CWS
CallEvo Web Services Library

Library link:
https://assets.callevo.net/core/cws.latest.js

Last Review:
December, 2024

Last Version:
CWS 2.00 Alpha

Developed In:
JavaScript

CallEvo Web Services (CWS) is a web interface framework that allows easy and intuitive creation and implementation of an agent that consumes CallEvo® services (CallEvo Api™, CallEvo Communications™ and CallEvo Pepper™).

The main objective of CWS is to allow the customer to create an independent communication application, with the design and structure that fits the line of business, linked to all our services available for this task.

Content

***Note:** The sample code is made with HTML, CSS, jQuery, JavaScript. The purpose is to give you a guide of the code that you can implement.*

Instance

To create the instance with the CWS library you must generate a code as follows:

```
let cws = new CallEvoWebServices();
```

Registration & Initialization

settingsInit

Records the information obtained from the agent user, such as: id, name, agent code, among others *(See getAuth () Method)*.

```
let audit_settingsInit = cws.settingsInit(globalData);
```

Returns *true* or *false.*

settingDialBack

Register DialBack phone number.

```
let audit_settingDialBack = cws.settingDialBack(dialback);
```

The structure of the Dial Back is: *(dialback: 1 – active, 0 – inactive)*

```
let dialback = {
    dialback: 1,
    number: '7201234567',
}
```

addEventListener

Register the listener mode function that the agent (client) will use to receive all the communication from the CWS library.
The way to register is as follows:

```
cws.addEventListener(myListener);
```

conn

It is used to start the communication processes; it is used once the login and campaign registration process has been completed. It is used in the main screen that the agent will operate.

```
cws.conn();
```

The most important thing in this step is to verify that all of the above registration methods have *TRUE* as the resulting value (settingsInit, settingDialBack).

Parameters

keepalive

Keeps communication alive.
true or *false*. Default value is false

```
cws.keepalive = true;
```

debug

Show the messages to web console.
true or *false*. Default value is false

```
cws.debug = false;
```

numberLines

This is the number of lines you wish to activate. These lines allow the agent to interact in a call.
The default value is 2. If you want the lines to be increased automatically, the value will be 0 (zero).
Note that for each line a row must be added to the call data (see CALLDATA of the Listener).

```
        cws.numberLines = 2;
```

actual_line

Returns the current line number

```
let currentline = cws.actual_line;
```

Constants

Constants are defined to unify responses, requests and validations, and to standardize them. They are necessary to express a comparison or a condition.

The form of use is, for example, ***cws.ON***, replacing case "ON":

```

switch (e.result) {
    case cws.ON:
        break;
}
```

Are as follows:

ACTIVATE
ACTIONBUTTONS
ACTUALLINE
AGENT
AGENTCAMPAIGN
AGENTCONNECT
AGENTDUMP
AGENTLOGIN
AGENTLOGINTIMEOUT
AGENTLOGOFF
AGICONNECT
BATHROOM
BREAK
BREAKCONFERENCE
BRIDGE
ACTIONBUTTONS
CALLBACK
CALLDATA
CALLERHANGUP
CALLINFO
CAMPAIGN
CANCEL
CANCHANGEAGENTSTATUS
CHAT2AGENT
CLOSE
CLEAR
COMPLETEAGENT
COMPLETECALLER
CONF
CONFERENCE
CONFIRM
CONN
CONNECT
COUNTCALLBACK
DIALBACK
DISABLED
DISCONNECT
DOUBLECALL
ENTERCONF
ERROR
EXITCONF
FAIL
FAILDNC
FAILNOCAMP
FAILTZ
GOTOLINE
HANGUP
HOLD
IDCONNECTION
INFO
INFOCALL
KICKAGENT
LINES
LOG
LOGOFF
LUNCH
MANUALCALL
MEETING
NEW
NOKEY
NOTREADY
OFF
OK
ON
OTHER
OUTCALL
PHONE
PHONEMSG
PING
PONG
PREVIEWCALL
READY
RECALL
REGISTEROBJECTS
RETRIEVE
SCRIPTURL
SELECTROW
SENDDTMF
SETDISPHANG
SETDISPHANGALL
SETSTATUS
SKIP
STARTRECORDING
STOPRECORDING
SUCCESS
TALKING
TRANSFER
VISIBLE
WEBRTCMSG
WEBSOCKET
WRAP

Important	All constants are expressed in capital letters.

Api Methods

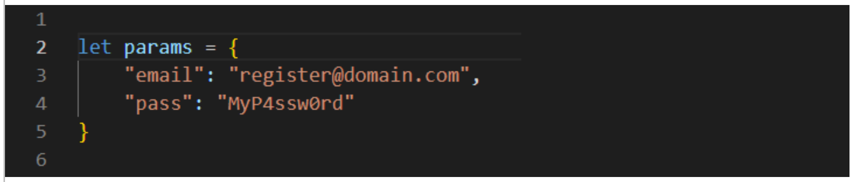
getAuth

Requests agent authentication via email and password. This method returns all the information of the registered user in JSON format.

This JSON contains all the information of the tenants registered with that email.

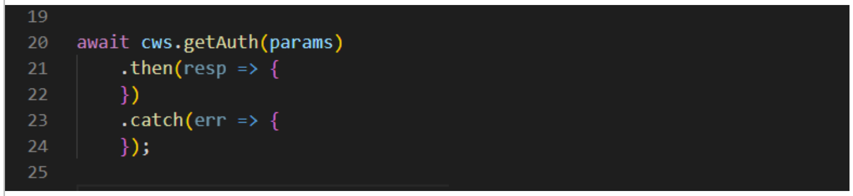
This information is used to present the agent with the option to select which tenant he/she wishes to work with.

Parameters to send:



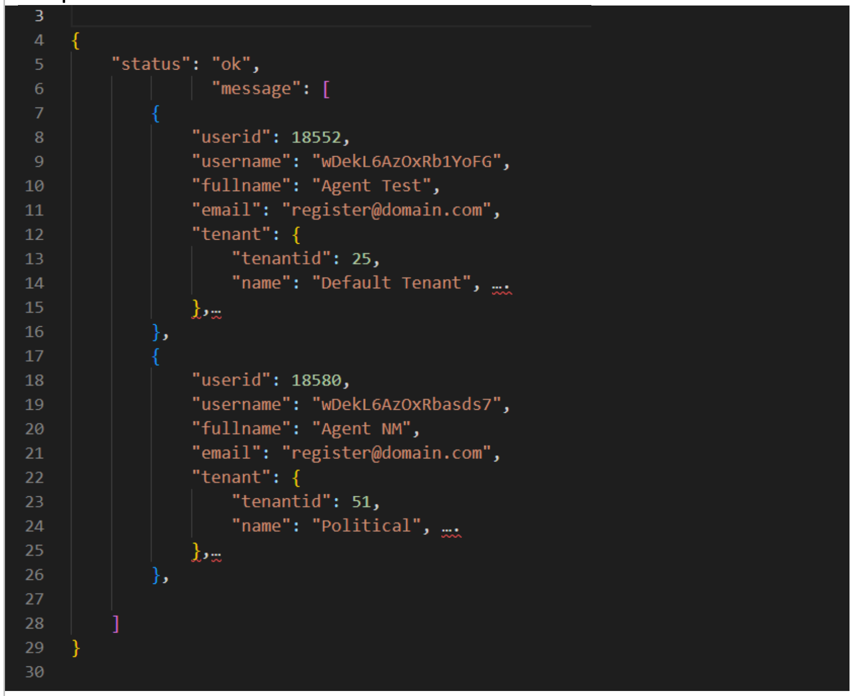
```
let params = {
    "email": "register@domain.com",
    "pass": "MyP4ssw0rd"
}
```

Format:



```
await cws.getAuth(params)
    .then(resp => {
    })
    .catch(err => {
    });
```

Response:



```
{
    "status": "ok",
            "message": [
        {
            "userid": 18552,
            "username": "wDekL6AzOxRb1YoFG",
            "fullname": "Agent Test",
            "email": "register@domain.com",
            "tenant": {
                "tenantid": 25,
                "name": "Default Tenant", …
            },…
        },
        {
            "userid": 18580,
            "username": "wDekL6AzOxRbasds7",
            "fullname": "Agent NM",
            "email": "register@domain.com",
            "tenant": {
                "tenantid": 51,
                "name": "Political", …
            },…
        },

    ]
}
```

You can create a dropdown object containing the ***userid*** and ***tenant name*** for the agent to select, as follows:

```

let html = "";
lstTenants.forEach(e => {
    html = `${html}<option value="${e.userid}">${e.tenant.name}</option>`
});
$("#slTenants").html(html);

```

Once we have selected the user and the tenant with which we want to enter the agent, we must use a registration method ***settingsInit***

```

lstTenants.forEach(e => {
    if (e.userid == userID){
        cws.settingsInit(e);
    }
});

```

getCampaignsTenant

Gets a list of campaigns that the agent has in the selected tenant to be able to register.

Format:

```

await cws.getCampaignsTenant()
    .then(resp => {
    })
    .catch(err => {
    });

```

Response

```

[
    {
        "camp_id": "31",
        "camp_name": "camptest2",
        "team": "teamone",
        "isselected": "false"
    },
    {
        "camp_id": "323",
        "camp_name": "inbound",
        "team": "teamone",
        "isselected": "false"
    },
    {
        "camp_id": "30",
        "camp_name": "message",
        "team": "teamone",
        "isselected": "false"
    },
```

From the list obtained we must present to the agent which campaigns to register (one or several)

The list of campaigns must be collected by the field camp_id and separated by commas, to be sent in the ***login*** registration method. Ex: “31,223,30,11,10,3” login

Registers the tenant's user in the selected campaigns to work with.

Format

```
let globalCamps = "31,323,30,11,10,13";
cws.login(globalCamps)
    .then(resp => {
    })
    .catch(err => {
    });
```

Response:

```
{
    "status": "OK",
    "action": "ADDLOGIN",
    "answer": "OK"
}
```

getManualCampaigns

Returns a list of manual campaigns in which you are registered.

Format:

```
await cws.getManualCampaigns()
    .then(resp => {
    })
    .catch(err => {
    });
```

Response:

```
[
    {
        "camp_id": "31",
        "camp_name": "camptest2",
    },
    {
        "camp_id": "11",
        "camp_name": "testmanual",
    },
]
```

getTransferAgentsData

Returns a list of agents for call transfer

Format:

```
await cws.getTransferAgentsData()
    .then(resp => {
    })
    .catch(err => {
    });
```

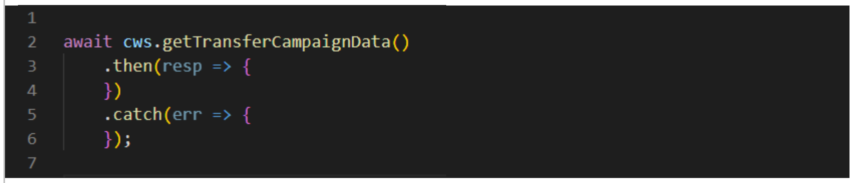
Response:

```
[
    {
        "data": "as4asd54as654d",
        "label": "Agent 2",
    },
    {
        "data": "u512s1aas514dq",
        "label": "Agent 2",
    },
]
```

getTransferCampaignData

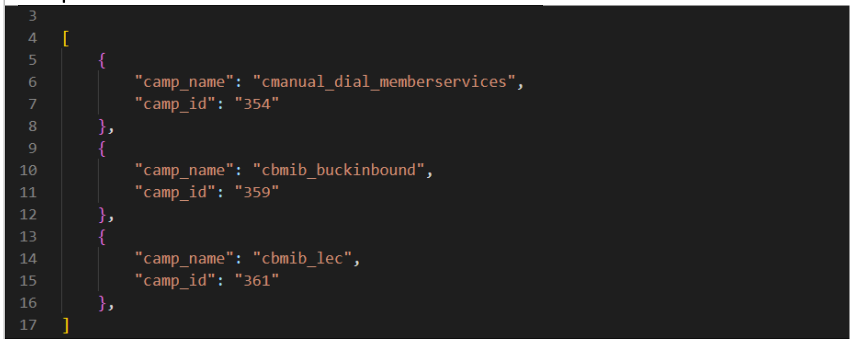
Returns a list of campaigns to transfer the call so that a free agent can answer.

Format:



```

await cws.getTransferCampaignData()
    .then(resp => {
    })
    .catch(err => {
    });

```

Response:



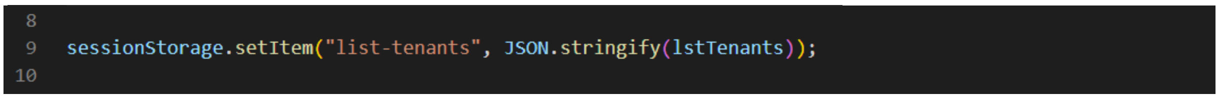
```

[
    {
        "camp_name": "cmanual_dial_memberservices",
        "camp_id": "354"
    },
    {
        "camp_name": "cbmib_buckinbound",
        "camp_id": "359"
    },
    {
        "camp_name": "cbmib_lec",
        "camp_id": "361"
    },
]
```

Session Storage examples

All data obtained as a result of the methods must be stored in local session variables or cookies, encrypted or not.

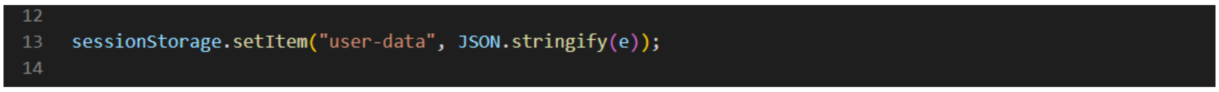
Example:

List of tenants



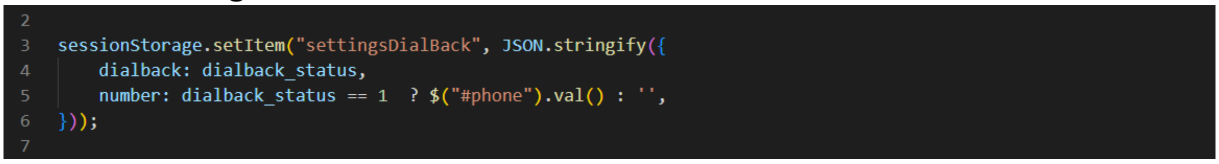
```
sessionStorage.setItem("list-tenants", JSON.stringify(lstTenants));
```

User selected



```
sessionStorage.setItem("user-data", JSON.stringify(e));
```

Dial back configuration



```
sessionStorage.setItem("settingsDialBack", JSON.stringify({
    dialback: dialback_status,
    number: dialback_status == 1  ? $("#phone").val() : '',
}));
```

General Methods

activeCalls

The number of currently active lines is obtained, i.e. lines that are in HOLD or TALKING mode.

```
let num_calls_active = cws.activeCalls();
```

onSend

It is a method that allows sending requests to the communications service. It is structured in two parts action and parameters:

```
cws.onSend(action, params);
```

Action | **Parameters**

OUTCALL

To make a manual call
Fields: camp_id, phone, type => "phone", leadId: 0 (default)

```
let params = {
    camp_id: parseInt($("#cmbCamp option:selected").val()),
    phone: $("#phonemanualcall").val(),
    type: "phone",
    leadId: 0,
};
cws.onSend('OUTCALL', params);
```

SETSTATUS

When the agent status dropdown (client side) is changed, the status must be sent to CWS for registration.

```
cws.onSend(cws.SETSTATUS, { status: cws.NOTREADY });
```

The status list of the agents will be loaded automatically when the object is registered and will depend on the status of the call.

CALLBACK

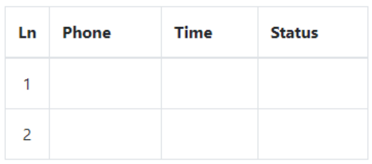
When selecting the phone number to be called, the ID to which the number belongs must be sent to the **onSend** method.

```
cws.onSend(cws.CALLBACK, { id: e.id });
```

selectRow

This method allows to communicate between the client side (HTML) and the CWS service, indicating that the user clicked on a line to visualize its information.

The agent must have a table in HTML with two or more rows and each of them will be a line.



Ln	Phone	Time	Status
1			
2			

Parameter is the whole row in an array [1,'','',''] or [2,'','',''] or [3,'','','']

If it is activated or not depending on the agent state, if it is in TALKING it cannot be changed in line with the exception of HOLD or NOTREADY.

Code example

```
var table = $('#tblCallData tbody').on("click", "tr", function () {
    let data = $(this)[0].outerText.split("\t");
    cws.selectRow(data);
});
```

Make sure that the **numberLines** parameter is defined with the lines you are setting here.

funcTransfer

This method sends a transfer request.

There are 3 types of transfer:

- To an agent
- To a campaign
- To a phone number

```
cws.funcTransfer(cws.AGENT, data);
cws.funcTransfer(cws.CAMPAIGN, data);
cws.funcTransfer(cws.PHONE, data);
```

AGENT

The data is obtained by calling the api method **getTransferAgentsData()**, and the field is data.

CAMPAIGN

The data is obtained by calling the api method **getTransferCampaignData(),** and the field is camp_id.

PHONE

The data is the content of an input box entered by the agent

funcHold

Sends a request to put the current line on HOLD or RETREIVE

```
cws.funcHold();
```

funcDTMF

Sends DTMF digits to a call.

```
let data = "*4";
cws.funcDTMF(data);
```

funcBridge

Activate the bridge mode between 2 active calls, you need to send as parameters the two active lines. Example :[1,2]

```
if (cws.activeCalls() == 2) {
  let lines = [];
  $(".checks").prop('checked', function () {
      let id = $(this).attr('value');
      lines.push(id);
  });
  cws.funcBridge(lines);
}
```

funcConference

Establishes a conference between one or two active lines. To perform this action, the call must be in HOLD or TALKING state.

```
cws.funcConference([1]);
cws.funcConference([1,2]);
```

funcDisconnect

Disconnects active conference calls

```
cws.funcDisconnect();
```

funcBreakConference

Disconnects active calls that are in conference by returning to the initial HOLD and TALKING state.

```
cws.funcBreakConference();
```

funcReCall

Call back to conference

```
cws.funcReCall();
```

funcHangUp

Send a hang up on the call

```
cws.funcHangUp();
```

onClose

Close all communications with CWS services.

```
cws.onClose();
```

sendPreviewCallResults

Sends the data selected by the agent when PREVIEW CALL was presented.

```
cws.sendPreviewCallResults(cws.CONFIRM);
cws.sendPreviewCallResults(cws.CANCEL);
cws.sendPreviewCallResults(cws.SKIP);
```

Listener

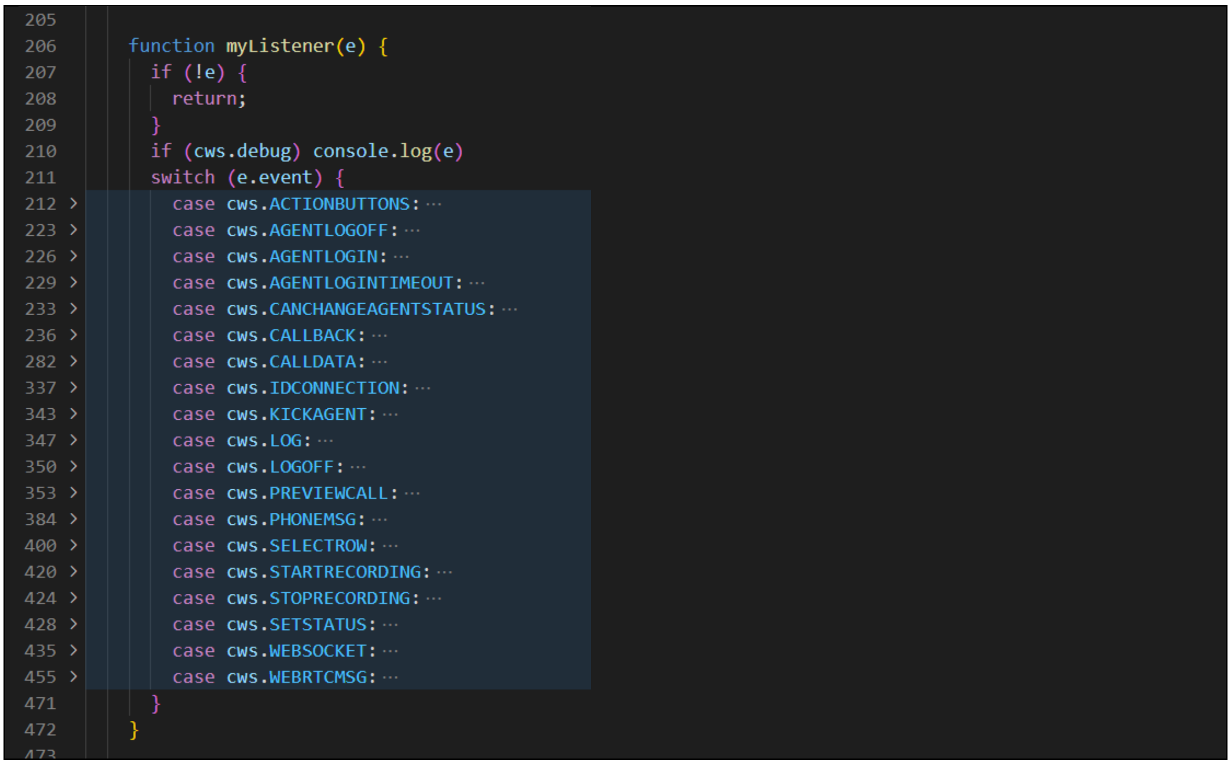
This listener receives all communication from the CWS library.

It will receive several events that present information about the operations performed: the current call, error data, actions to be performed.

How to register

```
cws.addEventListener(myListener);
```

Structure



```

function myListener(e) {
  if (!e) {
    return;
  }
  if (cws.debug) console.log(e)
  switch (e.event) {
    case cws.ACTIONBUTTONS: ⋯
    case cws.AGENTLOGOFF: ⋯
    case cws.AGENTLOGIN: ⋯
    case cws.AGENTLOGINTIMEOUT: ⋯
    case cws.CANCHANGEAGENTSTATUS: ⋯
    case cws.CALLBACK: ⋯
    case cws.CALLDATA: ⋯
    case cws.IDCONNECTION: ⋯
    case cws.KICKAGENT: ⋯
    case cws.LOG: ⋯
    case cws.LOGOFF: ⋯
    case cws.PREVIEWCALL: ⋯
    case cws.PHONEMSG: ⋯
    case cws.SELECTROW: ⋯
    case cws.STARTRECORDING: ⋯
    case cws.STOPRECORDING: ⋯
    case cws.SETSTATUS: ⋯
    case cws.WEBSOCKET: ⋯
    case cws.WEBRTCMSG: ⋯
  }
}
```

Events

ACTIONBUTTONS

The CWS library helps us to identify which buttons, (if included in the design), should be disabled or visible, by sending a list that includes the name of the reference and the value to apply (true/false).

The buttons are:
MANUALCALL, HOLD, RETRIEVE, TRANSFER, SENDDTMF, BRIDGE, CONF, DISCONNECT, BREAKCONFERENCE, RECALL, HANGUP.

```
JSON
{
    "event": "ACTIONBUTTONS",
    "action": "DISABLED/VISIBLE",
    "buttons": [
        {"name": "MANUALCALL", "value": true},
        {"name": "HOLD", "value": true},
        {"name": "RETRIEVE", "value": true},
        {"name": "TRANSFER", "value": true},
        {"name": "SENDDTMF", "value": true},
        {"name": "BRIDGE", "value": true},
        {"name": "CONF", "value": true},
        {"name": "DISCONNECT", "value": true},
        {"name": "BREAKCONFERENCE", "value": true},
        {"name": "RECALL", "value": true},
        {"name": "HANGUP", "value": true}
    ]
}
```

HTML example

```
<button id="btmLOGOFF">Log Off</button>
<button id="btmMANUALCALL">Manual Call</button>
<button id="btmHOLD">Hold</button>
<button id="btmRETRIEVE">Retrieve</button>
<button id="btmTRANSFER">Transfer</button>
<button id="btmSENDDTMF">Send DTMF</button>
<button id="btmBRIDGE">Bridge</button>
<button id="btmCONF">Conference</button>
<button id="btmDISCONNECT">Disconnect</button>
<button id="btmBREAKCONFERENCE">Break Conf</button>
<button id="btmRECALL">Recall</button>
<button id="btmHANGUP">Hang up</button>
```

Code example

```
case cws.ACTIONBUTTONS:
  e.buttons.forEach(btns => {
    if (e.action == cws.DISABLED) {
      $(`#btm${btns.name}`).attr("disabled", btns.value);
    } else if (e.action == cws.VISIBLE) {
      let val = btns.value ? "display: inline" : "display: none";
      let obj = `$("#btm${btns.name}").prop("style", "${val}");`
      eval(obj);
    }
  });
```

AGENTLOGIN

Informs the time when the agent is connected and login.
`JSON = {event:" AGENTLOGIN", result: "OK"}`

AGENTLOGOFF

Informs when the agent is disconnected and must exit to the main login screen.
`JSON = {event:" AGENTLOGOFF"}`

AGENTLOGINTIMEOUT

When it starts the connection with the communication services, a registration is produced that must be completed within 30 seconds, if not, it sends a time-out message and it is requested to exit to the main screen executing the logoff process.

```
JSON = {event: "AGENTLOGINTIMEOUT", message: ""}
```

CANCHANGEAGENTSTATUS

Within the disabled attribute (true/false) the CWS library indicates whether or not to allow the agent to change its state whether it is in or out of a call. The interaction of the agent with the drop down or object that has the following states: READY, NOTREADY, TALKING, BREAK, etc.

```
JSON = {event: "CANCHANGEAGENTSTATUS", disabled: true}
```

Code Example

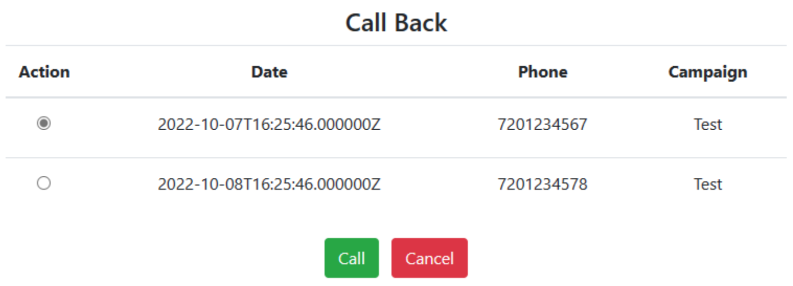
```
case cws.CANCHANGEAGENTSTATUS:
     $("#status_agent").attr("disabled", event.disabled)
     break;
```

CALLBACK

Receives the list of pending calls from the agent to call back. When the agent's status is NOTREADY, a list of phone numbers will be sent to the CALLBACK event for the agent to select which one to call. You must be able to decline the list and it will appear again when in NOTREADY status.

```
JSON = {event: "AGENTLOGINTIMEOUT", result: [
{id:  1,  phone:  "7201234567",  campaign:  "Test",  date:  "2022-10-
07T16:25:46.000000Z"},
{id:  2,  phone:  "7201234568",  campaign:  "Test",  date:  "2022-10-
08T16:25:46.000000Z"}, …
]}
```

Screen example



Call Back

Action	Date	Phone	Campaign
●	2022-10-07T16:25:46.000000Z	7201234567	Test
○	2022-10-08T16:25:46.000000Z	7201234578	Test

When selecting the phone number to be called, the ID to which the number belongs must be sent to the **onSend** method.

```
cws.onSend(cws.CALLBACK, { id: e.id });
```

CALLDATA

This action is essential for the agent to be updated on the phone, time, and status of each call. It constantly receives information about the status of the number of lines defined in the **numberLines** parameter.

```
JSON = {
        event: "CALLDATA",
        result: [
                {
                        id: 1,
                        phone: "",
                        time: "",
                        callstatus: ""
                },…
        ],
        operation: "NEW"
}
```

In the operation field we have 3 values: NEW, CLEAR, '' (blank space). NEW means that you need to add a new row in the calldata table. CLEAR means that you need to delete all rows and build again all rows, and the blank means that you need to update only the values.

Code Example

```
case cws.CALLDATA:
  if (e.result && e.result.length > 0) {
    if (e.operation == cws.NEW) {
      let idxnew = e.result[e.result.length - 1].id;
      let htmlnew = `
        <tr id="line${idxnew}">
            <td class="text-center"><b>${idxnew}</b></td>
            <td id="line${idxnew}_phone"></td>
            <td id="line${idxnew}_time"></td>
            <td id="line${idxnew}_status"></td>
            <td id="line${idxnew}_recording" class="hide">
               <i class="fas fa-microphone-alt mr-2 text-danger"></i>
            </td>
            <td id="line${idxnew}_uniqueid" class="hide"></td>
            <td id="line${idxnew}_url" class="hide"></td>
        </tr>`;
      $("#tblBody").append(htmlnew);
    } else if (e.operation == cws.CLEAR) {
      let html = "";
      let i = 0;
      e.result.forEach((e, idx) => {
        i = idx + 1;
        html = `${html}
          <tr id="line${i}">
              <td class="text-center"><b>${i}</b></td>
              <td id="line${i}_phone"></td>
              <td id="line${i}_time"></td>
              <td id="line${i}_status"></td>
              <td id="line${i}_recording" class="hide">
                 <i class="fas fa-microphone-alt mr-2 text-danger"></i>
              </td>
              <td id="line${i}_uniqueid" class=" hide"></td>
              <td id="line${i}_url" class="hide"></td>
          </tr>`;
        $(`#script${i}`).addClass("hide");
      })
      $("#tblBody").html(html);
    }

    e.result.forEach(ele => {
      $(`#line${ele.id}_phone`).html(ele.phone);
      $(`#line${ele.id}_time`).html(ele.time);
      $(`#line${ele.id}_status`).html(ele.callstatus);
      $(`#line${ele.id}_uniqueid`).html(ele.uniqueid);
      $(`#line${ele.id}_url`).html(ele.url);

      if (ele.phone == "") {
        $(`#script${ele.id}`).addClass("hide");
        $(`#script${ele.id}`).attr("src", "");
      } else {
        if (ele.id == cws.actual_line) {
          $(`#script${ele.id}`).removeClass("hide");
        } else {
          $(`#script${ele.id}`).addClass("hide");
        }
      }
    });
  }
  break;
```

IDCONNECTION

Receives the current connection ID.

```
JSON = {event: "IDCONNECTION", connectionid: ""}
```

KICKAGENT

When the administrator user needs to log out of the system, it sends a kick to the agent, and the agent needs to log out.

```
JSON = {event: "KICKAGENT", message: "You have been kick"}
```

LOG

Receives feedback on actions performed that may or may not be displayed on the screen. **INFO** / **WARN** / **ERROR** / **»** *(send message)* / **«** *(receive message)*.

```
JSON = {event: "LOG", result: "INFO", message: "Test"}
```

LOGOFF

Indicates that the agent should enter a logoff process, and should return to the initial login screen.

```
JSON = {event: "LOGOFF", message: ""}
```

PREVIEWCALL

Preview call is an event that occurs in PREVIEW type campaigns and it sends a call to the agent one without clicker, the only condition is that it is in READY state.

The application must present the agent with a screen that indicates if he/she wants to take, skip or cancel the call.

```
JSON = {event:"", camp_id: 0, phone:"", leadid=0, option=""}
```

```
case cws.PREVIEWCALL:
  previewcall = e;
  let phone = e.phone;
  html = `
      <div class="text-center">
          <h4>Preview Call</h4>
          <h1>${phone}</h1>
          <button id="btmPreviewCall" class="btn btn-success m-1">Call</button>
          <button id="btmPreviewCancel" class="btn btn-danger m-1">Cancel</button>
          <button id="btmPreviewSkip" class="btn btn-warning m-1">Skip</button>
      </div>
  `;
  $("#question").html(html);
  $("#question").removeClass("hide");

  $("#btmPreviewCall").on("click", function () {
    $("#question").addClass("hide");
    previewcall.option = cws.CONFIRM;
    cws.sendPreviewCallResults(previewcall);
  });
  $("#btmPreviewCancel").on("click", function () {
    $("#question").addClass("hide");
    previewcall.option = cws.CANCEL;
    cws.sendPreviewCallResults(previewcall);
  });
  $("#btmPreviewSkip").on("click", function () {
    $("#question").addClass("hide");
    previewcall.option = cws.SKIP;
    cws.sendPreviewCallResults(previewcall);
  });
  break;
```

PHONEMSG

The moment you connect to the messaging service COMMUNICATION LINE you will receive ON / OFF / ERROR as a response.

```
JSON = {event: "PHONEMSG", result: ""}
```

```
case cws.PHONEMSG:
  $("#phone_ligth").html(e.result == cws.ON ? cws.ON : cws.OFF);
  $("#phone_ligth").removeClass("text-danger");
  switch (e.result) {
    case cws.ON:
      $("#phone_ligth").addClass("text-success");
      break;
    case cws.OFF:
      $("#phone_ligth").addClass("text-danger");
      break;
    case cws.ERROR:
      $("#phone_ligth").addClass("text-danger");
      log(e.type, e.message);
      break;
  }
  break;
```

SELECTROW

Indicates which line is being used for visual highlighting in the application. You will receive an integer greater than zero, it is recommended to display the corresponding script.
`JSON = {event: "SELECTROW", row: 1}`

```
case cws.SELECTROW:
  $("#tblCallData .tr-selected").toggleClass("tr-selected");
  $("#line" + e.row).addClass("tr-selected");
  $("#actualline").html(e.row);

  /** HIDE ALL SCRIPTS */
  for (let i = 0; i < cws.numberLines; i++) {
    $(`#script${i}`).addClass("hide");
  }

  /** VERIFY IF THE SCRIPT IF LOADED*/
  let verify_url = $(`#script${e.row}`).attr("src");
  if (verify_url == undefined || verify_url == '') {
    if (e.url != "") {
      $(`#script${e.row}`).attr("src", e.url);
    }
  }
  /** SHOW THE CURRENT SCRIPT */
  $(`#script${e.row}`).removeClass("hide");
  break;
```

STARTRECORDING

Indicates that recording has started in the line ID=n (ex: 1).
`JSON = {event: "STARTRECORDING", result: "OK", id: 1}`

```
$(`#recording_header`).removeClass("hide");
$(`#line${e.id}_recording`).removeClass("hide");
```

STOPRECORDING

Indicates that the recording has been stopped in the line ID=n.
`JSON = {event: "STOPRECORDING", result: "OK", id:1}`

```
$(`#recording_header`).addClass("hide");
$(`#line${e.id}_recording`).addClass("hide");
```

SETSTATUS

Sends the agent status to be displayed, and the list of statuses that can be displayed in a drop down.

```
JSON
{
    "event": "SETSTATUS",
    "status": "NOTREADY",
    "list_status": [
        "READY",
        "NOTREADY",
        "BREAK",
        "BATHROOM",
        "MEETING",
        "LUNCH",
        "OTHER"
    ]
}
```

Code example

```
case cws.SETSTATUS:
  let html = "";
  e.list_status.forEach(st => {
    html = `${html}<option value='${st}' ${st==e.status
?'selected':''}>${st}</option>`;
  });
  $("#status_agent").html(html);
  break;
```

WEBRTCMSG

The moment you connect to the WEB RTC messaging service you will receive ON / OFF / ERROR as a response.

```
JSON = {event: "WEBRTCMSG", result: ""}
```

Code Example

```
case cws.WEBRTCMSG:
  $("#webrtc_ligth").html(e.result == cws.ON ? cws.ON : cws.OFF);
  $("#webrtc_ligth").removeClass("text-danger");
  switch (e.result) {
    case cws.ON:
      $("#webrtc_ligth").addClass("text-success");
      break;
    case cws.OFF:
      $("#webrtc_ligth").addClass("text-danger");
      break;
    case cws.ERROR:
      $("#webrtc_ligth").addClass("text-danger");
      break;
  }
  break;
```

WEBSOCKET

The moment you connect to the WEBSOCKET messaging service you will receive ON / OFF / CLOSE / ERROR.

```
JSON = {event: "WEBSOCKET", result: ""}
```

```
case cws.WEBSOCKET:
  $("#pepper_ligth").removeClass("text-danger");
  switch (e.result) {
    case cws.ON:
      $("#pepper_ligth").html(e.result);
      pepper_status = e.result == cws.ON ? true : false;
      $("#pepper_ligth").addClass("text-success");
      break;
    case cws.CLOSE:
    case cws.OFF:
    case cws.ERROR:
      $("#pepper_ligth").html(e.result);
      $("#pepper_ligth").addClass("text-danger");
      pepper_status = false;
      if (onLogOff) goToLogIn();
      break;
  }
  break;
```

Getting Started

Links to required libraries

External library required in the HTML document header

https://code.jquery.com/jquery-3.6.1.min.js
https://assets.callevo.net/core/webrtc.stable.js
https://assets.callevo.net/core/cws.latest.js
https://assets.callevo.net/core/adapter.min.js
https://assets.callevo.net/core/md5.min.js

Optional
https://cdn.usebootstrap.com/bootstrap/4.4.1/js/bootstrap.min.js
https://cdn.usebootstrap.com/bootstrap/4.4.1/css/bootstrap.min.css
https://cdn.usebootstrap.com/bootstrap/4.4.1/js/bootstrap.bundle.min.js
https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.29.4/moment-with-locales.min.js

The implementation of the CWS library, presents a very easy and safe to use philosophy, since it integrates all the services that CallEvo offers in one place.

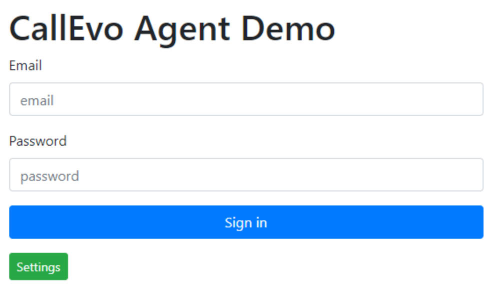
This library saves many hours of development and understanding about the development of a communications application and the compatibility with each service.

Now we can develop applications in less time and with the same effectiveness of the official application.

In this guide we will use
HTML / CSS
Java Script / JQuery

Index.html

We can divide the file into 4 main divs
[visible] Sign In
[hidden] Dial Back
[hidden] Tenants
[hidden] Campaigns



Sign In
Function: getAuth ()
Response: List of the tenants

In this screen we need some html labels (objects):

Input = text	Email
Input = text	Password
Input = radio	Dial back activation *(1-Yes and 0-No options)*
Input = text	Dial back phone
Button	Sing In

Ok, let's go
First, in the <script> section we build a new instance with the CWS library.

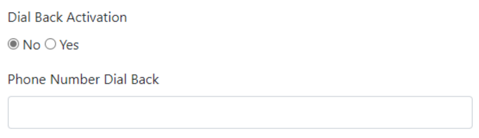
```
let cws = new CallEvoWebServices();
```

Regardless of what you have selected in DIALBACK, it must be saved for use when entering the agent. Session variables are recommended. The JSON structure is as follows:

```
{
     dialback: 0,
     number: “”,
}
```

Example code:

```
let cws = new CallEvoWebServices();
let params = {
    "email": $("#email").val(),
    "pass": $("#pass").val()
}
signIn(params);

async function signIn(params) {
    lstTenants = [];
    await cws.getAuth(params)
        .then(resp => {
            if (resp.status == cws.OK) {
                lstTenants = resp.message;
            } else {
                console.log("Error", resp.message);
                return;
            }

            if (lstTenants.length > 0) {
                lstTenants.sort((a, b) =>
                    a.tenant.name.localeCompare(b.tenant.name)
                );
                sessionStorage.setItem("list-tenants", JSON.stringify(lstTenants));

                let html = "";
                lstTenants.forEach(e => {
                    html = `${html}<option value="${e.userid}">${e.tenant.name}</option>`
                });
                $("#slTenants").html(html);
                $("#singin").addClass("hide");
                $("#tenants").removeClass("hide");
            } else {
                alert("You don't have access");
            }
        })
        .catch(err => {
            console.log(err)
        });
}
```

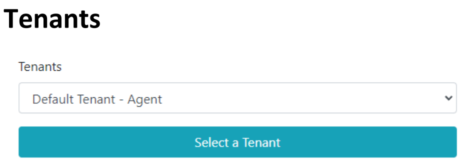

Dial Back Activation

[✓ No] Yes

Phone Number Dial Back

Dial Back

When you click on the Configuration button, the dial back div appears.

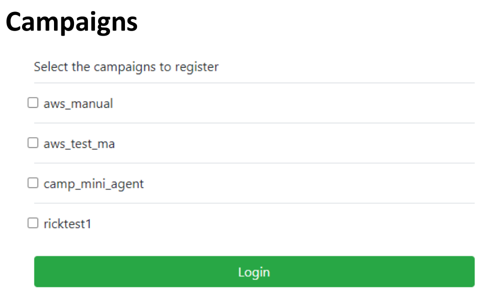

Tenants

Tenants

Default Tenant - Agent

You must build a dropdown with the list of tenants with two fundamental fields: <userid> in the value and tenant name <tenant.name>.

Once the userid of the tenant to be logged in is selected, it should request a list of the campaigns that tenant and user have active, (see getCampaignsTenant), and show the client several checkbox objects for selection.


Campaigns

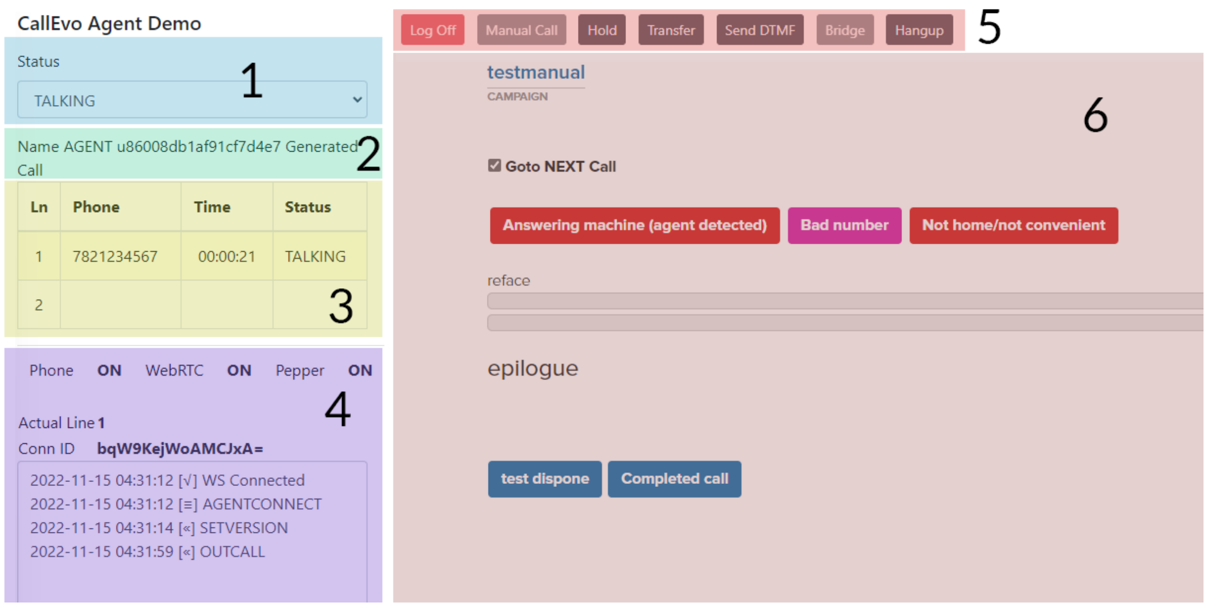
Select the campaigns to register

- ☐ aws_manual
- ☐ aws_test_ma
- ☐ camp_mini_agent
- ☐ ricktest1

This is a list of campaigns that the agent can select to register in the communication center to be able to make calls or receive calls according to the team to which these campaigns belong.

Once the campaigns are selected, all the collected information must be passed to the file that contains the operational agent (Ex: agent.html).

Agent.html


CallEvo Agent Demo

Status

TALKING

Name AGENT u86008db1af91cf7d4e7 Generated Call

Ln	Phone	Time	Status
1	7821234567	00:00:21	TALKING
2			

Phone ON WebRTC ON Pepper ON

Actual Line 1

Conn ID bqW9KejWoAMCJxA=

```
2022-11-15 04:31:12 [√] WS Connected
2022-11-15 04:31:12 [≡] AGENTCONNECT
2022-11-15 04:31:14 [«] SETVERSION
2022-11-15 04:31:59 [«] OUTCALL
```

HTML Code example

```
<body>
  <div class="container-fluid">
    <div class="row">
      <div class="col-3">
        <h5 id="support">CallEvo Agent Demo</h5>
      </div>
      <div class="col-9 text-right">
        <button id="btmLOGOFF">Log Off</button>
        <button id="btmMANUALCALL">Manual Call</button>
        <button id="btmHOLD">Hold</button>
        <button id="btmRETRIEVE">Retrieve</button>
        <button id="btmTRANSFER">Transfer</button>
        <button id="btmSENDDTMF">Send DTMF</button>
        <button id="btmBRIDGE">Bridge</button>
        <button id="btmCONF">Conference</button>
        <button id="btmDISCONNECT">Disconnect</button>
        <button id="btmBREAKCONFERENCE">Break Conf</button>
        <button id="btmRECALL">Recall</button>
        <button id="btmHANGUP">Hangup</button>
      </div>
    </div>
    <div class="row">
      <div class="col-3">
        <div class="form-group" class="form-caption">
          <label for="">Status</label>
          <select id="status_agent" class="form-control"></select>
        </div>

        <div id="callinfo" class="hide"></div>

        <table class="table">
          <tr>
            <td>Phone</td>
            <th id="phone_ligth" class="text-danger">OFF</th>
            <td>WebRTC</td>
            <th id="webrtc_ligth" class="text-danger">OFF</th>
            <td>Pepper</td>
            <th id="pepper_ligth" class="text-danger">OFF</th>
          </tr>
        </table>
        <table id="tblCallData" class="table table-stripeds table-bordereds" cellspacing="0">
          <thead>
            <tr>
              <th title="Line" width="5%" class="text-center">Ln</th>
              <th>Phone</th>
              <th>Time</th>
              <th>Status</th>
              <th id="recording_header" class="hide"></th>
              <th class="hide">UniqueID</th>
            </tr>
          </thead>
          <tbody id="tblBody"></tbody>
        </table>

        <table class="tables">
          <tr>
            <td>Actual Line</td>
            <th id="actualline" class="text-dark"></th>
          </tr>
          <tr>
            <td>Conn ID</td>
            <th id="connectionid" class="text-dark"></th>
          </tr>
        </table>
        <textarea id="response" class="form-control" rows="10"></textarea>
      </div>
      <div id="bodyscript" class="col-9">
        <div id="question" class="hide"></div>

        <div id="divScript"></div>
      </div>
      <div id="divOnlyForTest" class="col-3 hide"></div>
    </div>
  </div>
```

The screen distribution is:

1
Agent status dropdown

ID: #status_agent

HTML code

```
<div class="form-group" class="form-caption">
       <label for="">Status</label>
       <select id="status_agent" class="form-control"></select>
</div>
```

Event Change

```
$("#status_agent").on("change", function () {
       let newstatus = $(this).val();
       cws.onSend(cws.SETSTATUS, { status: newstatus });
});
```

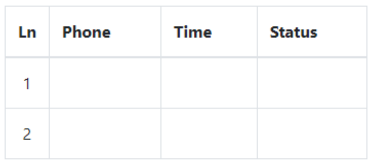
2
Call Info

It is all the information of a call of the current line.

HTML code

```
<div id="callinfo" class="hide"></div>
```

3
Call Data



Ln	Phone	Time	Status
1			
2			

It is a table of two or more rows (*must be defined in the parameter **numberLines***), to show the call data, such as: phone, time and status.

It can be distributed as a table, div or the HTML object of your choice according to your design.

These objects will be constantly interacting with the CALLDATA action in the receiveListener.

This group of objects are responsible for giving correct information to the agent.

HTML code

```html
<table id="tblCallData" class="table table-stripeds table-bordereds">
    <thead>
        <tr>
            <th title="Line" width="5%" class="text-center">Ln</th>
            <th>Phone</th>
            <th>Time</th>
            <th>Status</th>
            <th id="recording_header" class="hide"></th>
            <th class="hide">UniqueID</th>
        </tr>
    </thead>
    <tbody id="tblBody"></tbody>
</table>
```

In the receive listener event of the "CALLDATA" action (cws.CALLDATA), you can place this code JavaScript code (example)

```javascript
case cws.CALLDATA:
if (e.result && e.result.length > 0) {

    if (e.operation == cws.NEW) {
        let idxnew = e.result[e.result.length-1].id ;
        let htmlnew = `
            <tr id="line${idxnew}">
                <td class="text-center"><b>${idxnew}</b></td>
                <td id="line${idxnew}_phone" class="text-start"></td>
                <td id="line${idxnew}_time" class="text-center"></td>
                <td id="line${idxnew}_status" class="text-start"></td>
                <td id="line${idxnew}_recording" class="hide"><i class="fas fa-
microphone-alt mr-2 text-danger"></i></td>
                <td id="line${idxnew}_uniqueid" class="text-start hide"></td>
            </tr>`;
        $("#tblBody").append(htmlnew);
    }else if (e.operation == cws.CLEAR)   {
        let html = "";
        let i = 0;
        e.result.forEach((e,idx)=>{
            i = idx + 1;
            html = `${html}
                <tr id="line${i}">
                    <td class="text-center"><b>${i}</b></td>
                    <td id="line${i}_phone" class="text-start"></td>
                    <td id="line${i}_time" class="text-center"></td>
                    <td id="line${i}_status" class="text-start"></td>
                    <td id="line${i}_recording" class="hide"><i class="fas fa-
microphone-alt mr-2 text-danger"></i></td>
                    <td id="line${i}_uniqueid" class="text-start hide"></td>
                </tr>`;
        })
        $("#tblBody").html(html);
    }
    e.result.forEach(ele => {
        $(`#line${ele.id}_phone`).html(ele.phone);
        $(`#line${ele.id}_time`).html(ele.time);
        $(`#line${ele.id}_status`).html(ele.callstatus);
        $(`#line${ele.id}_uniqueid`).html(ele.uniqueid);
    });
}
break;
```

4 Aditional Info

Various flags and information about the agent's operation are obtained before, during and after the call. These may or may not be displayed to the agent.

We have 3 services to connect within CWS:
Pepper (messaging), Phone and WebRTC (voice, audio, line, client, communication).

Constants that receive this information in the receive listener:

Phone	cws.PHONEMSG
WebRTC	cws.WEBRTCMSG
Pepper	cws.WEBSOCKET
Active Lines	cws.SELECTROW
Log	cws.LOG
Connection ID	cws.IDCONNECTION

5 Operation Buttons

The operation buttons allow you to control the actions within each call.

Manual Call
Hold
Retrieve
Transfer
Send DTMF
Bridge
Conference
Disconnect
Break Conference
Recall
Hang up

HTML code:

```
<button id="btmLOGOFF">Log Off</button>
<button id="btmMANUALCALL">Manual Call</button>
<button id="btmHOLD">Hold</button>
<button id="btmRETRIEVE">Retrieve</button>
<button id="btmTRANSFER">Transfer</button>
<button id="btmSENDDTMF">Send DTMF</button>
<button id="btmBRIDGE">Bridge</button>
<button id="btmCONF">Conference</button>
<button id="btmDISCONNECT">Disconnect</button>
<button id="btmBREAKCONFERENCE">Break Conf</button>
<button id="btmRECALL">Recall</button>
<button id="btmHANGUP">Hangup</button>
```

Manual Call	Make a manual call.

An HTML code must be built to allow the agent to select the manual campaign and enter the phone number to make the call.

To get the active manual campaigns you must execute the Api Method getManualCampaigns ().

Once the camp_id and phone number are obtained, you must send the manual call request to

```
let params = {
  camp_id: 30,
  phone: “7201234567”,
  type: "phone",
  leadId: 0,
};
cws.onSend('OUTCALL', params);
```

Hold	Puts an active call on hold

```
cws.funcHold();
```

Retrieve	Retrieve a call in Hold

```
cws.funcHold();
```

Transfer	Transfer the current call to an agent, campaign or phone number.

To make a call there are a few steps to follow:

1. Know if there are active and free agents to transfer. This is obtained with the api method **getTransferAgentsData**.
2. To know if there are active campaigns to transfer. When a transfer is sent to a campaign, it takes the first free and active agent found. This is obtained by calling a **getTransferCampaignData** api method.
3. To a phone number.

It is important to present the agent with the options Agent, Campaign and Phone Number, depending on the selection, the following should be sent.

In the case of a call to a specific agent:
`cws.funcTransfer(cws.AGENT, data);`

	In the case of a transfer to a campaign `cws.funcTransfer(cws.CAMPAIGN, camp_id);` And in the case of a phone number `cws.funcTransfer(cws.PHONE, phone);`
Send DTMF	Send DTMF tones `cws.funcDTMF("*142");`
Bridge	Bridges between two active calls `cws.funcBridge([1,2]);`
Conference	Conference calls between calls `cws.funcConference([1,2]);`
Disconnect	Disconnect active calls `cws.funcDisconnect();`
Break Conference	Breaks the conference by returning the TALKING and HOLD lines to their original state. `cws.funcBreakConference();`
Recall	Make a call again `cws.funcReCall();`
Hang up	Hang up an active call `cws.funcHangUp();`

6
Script Section

In this area, when there is a call, is where the agent will be shown a script where he can fill in the information obtained from the client. This script is assigned in the campaign.

HTML code

```
<div id="bodyscript" class="col-9">
    <div id="question" class="hide"></div>
    <div id="divScript"></div>
</div>
```

The #question div we are using (in this example) is to send all the html code that is built from the client-side agent, to show some screens like: transfer, campaigns, among others.

Example
```
$("#btmSENDDTMF").on("click", function () {
      let html;
      html = `
          <div class="text-center mb-3">
              <h4>Manual Call</h4>
              <div class="form-group">
                  <label>Send DMTF</label>
              </div>
              <div class="form-group">
                  <label>Please enter a DTMF [0-9][*][#]</label>
                  <input type="text" id="senddtmf"  class="form-control"/>
              </div>
              <button id="btmDTMFSend">Send</button>
              <button id="btmDTMFCancel">Cancel</button>
          </div>
      `;
      $("#question").html(html);
      $("#question").removeClass("hide");
});
```

Returning to the creation and management of scripts, we can do it automatically or manually. Automatically it is when in the parameter ***cws.numberLines*** we define in 0 (Zero), and manually when we put a number equal or greater than 2.

We create a function that allows us to create the scripts we need depending on the number of lines defined, dynamically.

```
function addScripts(createAll = true) {
   let html = "";
   if (createAll) {
      for (let i = 1; i <= cws.numberLines; i++) {
         html = `${html} <iframe id="script${i}" sandbox="allow-same-origin
allow-scripts allow-popups allow-forms"  class="hide" scrolling="auto"
allowfullscreen frameborder="0"></iframe>`;
      }
      $("#divScript").html(html);
   } else {
      let id = cws.numberLines;
      html = `${html} <iframe id="script${id}" sandbox="allow-same-origin
allow-scripts allow-popups allow-forms"  class="hide" scrolling="auto"
allowfullscreen frameborder="0"></iframe>`;
      $("#divScript").append(html);
   }
}
```

To call this function we only have to take into account two processes, true when we load the screen, or close all calls and false when new lines are added (only in case of ***cws.numberLines = 0***).

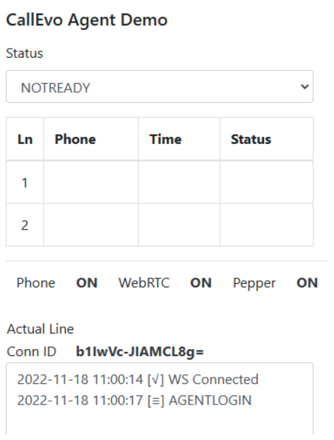
Example: *true*
```
     let cws = new CallEvoWebServices();
     cws.keepalive = true;
     cws.debug = false;
     cws.numberLines = 0;
     cws.addEventListener(myListener);
     let audit_settingsInit = cws.settingsInit(globalData);
     let audit_settingDialBack = cws.settingDialBack(dialback);
     addScripts(true);
```

Example: *false*

```
case cws.CALLDATA:
          if (e.result && e.result.length > 0) {
            if (e.operation == cws.NEW) {
              let idxnew = e.result[e.result.length - 1].id;
              let htmlnew = `
                <tr id="line${idxnew}">
                    <td class="text-center"><b>${idxnew}</b></td>
                    <td id="line${idxnew}_phone" class="text-start"></td>
                    <td id="line${idxnew}_time" class="text-center"></td>
                    <td id="line${idxnew}_status" class="text-start"></td>
                    <td id="line${idxnew}_recording" class="hide"><i
class="fas fa-microphone-alt mr-2 text-danger"></i></td>
                    <td id="line${idxnew}_uniqueid" class="text-start
hide"></td>
                    <td id="line${idxnew}_url" class="text-start hide"></td>
                </tr>`;
              $("#tblBody").append(htmlnew);
              addScripts(false);  ←------------------- HERE
```

How to know that everything is OK:

1. If the library is correctly instantiated, in the first getAuth() method it will give you results.
2. When you enter the agent, after you do the registration and login process. It will show the agent status in NOTREADY, Phone, WebRTC and Pepper in ON, Conn ID with its respective serial.
3. The "Manual Call" button will be the only one enabled.



CallEvo Agent Demo

Status

NOTREADY

Ln	Phone	Time	Status
1			
2			

Phone **ON** WebRTC **ON** Pepper **ON**

Actual Line

Conn ID **b1IwVc-JIAMCL8g=**

```
2022-11-18 11:00:14 [√] WS Connected
2022-11-18 11:00:17 [≡] AGENTLOGIN
```