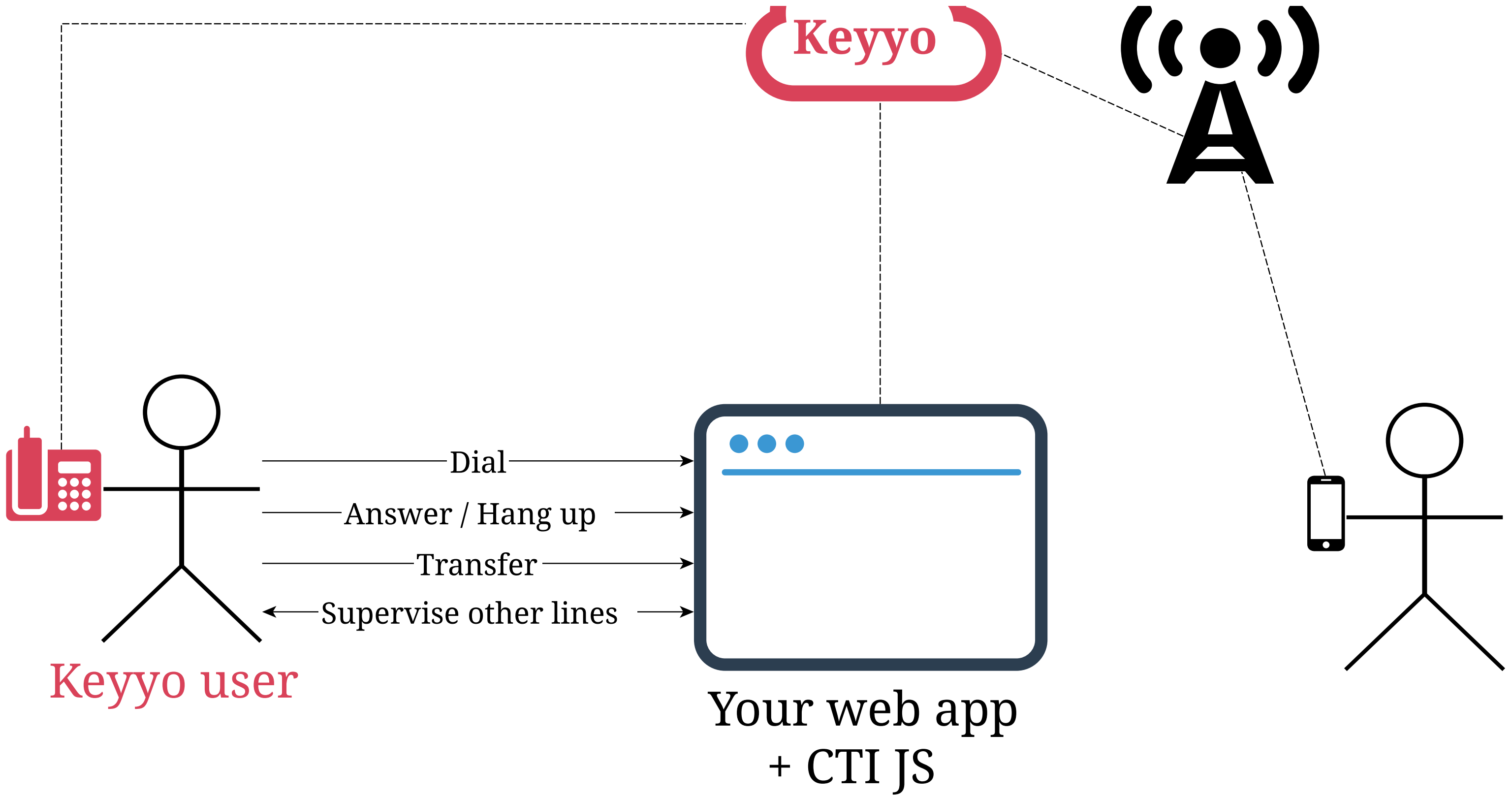
- MANAGER API
- CLIENT-SIDE API
- SERVER-SIDE API
- OAuth2
- API for PMS (Property Management System)
- Contact us

Overview

Introduction

CTI (Computer Telephony Integration) JS is a JavaScript library that allows you to supervise and control Keyyo lines.



Basic steps

- Register your web app
- Get an access token (you can refer to the OAuth2 documentation)
- Retrieve a CSI token with the access token (you can refer to the CSI token sample on GitHub and the Manager API documentation)
- Use the CTI JS library with the CSI token to connect to Keyyo, and to supervise and control the line(s)

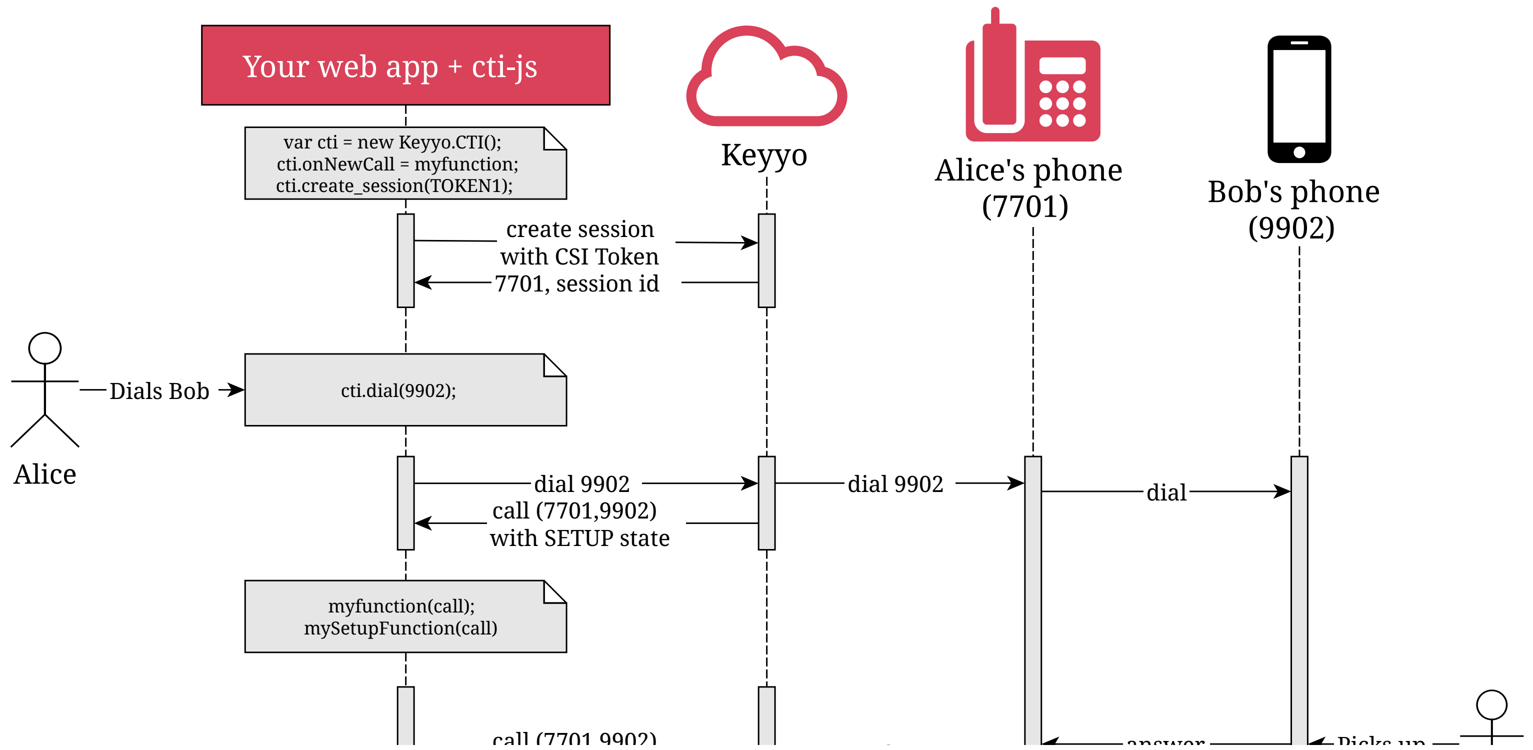
Scenarios

Two people, Alice (number: 7701) — a Keyyo user — and Bob (number: 9902) — a friend of Alice — want to communicate over the phone.

Alice uses a headset microphone and uses her phone remotely thanks to your web app using Keyyo CTI.

Outgoing call scenario

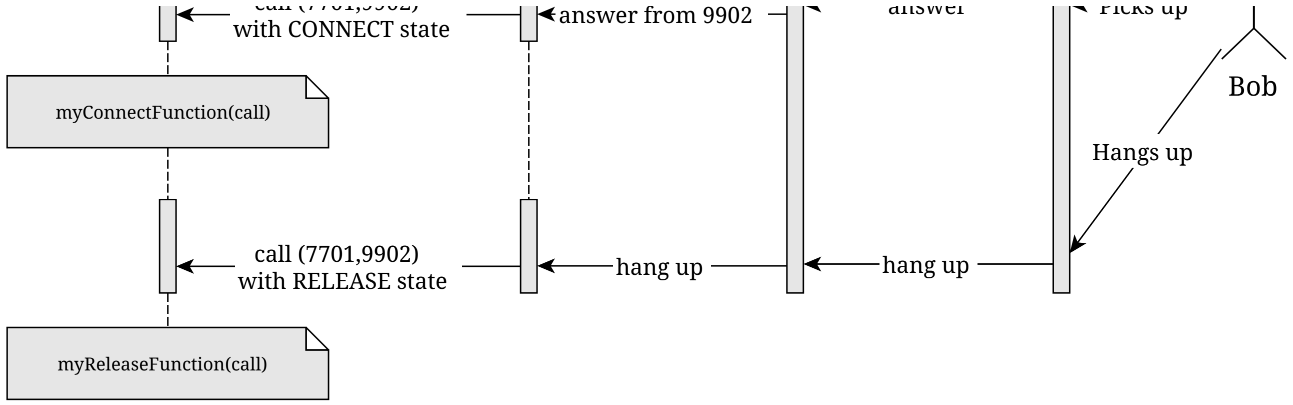
- Alice uses her browser to call Bob.
- Bob picks up his phone.
- Your web app displays additional info in Alice's browser.
- They speak.
- Bob hangs up.
- Alice sees on her phone and in her browser that the call has ended.



```
var cti = new Keyyo.CTI();
cti.onNewCall = myfunction;
cti.create_session(TOKEN1);
```

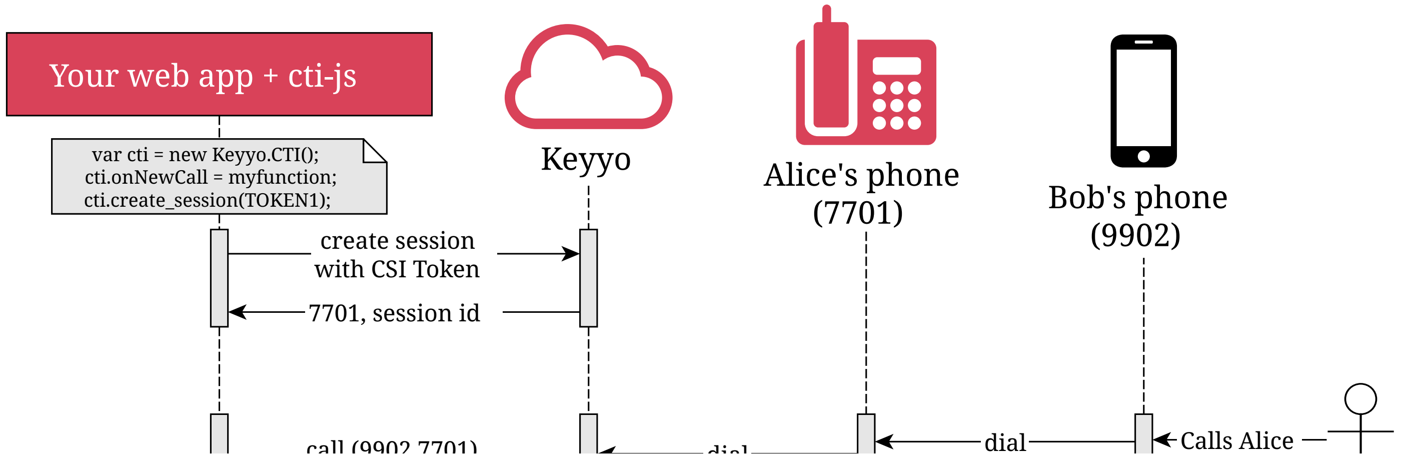
```
cti.dial(9902);
```

```
myfunction(call);
mySetupFunction(call)
```

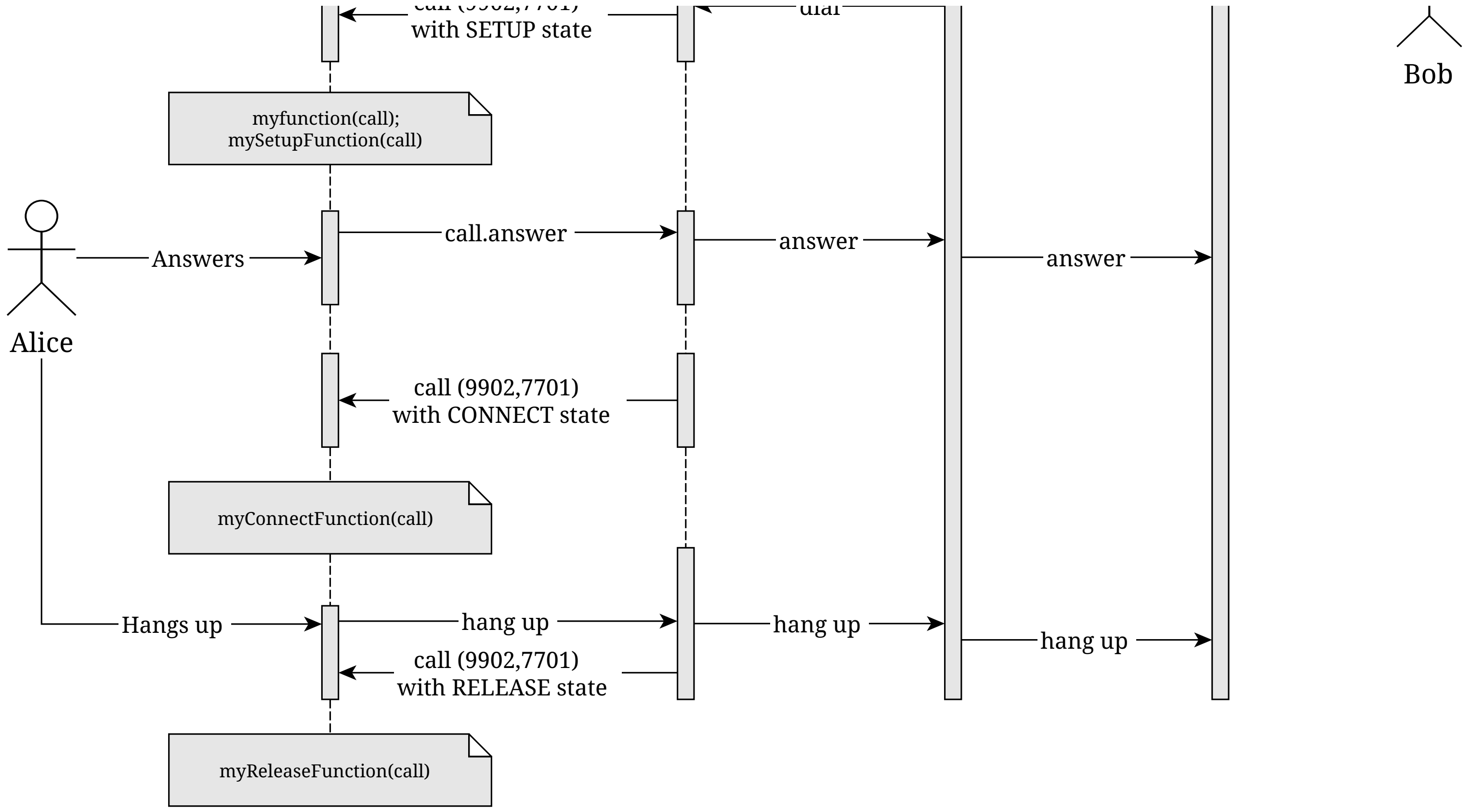


Incoming call scenario

- Bob calls Alice.
- Alice answers her phone.
- Alice sees the call info in her browser.
- They speak.
- Alice uses her browser to hang up.



```
var cti = new Keyyo.CTI();
cti.onNewCall = myfunction;
cti.create_session(TOKEN1);
```



Keyyo CTI JavaScript SDK

The JavaScript SDK for Keyyo CTI is a library that allows you to easily interact with your Keyyo telephony in your client-side web application.

Requirements

In order to use this library, you must be registered on Keyyo developer portal and have declared an application.

Installation

Include `keyyo-cti.min.js` in the `<head>` tag of your HTML.

HTML

```html
<script type="text/javascript" src="https://api.keyyo.com/libs/keyyo-cti/1.1/keyyo-cti.mi
```

Once the library is included, initialize Keyyo CTI and use the `create_session` method to authenticate you with your CSI token. It's an access token linked to a CSI, valid during 1 hour. With this token, you can use the Keyyo CTI API.
For more details on how to retrieve a CSI token, you can refer to the CSI token sample on GitHub and the Manager API documentation.

JS

```js
// Initialize Keyyo CTI
var cti = new Keyyo.CTI();
cti.create_session("<Your CSI token>", function(err, res) {

        if (err) {
                console.log(err);
                return;
        }

        console.log("connected");
});
```

Basic usage

All API methods use the same logic. By calling the method, you set parameters if needed and retrieve the response through a callback.

The response contains two parameters: `error` and `result`. Only one of them is set: the first one if an error occurred, otherwise the second one.

JS

```js
var cti = new Keyyo.CTI();
cti.method_name(params..., function(error, result) {
        if (error) {
                // Handle error
                console.log(error);
                return;
        }

        // Success
        console.log(result);
});
```

Sessions

Before calling an API method, you must be connected.
A connection is established when you successfully call the create_session or restore_session method.
Either way, a connection is associated to a session.

- When a session is created, an internal counter is set server-side to 1.
- When a subscription is done, it is persistently stored inside the session server-side: you will find your subscriptions back when you restore a session.
- When a websocket is closed, the session is automatically closed.
- Each time a session is closed, an internal counter is decremented server-side.
- Each time a session is restored, an internal counter is incremented server-side.
- When this internal counter reaches 0, an internal timer of 5 minutes is launched server-side.
  - If the timer expires, the session is automatically destroyed
  - If the session is restored before the timer expires, the timer is stopped
- When a session is (manually or automatically) destroyed, all info related to this session are destroyed server-side and the session is not usable anymore.

Subscriptions

When a user creates a session with a CSI token, the session is associated to the CSI (Common Service Identifier) of the user. An implicit subscription is made to the corresponding Keyyo line, so you are notified of all call events related to this line (see Handle call events).

If your organization has more than one Keyyo line (i.e. more than one CSI), you can choose to be also notified of the calls related to other lines of your organization. To be notified of the calls related to another line of your organization, you have to explicitly subscribe to the related number.

You can also subscribe to the special `callpark` number. If you subscribe to `callpark`, you will be notified of all the current calls that have been parked.

API

CTI

Constructor

Options

`cookie_auto`	`boolean`	By default this value is set to `true`, if you want to handle the session yourself, set this option to false
`cookie_name`	`string`	The cookie name if `cookie_auto` is true
`cookie_expires`	`integer`	The session expiration, in days, if `cookie_auto` is true

Result

object	An instance of CTI

JS

```
var options = {
                cookie_name : "my_session_name",
                cookie_expires: 365
```

```
        }
        var cti = new Keyyo.CTI(options);
```

Create session

Function

`create_session`	Create a session with your CSI token

Arguments

`csi_token`	Your CSI token. To retrieve a CSI token, refer to the CSI token sample on GitHub and the Manager API documentation.
`auto_answer (optional)`	Auto-answer's state of the whole session, by default, the value is true

Result

`number`	`string`	CSI linked to the token
`now`	`timestamp`	Server time in seconds
`session_id`	string	The session id

JS

```
cti.create_session("<Your CSI token>", function(err, res) {

        if (err) {
                console.log(err);
                return;
        }

        console.log("connected");
});
```

Restore session

Function

`restore_session`	Restore your session. Use it when you refresh your web page, or when you reopen your browser.

Arguments

`csi_token`	Your CSI token. To retrieve a CSI token, refer to the CSI token sample on GitHub and the Manager API documentation.
`session_id (optional)`	Your session ID

Result

`number`	string	CSI linked to the token
`now`	timestamp	Server time in seconds

JS

```
// Basic usage
// The session is recreated if cookie_auto is not set to false
cti.restore_session(csi_token, function(err, res) {

        if (err) {

                // Invalid CSI token (expires or not exists)
                if (err.status_code == 400) {
                        // Refresh your CSI token
                }
                return;
        }
        console.log(res);
});

// Or if you handle the session yourself, because cookie_auto is set to false
cti.restore_session(csi_token, "your_session_id", function(err, res) {

        if (err) {

                // Invalid CSI token (expires or not exists)
                if (err.status_code == 400 && err.error_code == "invalid_csi_token") {
                        // Refresh your CSI token
                } else if (err.status_code == 400 && err.error_code == "missing_parameter'
                        // Your session id is missing as parameter
                } else if (err.status_code == 401) {
                        // Your session is invalid
                        // Create a new session
                }
                return;
        }

        console.log(res);
});
```

Destroy session

Function

`destroy_session`	Close the connection with the server and destroy the session.

Arguments

Result

JS

```
cti.destroy_session(function(err, res) {
        if (err) {
                console.log(err);
                return;
        }

        console.log("logout");
});
```

Subscribe

Function

`subscribe`	Subscribe to a number

Arguments

`number`	`string`	Number to subscribe (International format), or "callpark"

Result

`integer`	Returns 1 if it's a success

JS

```
cti.subscribe("33172587948", function(err, res) {
        if (err) {
                console.log(err);
                return;
        }

        console.log(res);
});
```

Unsubscribe

Function

`unsubscribe`	Unsubscribe from a number

Arguments

`number`	`string`	Number to unsubscribe (International format), or “callpark”

Result

`integer`	Returns 1 if it's a success

JS

```js
cti.unsubscribe("33172587948", function(err, res) {
        if (err) {
                console.log(err);
                return;
        }

        console.log(res);
});
```

Get subscriptions

Function

`get_subscriptions`	Retrieve your subscriptions

Result

`numbers`	`array`	List of your subscriptions

JS

```
cti.get_subscriptions(function(err, res) {

        if (err) {
                console.log(err);
                return;
        }

        res.numbers.forEach(function(number) {
                console.log(number);
        });
});
```

Set auto-answer's state

Function

`set_auto_answer`	Enable or disable auto-answer of callee when making or transferring a call for the whole session

Arguments

`auto_answer`	`boolean`	Auto-answer state (true: enabled, false: disabled)

Result

`integer`	Returns 1 if it's a success

JS

```
cti.set_auto_answer(true, function(err, res) {
        if (err) {
                console.log(err);
                return;
        }
        console.log(res);
});
```

Get auto-answer's state

Function

`get_auto_answer`	Get the auto-answer's state of the session

Result

`auto_answer`	`boolean`	Auto-answer state (true: enabled, false: disabled)

JS

```
cti.get_auto_answer(function(err, res) {
        if (err) {
                console.log(err);
                return;
        }
        console.log(res.auto_answer);
});
```

Dial number

Function

`dial`	Call a number

Arguments

`number`	`string`	Number to call (International format)

Result

`integer`	Returns 1 if it's a success

JS

```js
cti.dial("33172587948", function(err, res) {

        if (err) {
                console.log(err);
                return;
        }

        console.log(res)

});
```

Send message

Function

`send_message`	Send a SIP message

Arguments

`number`	Number of the recipient (International format)
`message`	Text to send

Result

`integer`	Returns 1 if it's a success

JS

```
cti.send_message("33172587948", "The message...", function(err, res) {
        if (err) {
                console.log(err);
                return;
        }

        console.log(res);
});
```

Get calls

Function

`get_calls`	Get active, made, received and missed calls

Arguments

No parameters required

Return

`array`	An array of Call objects

JS

```
var calls = cti.get_calls();
calls.forEach(function(call){

        // Display active calls
        if (call.type != "RELEASE") {
                console.log(call);
        }
})
```

Get call

Function

`get_call`	Get a specific call

Arguments

callref	string	Call identifier

Return

`object`	Call object

JS

```
var call = cti.get_call("<Call Ref Id>");
```

Handle call events

Event

`onNewCall`	Handle a new call event

Result

`call`	`Object`	A Call object is passed as parameter to the callback

JS

```
cti.onNewCall = function(call) {
        call.onSetup = function() { }
        call.onConnect = function() { }
        call.onRelease = function() { }
        call.onMissed = function() { }
}
```

Attributes

`connected`	`boolean`	Result true if you are connected

Errors

Errors can be retrieved for each API methods by using the callback, if an error occurred the first parameter contains the following properties :

Properties

`status_code`	`integer`	HTTP status code
`message`	`string`	The error message
`error_code`	`integer`	Error code

Errors codes

HTTP status code	Error code	Description
`400`	`invalid_csi_token`	Your CSI token is invalid or has expired
`400`	`missing_parameter`	A required parameter is missing
`400`	`invalid_params`	Invalid parameter format
`401`	`invalid_session_id`	Your session id is invalid or has expired

JS

```js
cti.create_session("Your CSI token", function(err, res) {
        // Handle error if err is set
        if (err) {
                console.log(err.status_code, err.message);
                return;
        }
});
```

CALL

Setup

Event

`setup`	Fired when `callee`'s phone is ringing

JS

```
call.onSetup = function() {
        // Handle setup
}
```

Connect

Event

`connect`	Fired when `callee` has picked up his phone

JS

```
call.onConnect = function() {
        // Handle connect
}
```

Release

Event

`release`	Fired when the call has ended

JS

```
call.onRelease = function() {
        // Handle release
}
```

Missed

Event

`missed`	Fired when the call has ended and `callee` never picked up his phone

JS

```
call.onMissed = function() {
        // Handle missed
}
```

Answer

Function

`answer`	Answering a call

Arguments

No parameters required

Result

`integer`	Returns 1 if it's a success

JS

```
call.answer(function(err, res) {
        if (err) {
                console.log(err);
                return;
        }
        console.log(res);
});
```

Hang up

Function

hang_up	Terminate a call

Arguments

No parameters required

Result

`integer`	Returns 1 if it's a success

JS

```
call.hang_up(function(err, res) {

        if (err) {
                console.log(err);
                return;
        }

        console.log(res);
});
```

Reject

Function

`reject`	Reject an incoming call to the callee's voicemail (only if the callee is the session's number and if the call has not been picked ud or ended)

Arguments

No parameters required

Result

`integer`	Returns 1 if it's a success

JS

```
call.reject(function(err, res) {

        if (err) {
                console.log(err);
                return;
        }

        console.log(res);
        console.log(res);
});
```

Transfer

Function

`transfer`	Transfer a call to another recipient

Arguments

`call_side`	`string`	"callee" or "caller" that you want to transfer
`to`	`string`	Number of the new recipient (International format)

Result

`integer`	Returns 1 if it's a success

JS

```
call.transfer("caller", "33172587949", function(err, res) {
        if (err) {
                console.log(err);
                return;
        }
        console.log(res);
});
```

Make a supervised transfer

Function

`supervised_transfer`	Call the callee first and merge the two calls only if the callee answers

Arguments

`call_side`	`string`	"callee" or "caller" that you want to transfer
`to`	`string`	Number of the new recipient (International format)
`invite_timer (optional)`	`integer`	Duration (in seconds) before cancelling the transfer if the callee does not answer, by default this value is 0 (no cancelling)

Result

`integer`	Returns 1 if it's a success

JS

```
call.supervised_transfer("caller", "33172587949", 30, function(err, res) {

        if (err) {
                console.log(err);
                return;
        }

        console.log(res);
});
```

Merge

Function

`merge`	Merge two calls

Arguments

`call_side`	`string`	callee" or "caller" that you want to merge"
`second_call`	`object`	Second call object to merge
`second_call_side`	`string`	callee" or "caller" of `call` parameter"

Result

`integer`	Returns 1 if it's a success

JS

```js
// Merge the callee of the first call, and merge it with the caller of another call
call.merge("callee", call, "caller", function(err, res) {

        if (err) {
                console.log(err);
                return;
        }

        console.log(res);
});
```

Pause

Function

`pause`	Pause an ongoing call (only if the session's number if one of the peers) by parking it to a specific callpark slot : the session's number. Can be unpaused by unparking the parked call

Arguments

No parameters required

Result

`integer`	Returns 1 if it's a success

JS

```
call.pause(function(err, res) {

        if (err) {
                console.log(err);
                return;
        }

        console.log(res);
});
```

Park

Function

`park`	Park a call

Arguments

`side`	`string`	"callee" or "caller" that you want to park"
`slot (optional)`	`integer`	Slot where you want to park the call, by default this value is -1

Result

`integer`	Returns 1 if it's a success

JS

```
call.park("callee", function(err, res) {

        if (err) {
                console.log(err);
                return;
        }

        console.log(res);
});
```

Unpark

Function

`unpark`	Unpark a call

Arguments

No parameters required

Result

`integer`	Returns 1 if it's a success

JS

```
call.unpark(function(err, res) {

        if (err) {
                console.log(err);
                return;
        }

        console.log(res);
});
```

Get ringing duration

Function

`get_ringing_duration`	Get ringing duration

Arguments

No parameters required

Return

`integer`	Ringing duration in seconds

JS

```
var timer = null;
var call = cti.get_call("<Call Ref Id>")
clearInterval(timer);
timer = setInterval(function() {
        console.log(call.get_ringing_duration());
}, 1000);
```

Get duration

Function

`get_duration`	Get call duration

Arguments

No parameters required

Return

`integer`	Call duration in seconds

JS

```
var timer = null;
var call = cti.get_call("<Call Ref Id>")
clearInterval(timer);
timer = setInterval(function() {
        console.log(call.get_duration());
}, 1000);
```

Attributes

`callref`	`string`	Call identifier
`caller`	`string`	The international number of the caller
`callee`	`string`	The international number of the recipient (which received the call)
`state`	`string`	"SETUP" : The callee's phone is ringing "CONNECT" : The callee has picked up his phone "RELEASE" : The call has ended "MISSED" : The call has ended and the callee never picked up his

		phone
`setup_date`	`timestamp`	Call setup date in seconds
`connect_date`	`timestamp`	Call connect date in seconds
`release_date`	`timestamp`	Call realease date in seconds
`callpark_slot`	`string`	If the call is parked, contains the slot. Useful for watching specific slot(s) and unparking