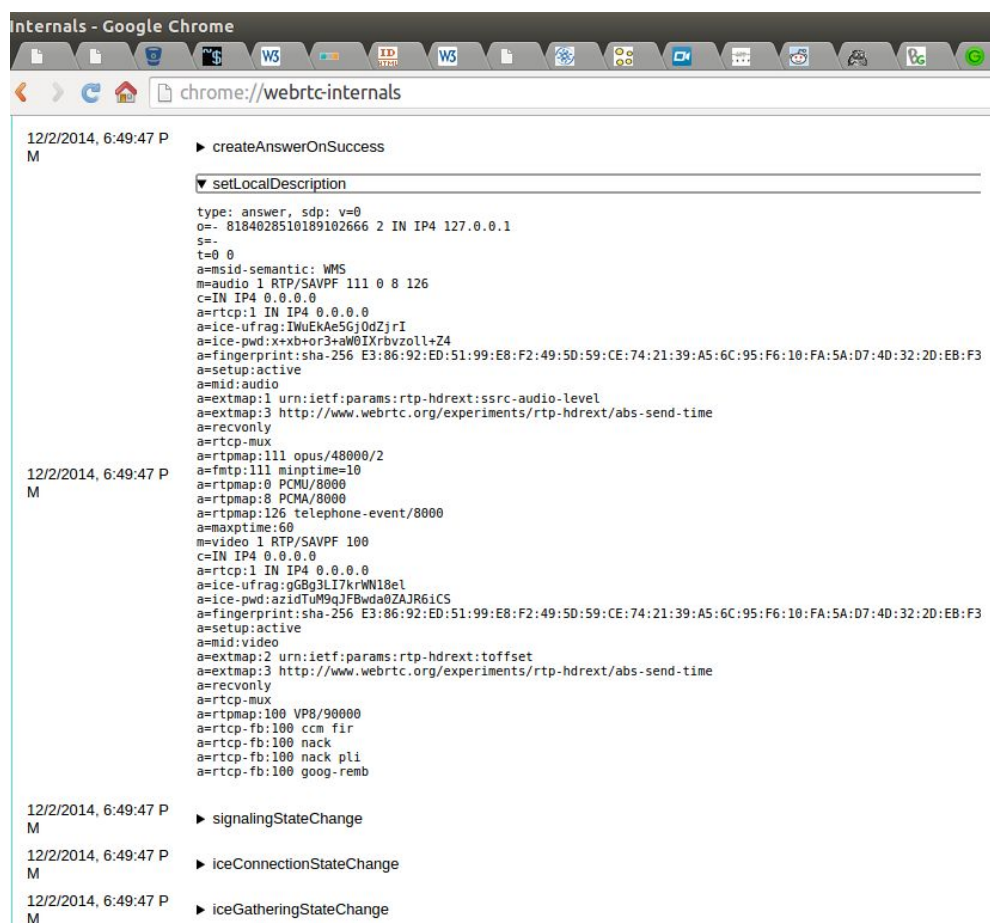
Prerequisites

It is required to have TFX Chrome extension installed and running from Chrome App Store under above . To do this follow the steps described in **TangoFX v0.1 User's manual**.

How does TFX Sessions work ?

TFX Sessions uses the browser's media API's , like getUserMedia and Peerconnection to establish p2p media connection . Before media can traverse between 2 end points the signalling server is required to establish the path using Offer- Answer Model .

TFX Sessions uses socketio based handshake between peers to ascertain that they are valid endpoints to enter in a communication session . This is determined by SDP (Session Description Parameters) . The same can be observed in <chrome://webrtc-internals/>



```
Internals - Google Chrome
chrome://webrtc-internals

12/2/2014, 6:49:47 P
M ▶ createAnswerOnSuccess
  ▼ setLocalDescription
    type: answer, sdp: v=0
    c= 8184028510189102666 2 IN IP4 127.0.0.1
    s=-
    t=0 0
    a=msid-semantic: WMS
    m=audio 1 RTP/SAVPF 111 0 8 126
    c=IN IP4 0.0.0.0
    a=rtcp:1 IN IP4 0.0.0.0
    a=ice-ufrag:1WuEkAe5Gj0dZjrI
    a=ice-pwd:x+xb+or3+aW0IArbvzoll+Z4
    a=fingerprint:sha-256 E3:86:92:ED:51:99:E8:F2:49:5D:59:CE:74:21:39:A5:6C:95:F6:10:FA:5A:D7:40:32:2D:EB:F3
    a=setup:active
    a=mid:audio
    a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
    a=extmap:3 http://www.webrtc.org/experiments/rtp-hdrext/abs-send-time
    a=recvonly
    a=rtcp-mux
    a=rtpmap:111 opus/48000/2
    a=rtcpmap:111 minptime=10
    a=rtpmap:0 PCMU/8000
    a=rtpmap:8 PCMA/8000
    a=rtpmap:126 telephone-event/8000
    a=maxptime:60
    m=video 1 RTP/SAVPF 100
    c=IN IP4 0.0.0.0
    a=rtcp:1 IN IP4 0.0.0.0
    a=ice-ufrag:gGBg3LI7krWN18el
    a=ice-pwd:azidTuM9qJFBwda0ZAJR6iCS
    a=fingerprint:sha-256 E3:86:92:ED:51:99:E8:F2:49:5D:59:CE:74:21:39:A5:6C:95:F6:10:FA:5A:D7:40:32:2D:EB:F3
    a=setup:active
    a=mid:video
    a=extmap:2 urn:ietf:params:rtp-hdrext:toffset
    a=extmap:3 http://www.webrtc.org/experiments/rtp-hdrext/abs-send-time
    a=recvonly
    a=rtcp-mux
    a=rtpmap:100 VP8/90000
    a=rtcp-fb:100 ccm fir
    a=rtcp-fb:100 nack
    a=rtcp-fb:100 nack pli
    a=rtcp-fb:100 goog-remb

12/2/2014, 6:49:47 P
M ▶ signalingStateChange

12/2/2014, 6:49:47 P
M ▶ iceConnectionStateChange

12/2/2014, 6:49:47 P
M ▶ iceGatheringStateChange
```

How to make widgets using TFX API ?

To make widgets for TFX , just write your simple web program which should consist of one main html webpage and associated css and js files for it .

1. Find an interesting idea which is requires minimal js and css . Remember it is a widget and not a full fleshed web project , however js frameworks like requirejs , angularjs , emberjs etc , work as well.
2. Make a compact folder with the name of widget and put the respective files in it. For example the html files or view files would go to src folder , javascript files would goto js folder , css files would goto css folder , pictures to picture folder , audio files to sound folder and so on .
3. Once the widget is performing well in standalone environment , we can add a sync file to communicate the peer behaviors across TFX network . For this we primarily use 2 methods :
 - SendMessage : To send the data that will be traversed over DataChannel API of TFX . The content is in json format and will be shared with the peers in the session .
 - OnMessage : To receive the message communicated by the TFX API over network
4. Submit the application to us or test it yourself by adding the plugin description in in widgetmanifest.json file . Few added widgets are

```
[
{
  "plugintype": "code",
  "id"      : "LiveCode",
  "type"       : "code",
  "title"  : "Code widget",
  "icon"       : "btn btn-style glyphicon glyphicon-tasks",
  "url"  : "../plugins/AddCode/src/codewindow.html"
},

{
  "plugintype": "draw",
  "id"      : "Draw",
  "type"       : "draw",
  "title"  : "Code widget",
  "icon"       : "btn btn-style glyphicon glyphicon-pencil",
  "url"  : "../plugins/AddDraw/src/drawwindow.html"
},
```

```

{
  "plugintype": "pingpong",
  "id"      : "Pingpong",
  "type"   : "pingpong",
  "title": "ping pong widget",
  "icon"   : "btn btn-style glyphicon glyphicon-record",
  "url"    : "../plugins/AddPingpong/src/main.html"
}
]

```

5. For proper orientation of the application make sure that overflow is hidden and padding to left is atleast 60 px so that it doesnt overlap with panel

```

padding-left: 60px;
overflow: hidden;

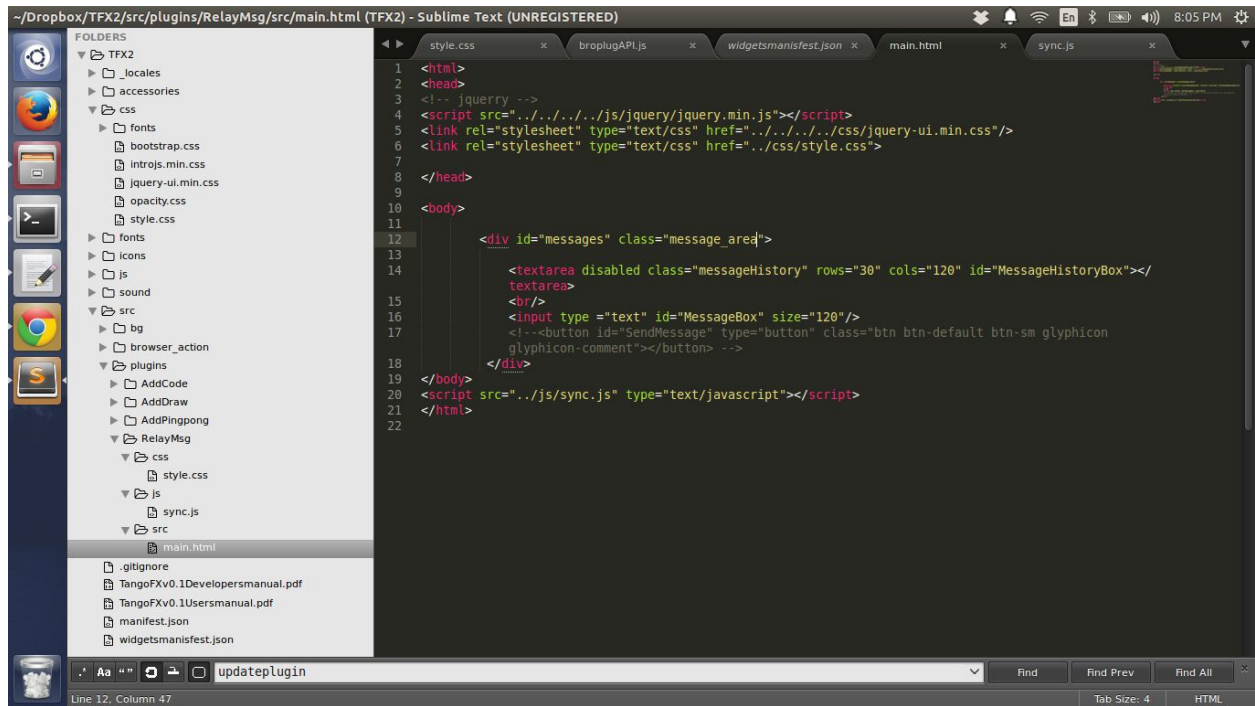
```

6. Voila the widget is ready to go .

Simple messaging widget

For demonstration purpose we have summarised the exact steps followed to create the sample widget

1. Think of a general chat scenario as present in various messaging si
2. Made a folder structure with separation for js , css and src. Add the respective files in folder. It would look like following figure:



3. The html main page is

```
<html>
<head>
<!-- jquery -->
<script src="../../../../js/jquery/jquery.min.js"></script>
<link rel="stylesheet" type="text/css" href="../../../../css/jquery-ui.min.css"/>
<link rel="stylesheet" type="text/css" href="../../css/style.css">
</head>

<body>
<div id="messages" class="message_area">
<textarea disabled class="messageHistory" rows="30" cols="120"
id="MessageHistoryBox"></textarea>
    <br/>
<input type="text" id="MessageBox" size="120"/>
</div>
</body>
<script src="../../js/sync.js" type="text/javascript"></script>
</html>
```

4. The contents of css file are

```
body {padding: 0; margin: 0; overflow: hidden;}

.message_area{
    padding-left: 60px;
```

```

}
.messageHistory{
    background-color: transparent;
    background: transparent;
}

```

5. The contents of js file

```

//send message when mouse is on message div and enter is hit
$("#messages").keyup(function(event){
    if(event.keyCode == 13){
        var msg=$("#MessageBox").val();
        //send to peer
        var data ={
            "msgcontent":msg
        }
        sendMessage(data);
        addMessageLog(msg);
        $("#MessageBox").val('');
    }
});

function addMessageLog(msg){
    //add text to text area for message log for self
    $("#MessageHistoryBox").text( $("#MessageHistoryBox").text() + '\n'+ 'you : '+ msg);
}

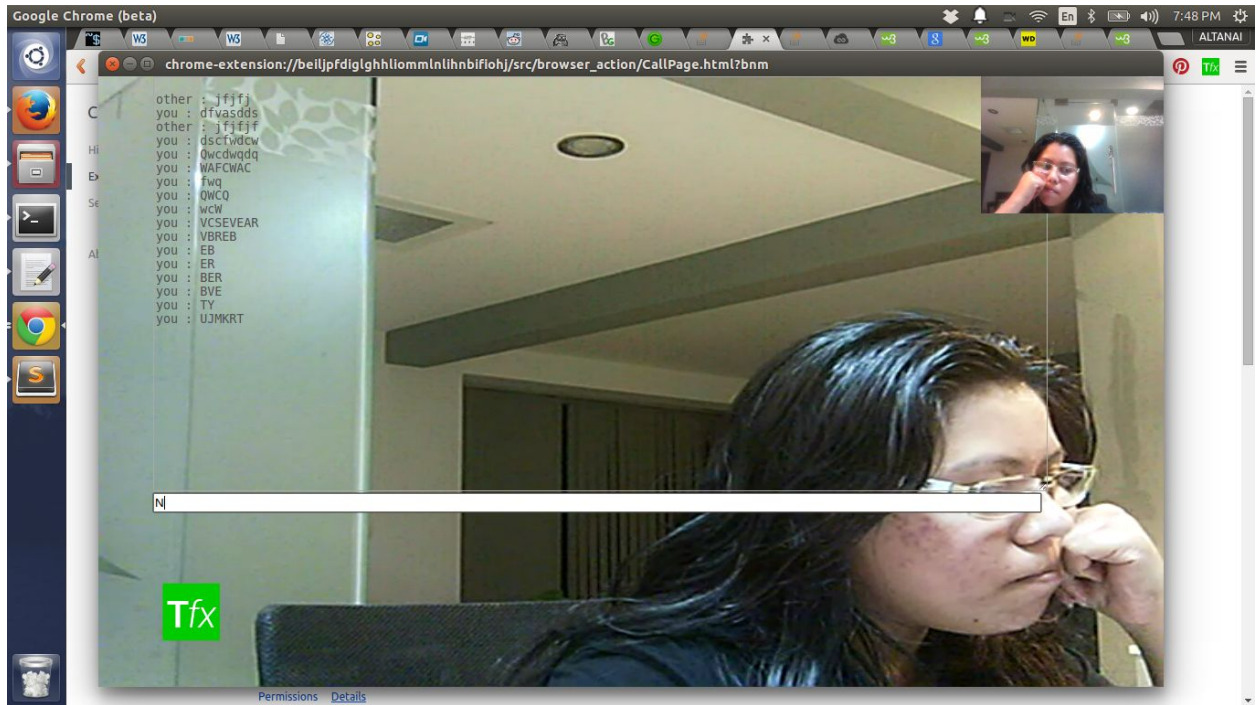
// handles send message
function sendMessage(message) {
    var widgetdata={
        "type":"plugin",
        "plugintype":"relaymsg",
        "action":"update",
        "content":message
    };
    // postmessage
    window.parent.postMessage(widgetdata,'*');
}

//to handle incoming message
function onmessage(evt) {
    //add text to text area for message log from peer
    if(evt.data.msgcontent!=null ){
        $("#MessageHistoryBox").text( $("#MessageHistoryBox").text() + '\n'+ 'other : '+
        evt.data.msgcontent );
    }
}

window.addEventListener("message",onmessage,false);

```

6. The end result is :



Developing a cross origin Widget (XHR)

Let us demonstrate the process and important points to create a cross- origin widget :

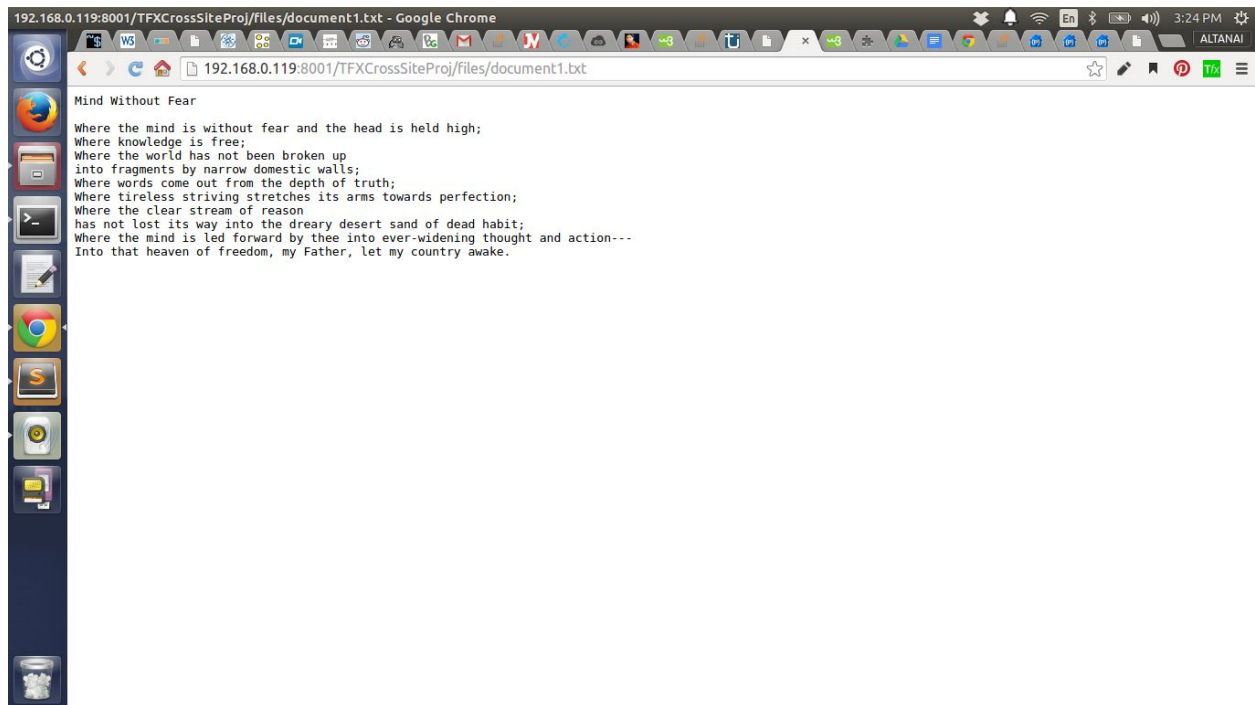
step 1 : Develop a separate web project and run it on a https

step 2 : Add the widget frame in TFX . Following is the code I added to make an XHR request over GET

```
var xmlhttp;
xmlhttp=new XMLHttpRequest();
xmlhttp.onreadystatechange=function()
{
    if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
        document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
    }
}
```

```
xmlhttp.open("GET","https://192.168.0.119:8000/TFXCrossSiteProj/files/document1.txt",true);
xmlhttp.send();
```

step 3 : Using self made https we have have to open the url separately in browser and give it explicit permission to open in advanced setting. Make sure the original file is visible to you at the widgets url .

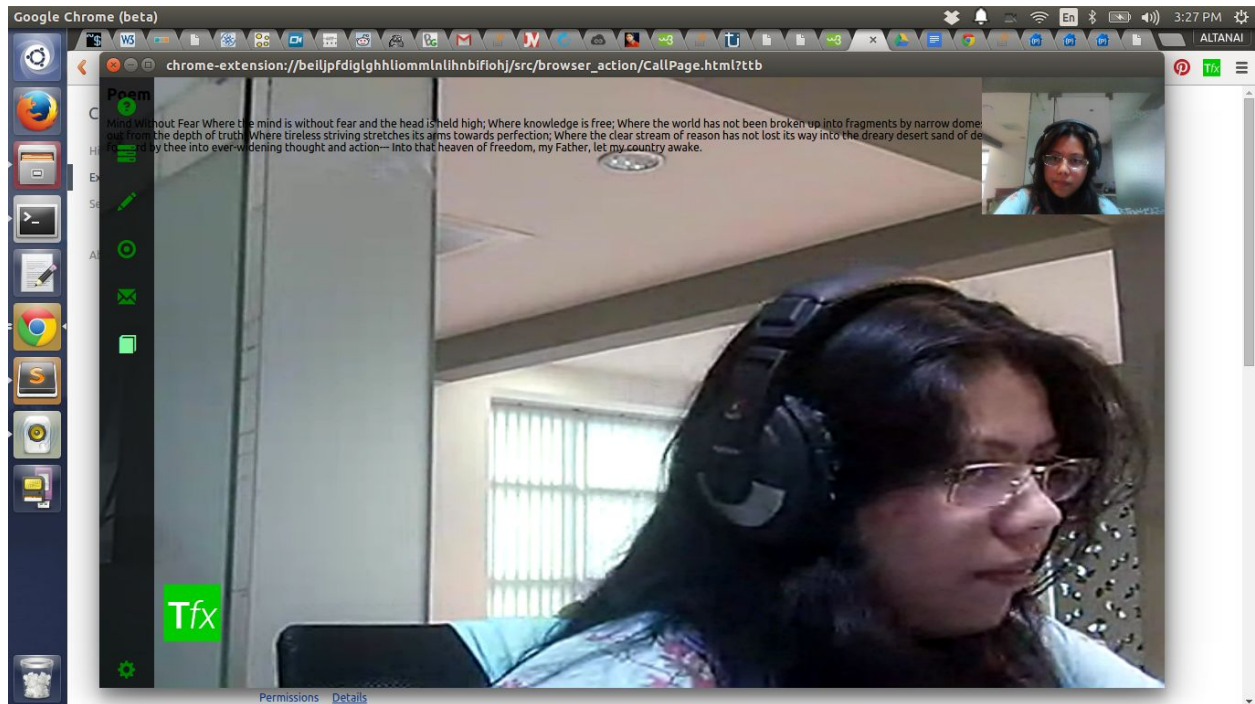


step 4: Adding permission to manifest for access the cross origin requests

```
"permissions": [  
  "tabs", "http://*.google.com/",  
  "https://192.168.0.119:8000/TFXCrossSiteProj"  
],
```

step 5 : Rest of the process are similar to develop a regular widget ie css and js .
Resulting screenshot

Resulting widget on TFX



Note 1 :

In absence of changes to manifest file the cross origin request is meet with a `Access-Control-Allow-Origin` error .

Note 2:

While using POST the TFX responds with Failed to load resource: the server responded with a status of 404 (Not Found)

Note 3:

Also if instead of https http is used the TFX still responds with Failed to load resource: the server responded with a status of 404 (Not Found)

.....