

JavaScript API

The *INVOX Medical JavaScript API* reference explains in detail each function offered, as well as the code with usage examples. Functions are grouped as follows:

Group	Description
Essential functions	Essential functions to implement the basic functionality of INVOX Medical SDK.
Dictation functions	Functions related to dictation.
Status bar functions	Functions related to status bar.
Writer functions	Functions related to text writing operations.
Other functions	Functions related to get information and session.
Customizing components	Functions to customize INVOX Medical SDK components.
Local microphone driver	Available microphone controls and functions for customizing them.
Voice commands	Voice commands available and functions to customize them.
Capture internal events	Set of events triggered during the session.
Adaptation functions	Functions related to Recognition Model Adaptation process.
Dictionary functions	Functions to manage INVOX Medical Dictionary Repository.
Templates functions	Functions to manage INVOX Medical Templates Repository.
Transformations functions	Functions to manage INVOX Medical Transformations Repository.
Logger functions	Functions to use and manage INVOX Logger.

Basic Concepts

The following elements are **not available as a web component**, but you can create your own through the customization functions. **They are the pieces that make up the functionality of INVOX Medical.**

INVOX Medical is made up of a set of **elements to provide feedback to the user.**

Before you start using the API, you should understand the purpose of each element and how to integrate it into your web application.

Progress Bar

The Progress Bar **displays the download percentage** of required resources **and messages of the user authentication process.**

Status Bar

The Status Bar **displays messages about the current session.** Among these are messages related to the state of the recognizer, the authentication process or possible errors with the dictation service.

Audio Level Indicator

The Audio Level Indicator, or *VUMeter*, **displays feedback from the microphone** to the user. In this way, the user will be able to verify that audio is being received correctly.

Recognizer

The Recognizer takes care of recognition when it is in the active state. **This element works like a microphone that can be turned on or off**, this means that it can be picking up the user's audio to turn it into text or pause it.

So it has two states:

- **Active:** the recognizer is listening to the audio.
- **Paused:** the recognizer is stopped and is not listening to audio.

Hypothesis Viewer

The Hypothesis Viewer is used to display the **audio-to-text translation process performed by the recognition engine**.

During dictation, it allows to observe in real time the evolution of the recognized text. This translation is called *hypothesis*, and five types are distinguished: partial, rejected, command, macro and accepted.

TextWriter

INVOX Medical must know how to perform write operations on an editor or any target element.

The TextWriter **is responsible for performing the basic operations in a text editor**. It works as a proxy that references the editor instance and allows INVOX Medical to type the recognized text, move through the text using voice commands, select words or entire lines, delete text, redo, undo, etc.

For example, if our web application uses CKEditor5 then there must be a TextWriter that knows how to perform the basic operations in this particular editor.

Login

Function	Description	Parameters	Returns
Login(credentials, connectionConfig)	Starts connection with INVOX Medical Service.	Object, Object	Promise<void>

Description: This function allows you to login user into INVOX Medical Service using a username and password. This function also establish communication with the Dictation Service.

Parameters:

- **credentials**: Object. Contains the user credentials.
 - **user**: String. Contains the user's name.
 - **password**: String. Contains the user's password.
- **connectionConfig**: Object. Contains the parameters necessary to establish the connection with the Dictation Service.
 - **host**: String. Sets the host.
 - **port**: String. Sets the port.
 - **useDictationService**: Boolean. Sets whether the connection is made on the Remote Dictation Service or via the Local Dictation Service.

Returns:

- Successful connection promise. **Promise<void>**. The promise indicates successful connection, not successful login.

Usage:

```
const credentials = {
  user: "example",
  password: "example",
}
const connectionConfig = {
  host: "example.cloud.net",
  port: "8443",
  useDictationService: true
}

INVOX.Login(credentials, connectionConfig)
  .then(() => alert("Successful connection!"))
  .catch((e) => console.error(e));
```

Logout

Function	Description	Parameters	Returns
Logout()	Ends connection with INVOX Medical Service.	-	Promise<void>

Description: This function allows you to logout from INVOX Medical Service. This function ends the communication with the Dictation Service.

Parameters: No parameters required.

Returns:

- Successful disconnection promise. `Promise<void>`. The promise indicates successful disconnection.

Usage:

```
INVOX.Login()
  .then(() => console.log("Successful login!"))
  .catch((e) => console.error(e));
```

SwitchDictation

Function	Description	Parameters	Returns
SwitchDictation()	Allows you to turn off or on the speech recognition.	-	void

Description: This function allows you to turn off speech recognition when dictating and to turn on when is paused.

Parameters: No parameters required.

Returns: No value returned.

Usage:

```
INVOX.SwitchDictation();
```

SetTextWriter

Function	Description	Parameters	Returns
SetTextWriter(textWriter)	Allows you to set the specific TextWriter of an editor.	Object	void

Description:

Depending on the editor you use in your web application you need to make use of a specific TextWriter.

The function allows you to set the TextWriter implementation. INVOX Medical provides the following TextWriters implementations:

Target to write	TextWriter to use
TextArea	INVOX.TextAreaTextWriter
Others	Not implemented

Parameters:

- *textWriter*: Object. TextWriter implementation of the editor to use.

Returns: No value returned.

SetWriterTarget

Function	Description	Parameters	Returns
SetWriterTarget(editorInstance)	Allows you to set the target editor by its identifier.	Object	void

Description: The function allows you to configure the current editor (Writer Target) where INVOX Medical manages and writes the recognized text.

Parameters:

- *editorInstance*: Object. Editor instance we want to set as target.

Returns: No value returned.

Usage:

```
const ckeditorInstance = ...;
INVOX.SetWriterTarget(ckeditorInstance);
```

SetLang

Function	Description	Parameters	Returns
SetLang(lang)	Sets the language of the information.	String	void

Description:

The default language is “en-US”.

This function allows you to change the language of the session messages and the information displayed in the INVOX Medical Web Components.

Parameters:

- **lang:** String. Indicates the language. Supported languages:
 - `INVOX.lang.en_US`: ‘en-US’
 - `INVOX.lang.es_ES`: ‘es-ES’
 - `INVOX.lang.pt_PT`: ‘pt-PT’
 - `INVOX.lang.pt_BR`: ‘pt-BR’

Returns: No value returned.

Usage:

```
INVOX.SetLang(INVOX.lang.en_US);
```

GetLangs

Function	Description	Parameters	Returns
GetLangs()	Returns the list of available languages.	-	String []

Description:

The default language is “en-US”.

This function returns the list of languages supported by INVOX Medical, accessible through ***INVOX.lang***.

Parameters: No parameters required.

Returns:

- Available Language Supported. **String []**. Supported languages:
 - `INVOX.lang.en_US`: ‘en-US’
 - `INVOX.lang.es_ES`: ‘es-ES’
 - `INVOX.lang.pt_PT`: ‘pt-PT’
 - `INVOX.lang.pt_BR`: ‘pt-BR’

Usage:

```
let langs = INVOX.GetLangs();
console.log(langs);

// output: ['es-ES', 'en-US', 'pt-PT', 'pt-BR']
```

ShowHelp

Function	Description	Parameters	Returns
ShowHelp()	Displays the help defined by the INVOX.OnClickHelp() function.	-	void

Description: This function enables the INVOX.OnClickHelp function, which can be customized to display a help to the user.

Parameters: No parameters required.

Returns: No value returned.

Usage:

```
INVOX.ShowHelp();
```

SetDictationRunning

Function	Description	Parameters	Returns
SetDictationRunning()	Allows you to turn on speech recognition.	-	void

Description: This function allows you to turn on speech recognition.

Parameters: No parameters required.

Returns: No value returned.

Usage:

```
INVOX.SetDictationRunning();
```

SetDictationPaused

Function	Description	Parameters	Returns
SetDictationPaused()	Allows you to turn off speech recognition.	-	void

Description: This function allows you to turn off speech recognition when dictating.

Parameters: No parameters required.

Returns: No value returned.

Usage:

```
INVOX.SetDictationPaused();
```

UpdateStatusBarToCurrentScope

Function	Description	Parameters	Returns
UpdateStatusBarToCurrentScope()	Updates the status message to show speech recognition state.	-	void

Description: This function allows you to update the Status Bar showing the current status of the recognizer. The message indicates if the speech recognition is turned on or off.

Parameters: No parameters required.

Returns: No value returned.

Usage:

```
INVOX.UpdateStatusBarToCurrentScope();
```

GetWriterTarget

Function	Description	Parameters	Returns (sync)	Returns (async)
GetWriterTarget()	Returns the instance of the current editor.	-	Object	Promise<Object>

Description: The function allows you to get the current editor instance (Writer Target).

Parameters: No parameters required.

Returns:

Note: When an asynchronous Writer is used, the function returns a Promise instead of a synchronous value.

- Writer Target: **Object**. Refers to the editor instance established.

Usage: Using an synchronous Writer:

```
const editorInstance = INVOX.GetWriterTarget();
```

Using an asynchronous Writer:

```
// Option 1: Async/Await
try {
  const editorInstance = await INVOX.GetWriterTarget();
} catch(error) {
  console.error(error);
}

// Option 2: Promises
INVOX.GetWriterTarget()
  .then((editorInstance) => console.log(editorInstance))
  .catch(error => console.error(error));
```

Write

Function	Description	Parameters	Returns (sync)	Returns (async)
Write(text)	Writes the text in the current editor.	String	void	Promise<void>

Function	Description	Parameters	Returns (sync)	Returns (async)
Write(text, editorInstance?)	Writes the text in the specified editor.	String, Object	void	Promise<void>

Description: This function allows you to write text in the caret position directly in the current editor (Writer Target) or in the editor specified by parameter.

Parameters:

- *text*: String. Text to write.
- *editorInstance*: Object (Optional). Editor instance.

Returns:

Note: When an asynchronous Writer is used, the function returns a Promise instead of a synchronous value.

No value returned.

Usage: Using an synchronous Writer:

```
const editorInstance = document.querySelector("#inbox-textarea-2");
```

```
INVOX.Write("Your text...");
// or
INVOX.Write("Your text...", editorInstance);
```

Using an asynchronous Writer:

```
// Option 1: Async/Await
try {
  await INVOX.Write("Your text...");
} catch(error) {
  console.error(error);
}

// Option 2: Promises
INVOX.Write("Your text...")
  .then(() => console.log("Write operation finished"))
  .catch(error => console.error(error));
```

PrependText

Function	Description	Parameters	Returns (sync)	Returns (async)
PrependText(text)	Prepends text in the current editor.	String	void	Promise<void>
PrependText(text, editorInstance?)	Prepends text in the specified editor.	String, Object	void	Promise<void>

Description: The function writes the text at the beginning of the content of the current editor (Writer Target) or to the editor specified by parameter.

Parameters:

- *text*: String. Text to prepend.
- *editorInstance*: Object (Optional). Editor instance.

Returns:

Note: When an asynchronous Writer is used, the function returns a Promise instead of a synchronous value.

No value returned.

Usage: Using an synchronous Writer:

```
const editorInstance = document.querySelector("#inbox-textarea-2");
```

```
INVOX.PrependText("Your text...");
```

```
// or
```

```
INVOX.PrependText("Your text...", editorInstance);
```

Using an asynchronous Writer:

```
// Option 1: Async/Await
```

```
try {  
  await INVOX.PrependText("Your text...");  
} catch(error) {  
  console.error(error);  
}
```

```
// Option 2: Promises
```

```
INVOX.PrependText("Your text...")  
  .then(() => console.log("Write operation finished"))  
  .catch(error => console.error(error));
```

AppendText

Function	Description	Parameters	Returns (sync)	Returns (async)
AppendText(text)	Appends text in the current editor.	String	void	Promise<void>
AppendText(text, editorInstance?)	Appends text in the specified editor.	String, Object	void	Promise<void>

Description: The function writes the text to the end of the content of the current editor (Writer Target) or to the editor specified by parameter.

Parameters:

- *text*: String. Text to append.
- *editorInstance*: Object (Optional). Editor instance.

Returns:

Note: When an asynchronous Writer is used, the function returns a Promise instead of a synchronous value.

No value returned.

Usage: Using an synchronous Writer:

```
const editorInstance = document.querySelector("#inbox-textarea-2");
```

```
INVOX.AppendText("Your text...");
```

```
// or
```

```
INVOX.AppendText("Your text...", editorInstance);
```

Using an asynchronous Writer:

```
// Option 1: Async/Await
```

```
try {  
  await INVOX.AppendText("Your text...");  
} catch(error) {  
  console.error(error);  
}
```

```
// Option 2: Promises
```

```
INVOX.AppendText("Your text...")  
  .then(() => console.log("Write operation finished"))  
  .catch(error => console.error(error));
```

ClearText

Function	Description	Parameters	Returns (sync)	Returns (async)
ClearText()	Clear the text in the current editor.	-	void	Promise<void>
ClearText(editorInstance)	Clear the text in the specified editor.	Object	void	Promise<void>

Description: The function clears the content of the current editor (Writer Target) or to the editor specified by parameter.

Parameters:

- **editorInstance:** Object (Optional). Editor instance.

Returns:

Note: When an asynchronous Writer is used, the function returns a Promise instead of a synchronous value.

No value returned.

Usage: Using an synchronous Writer:

```
const editorInstance = document.querySelector("#inbox-textarea-2");
```

```
INVOX.ClearText();
```

```
// or
```

```
INVOX.ClearText(editorInstance);
```

Using an asynchronous Writer:

```
// Option 1: Async/Await
```

```
try {  
  await INVOX.ClearText();  
} catch(error) {  
  console.error(error);  
}
```

```
// Option 2: Promises
```

```
INVOX.ClearText()  
  .then(() => console.log("Write operation finished"))  
  .catch(error => console.error(error));
```

GetText

Function	Description	Parameters	Returns (sync)	Returns (async)
GetText()	Get the text from the current editor.	-	String	Promise<String>
GetText(editorInstance?)	Get the text from the specified editor.	Object	String	Promise<String>

Description: This functions allows to get the full content from the current editor (Writer Target) or from the editor specified by parameter.

Parameters:

- **editorInstance:** Object (Optional). Editor instance.

Returns:

Note: When an asynchronous Writer is used, the function returns a Promise instead of a synchronous value.

- Full Text: **String**. Full text from the editor content.

Usage: Using an synchronous Writer:

```
const editorInstance = document.querySelector("#inbox-textarea-2");
```

```
INVOX.GetText();  
// or  
INVOX.GetText(editorInstance);
```

Using an asynchronous Writer:

```
// Option 1: Async/Await  
try {  
  const text = await INVOX.GetText();  
  console.log(text);  
} catch(error) {  
  console.error(error);  
}  
  
// Option 2: Promises  
INVOX.GetText()  
  .then((text) => console.log(text))  
  .catch(error => console.error(error));
```

SetSelection

Function	Description	Parameters	Returns (sync)	Returns (async)
SetSelection(range)	Allows yo to select text by a range in the current editor.	Object	void	Promise<void>
SetSelection(range, editorInstance?)	Allows yo to select text by a range in the specified editor.	Object, Object	void	Promise<void>

Description: This function allows you to select a text in the content of the current editor (Writer Target) or in the the editor specified by parameter.

Parameters:

Note: Use the `INVOX.Range(cursorStartIndex, cursorEndIndex)` constructor to create the selection range.

- **range:** Object. Object of type `INVOX.Range` that indicates the location of the cursor.
- **editorInstance:** Object (Optional). Editor instance.

Returns:

Note: When an asynchronous Writer is used, the function returns a Promise instead of a synchronous value.

No value returned.

Usage: Using an synchronous Writer:

```
const selectionRange = new INVOX.Range(1,5);
const editorInstance = document.querySelector("#inbox-textarea-2");
```

```
INVOX.SetSelection(selectionRange);
// or
INVOX.SetSelection(selectionRange, editorInstance);
```

Using an asynchronous Writer:

```
const selectionRange = new INVOX.Range(1,5);

// Option 1: Async/Await
try {
  await INVOX.SetSelection(selectionRange);
} catch(error) {
  console.error(error);
}
```

```

}

// Option 2: Promises
INVOK.SetSelection(selectionRange)
  .then(() => console.log("Write operation finished"))
  .catch(error => console.error(error));

```

GetMicrophoneName

Function	Description	Parameters	Returns
GetMicrophoneName()	Returns the current microphone name.	-	String

Description: This function returns the name of the active microphone.

Parameters: No parameters required.

Returns:

- Microphone Name. **String**.

Usage:

```
INVOK.GetMicrophoneName();
```

GetUserInfo

Function	Description	Parameters	Returns
GetUserInfo()	Returns the information of the active user.	-	Object

Description: The function allows you to get information about the current user.

Parameters: No parameters required.

Returns:

- User information. `Object`. Information about the user.
 - Login: `String`.
 - Name: `String`.
 - Email: `String`.
 - Specialty: `String`.
 - Organization: `String`.
 - IsFirstLogin: `Boolean`.
 - LastLogin: `String`.

Usage:

```
INVOX.GetUserInfo();
```

GetCurrentSession

Function	Description	Parameters	Returns
GetCurrentSession()	Returns all the information related to the current session.	-	<code>Object</code>

Description: This function allows you to get the data related to the currently active session.

Parameters: No parameters required.

Returns:

- Session: `Object`. Current session instance.

Usage:

```
INVOX.GetCurrentSession();
```

In case you are not logged in, *INVOX Medical SDK* will display an informational message and indicate that you must first log in to use it.

GetCurrentLang

Function	Description	Parameters	Returns
GetCurrentLang()	Returns a string with the current language of the session.	-	String

Description:

The default language is “en-US”.

This function returns the current language of the messages displayed by the *Status Bar*.

Parameters: No parameters required.

Returns:

- Current Language: **String**. Supported languages:
 - **INVOX.lang.en_US**: ‘en-US’
 - **INVOX.lang.es_ES**: ‘es-ES’
 - **INVOX.lang.pt_PT**: ‘pt-PT’
 - **INVOX.lang.pt_BR**: ‘pt-BR’

Usage:

```
const lang = INVOX.GetCurrentLang();
console.log(lang);

// output: 'en-US'
```

CustomizeComponents

Function	Description	Parameters	Returns
CustomizeComponents(callback)	Allows you to handle INVOX Medical feedback before session starts.	Function	void

Description: The function allows you to handle INVOX Medical feedback before logging in. For example, you can handle the status of the session to display messages to the user.

You can overwrite this handlers again after the session starts without using this function.

Parameters:

- **callback**: Function. Function to set feedback handlers before session starts.

Returns: No value returned.

Usage:

```
// 1. Prepare your handlers
function customizeFeedback () {
    INVOX.OnChangeProgressBar((msg, type) => updateProgressbar(msg.Percentage));
    INVOX.OnChangeStatusBar((msg) => showProgressStatus(msg));
    /* All your required customizations */
}

// 2. Apply your customization (only works before session starts)
INVOX.CustomizeComponents(customizeFeedback);

// 3. Login starts...

// 4. (Optional) Overwrite feedback components if you requires
INVOX.OnChangeStatusBar((msg) => showSessionStatus(msg));
```

Status Bar

The status bar shows the status changes that occurs during the authentication and resource download process, as well as informative messages that may arise during the session.

OnChangeStatusBar

Function	Description	Parameters	Returns
OnChangeStatusBar(callback)	Allows you to handle session status events.	Function	void

Description: This function allows you to handle status events to show them to the user if requires.

Parameters:

- **callback**. Function. Function that will execute the new behaviour each time a new message arrives. It receives a string with the status message and its type, which can be:

- *msg*: String. Status message.
- *type*: Number. Type of status message:
 - * INVOX.MessageType.ERROR
 - * INVOX.MessageType.INFO
 - * INVOX.MessageType.WARNING
 - * INVOX.MessageType.SUCCESS

Returns: No value returned.

Usage:

```
INVOX.OnChangeStatusBar(function (msg, type) {
  let HTMLStatusBar = document.getElementById("invox-statusbar");

  HTMLStatusBar.value = `[${type}] ${msg}`;
  HTMLStatusBar.style.backgroundColor = type == INVOX.MessageType.ERROR ? "red" : "blue";
});
```

Progress Bar

The loading bar shows the progress of the resource download every time you log in to INVOX Medical.

OnChangeProgressBar

Function	Description	Parameters	Returns
OnChangeProgressBar(callback)	Allows you to handle progress activity.	Function	void

Description: This function allows you to define the behaviour during the resource download.

Parameters:

- *callback*. Function. Function that receives the progress activity.
 - *msg.Percent* : Number. Current load percentage.
 - *msg.PercentStep* : Number. Percentage of the current step.
 - *msg.Description* : String. Description of the current step. **Can be undefined.**

Returns: No value returned.

Usage:

```
INVOX.OnChangeProgressBar(function (msg) {  
    let description = null;  
    if (msg.Description !== undefined) {  
        description = ` ${msg.Description}`;  
    }  
    let HTMLProgressBar = document.getElementById("inbox-progressbar");  
    HTMLProgressBar.innerText = `${+msg.Percent}% ${description}`;  
    HTMLProgressBar.style.width = `${+msg.Percent}%`;  
});
```

OnFinishProgressBar

Function	Description	Parameters	Returns
OnFinishProgressBar(callback)	Allows you to handle progress ends event.	Function	void

Description: This function allows you to define the behaviour when the process ends.

Parameters:

- **callback:** Function. Function triggered when progress reach the end.

Returns: No value returned.

Usage:

```
INVOX.OnFinishProgressBar(function () {  
    HTMLProgressBar.innerText = "100% Successful login!"  
});
```

Audio Level Indicator

Function	Description	Parameters	Returns
OnUpdateVumeterMicActivity(callback)	Allows you to handle the audio activity.	Function	void

Description: The function allows you to handle the audio level indicator using a callback function.

Parameters:

- **callback** : Function. Method that receives a percentage of the audio input activity.
 - **activity** : Number. Percentage of the audio level.

Returns: No value returned.

Usage:

```
INVOX.OnUpdateVumeterMicActivity(function (activity) {  
    let HTMLVUMeter = document.getElementById("invox-progress-vumeter");  
    let HTMLVUMeterValue = document.getElementById("invox-vumeter-value");  
  
    HTMLVUMeter.style.width = `${parseInt(activity)}%`;  
    HTMLVUMeterValue.innerText = `${parseInt(activity)}%`;  
});
```

Recognizer

Customize the behaviour when the recognizer is started

Function	Description	Parameters	Returns
OnStartedRecognizer(callback)	Allows you to handle the recognizer init event.	Function	void

Description: The function allows you to customize the behaviour when the recognizer is successfully started.

Parameters:

- **callback** : Function. Function called when the recognizer is successfully started.

Usage:

```
INVOX.OnStartedRecognizer(function () {  
    let HTMLButtonDictationPaused = document.getElementById('invox-btn-dictation-off');  
    let HTMLButtonDictationRunning = document.getElementById('invox-btn-dictation-on');  
    let HTMLButtonDictationSwitch = document.getElementById('invox-btn-switch-dictation');  
  
    HTMLButtonDictationPaused.onclick = INVOX.SetDictationPaused;  
    HTMLButtonDictationRunning.onclick = INVOX.SetDictationRunning;
```

```

HTMLButtonDictationSwitch.onclick = INVOX.SwitchDictation;

INVOX.SetDictationPaused();
});

```

Customize the behaviour when the recognizer is paused

Function	Description	Parameters	Returns
OnPausedRecognizer(callback)	Allows you to handle the recognizer paused events.	Function	void

Description: The function allows you to customize the behaviour when the recognizer is turned off.

Parameters:

- **callback:** Function. Function to be executed when the recognizer is turned off.

Usage:

```

INVOX.OnPausedRecognizer(function () {
    let HTMLButtonDictationPaused = document.getElementById('invox-btn-dictation-off');
    let HTMLButtonDictationRunning = document.getElementById('invox-btn-dictation-on');
    let HTMLButtonDictationSwitch = document.getElementById('invox-btn-switch-dictation');

    HTMLButtonDictationSwitch.innerHTML = "Switch to On";
    HTMLButtonDictationSwitch.classList.remove('invox-off-btn');
    HTMLButtonDictationSwitch.classList.add('invox-on-btn');

    HTMLButtonDictationPaused.classList.add('invox-off-focus-btn');
    HTMLButtonDictationRunning.classList.remove('invox-on-focus-btn');

    HTMLRunningIndicator.classList.remove('invox-running-indicator');
    HTMLRunningIndicator.classList.add('invox-paused-indicator');
});

```

Customize the behaviour when the recognizer is active

Function	Description	Parameters	Returns
OnRunningRecognizer(callback)	Allows you to handle the recognizer starts events.	Function	void

Description: The function allows you to customize the behaviour when the recognizer is turned on.

Parameters:

- **callback:** Function. Function to be executed when the recognizer is turned on.

Usage:

```
INVOX.OnRunningRecognizer(function () {
  let HTMLButtonDictationPaused = document.getElementById('invox-btn-dictation-off');
  let HTMLButtonDictationRunning = document.getElementById('invox-btn-dictation-on');
  let HTMLButtonDictationSwitch = document.getElementById('invox-btn-switch-dictation');

  HTMLButtonDictationSwitch.innerHTML = "Switch to Off";
  HTMLButtonDictationSwitch.classList.remove('invox-on-btn');
  HTMLButtonDictationSwitch.classList.add('invox-off-btn');

  HTMLButtonDictationPaused.classList.remove('invox-off-focus-btn');
  HTMLButtonDictationRunning.classList.add('invox-on-focus-btn');

  HTMLRunningIndicator.classList.remove('invox-paused-indicator');
  HTMLRunningIndicator.classList.add('invox-running-indicator');
});
```

Customize the behaviour when the recognizer throws an error

Function	Description	Parameters	Returns
OnErrorRecognizer(callback)	Allows you to handle the recognizer error events.	Function	void

Description: The function allows you to customize the behaviour when the recognizer trigger an error.

Parameters:

- **callback:** Function. Function to be executed when the recognizer has raised an error.

Returns: No value returned.

Usage:

```
INVOX.OnErrorRecognizer(function () {  
    alert("An error occurred in the Recognizer");  
});
```

Hypothesis Viewer

Function	Description	Parameters	Returns
OnChangeVisorHypothesis(callback)	Shows you to handle all kind of hypothesis.	Function, Number	void
OnChangeVisorHypothesisPartialHypothesis(callback)	Shows you to handle only partial hypothesis.	Function	void
OnChangeVisorHypothesisRejectedHypothesis(callback)	Shows you to handle rejected hypothesis.	Function	void
OnChangeVisorHypothesisCommandHypothesis(callback)	Shows you to handle command hypothesis.	Function	void
OnChangeVisorHypothesisMacroHypothesis(callback)	Shows you to handle only macros hypothesis.	Function	void
OnChangeVisorHypothesisAcceptedHypothesis(callback)	Shows you to handle accepted hypothesis.	Function	void

Description:

The hypothesis viewer displays the audio-to-text translation process performed by the recognition engine.

This set of functions allows you to handle each kind of hypothesis to show to user if requires.

Type of the hypothesis	Function assigned
Partial hypothesis	<i>OnChangeVisorHypothesisPartialHypothesis</i>
Rejected hypothesis	<i>OnChangeVisorHypothesisRejectedHypothesis</i>
Command hypothesis	<i>OnChangeVisorHypothesisCommandHypothesis</i>
Macro hypothesis	<i>OnChangeVisorHypothesisMacroHypothesis</i>
Accepted hypothesis	<i>OnChangeVisorHypothesisAcceptedHypothesis</i>

Parameters:

- **callback**. Function. Function to be executed when any hypothesis is received.
 - **msg** : String. Message with the hypothesis.
 - **dictationEventType** : Number. Type of the hypothesis in the *OnChangeVisorHypothesis* function:
 - * INVOX.dictationEventType.ACCEPTED
 - * INVOX.dictationEventType.REJECTED
 - * INVOX.dictationEventType.PARTIAL

```
* INVOX.dictationEventType.COMMAND
* INVOX.dictationEventType.MACRO
```

Returns: No value returned.

Usage:

```
INVOX.OnChangeVisorHypothesis(function (msg, dictationEventType) {
    let HTMLVisor = document.getElementById("invox-hyphotesis-viewer");
    let dictationType = {};

    dictationType[INVOX.dictationEventType.ACCEPTED] = 'ACCEPTED';
    dictationType[INVOX.dictationEventType.PARTIAL] = 'PARTIAL';
    dictationType[INVOX.dictationEventType.COMMAND] = 'COMMAND';
    dictationType[INVOX.dictationEventType.MACRO] = 'MACRO';

    HTMLVisor.innerHTML = `[${dictationType[dictationEventType]}] ${msg}`;
});

// Specific behaviour when receives a rejected hypothesis
INVOX.OnChangeVisorHypothesisRejectedHypothesis(function (msg) {
    logger.ERROR(`[REJECTED HYPHOTESIS] ${msg}`);
});
```

Microphone Permissions Management

These features are only available in the Remote Dictation Service scenario.

Function	Description	Parameters	Returns
OnGrantedAudioSource(callback)	Allows you to handle audio permissions grant.	Function	void
OnDeniedAudioSource(callback)	Allows you to handle audio permissions denied.	Function	void

Description: This function allows you to customize the behaviour of the actions to be taken when the browser requests permission to use the microphone.

Parameters:

- **callback:** Function. Function to be executed in each of the cases.

Returns: No value returned.

Usage:

```
INVOX.OnGrantedAudioSource(function(){
    let micName = INVOX.GetMicrophoneName();
    alert(`Using microphone: ${micName}`);
});
INVOX.OnDeniedAudioSource(function(){
    INVOX.SetDictationPaused();
});
```

Help Display

Function	Description	Parameters	Returns
OnClickHelp(callback)	Allows you to overwrite the show help action.	Function	void

Description: This function allows you to customize your show help action.

Parameters:

- **callback:** Function. Function to be executed when displaying help.

Returns: No value returned.

Usage:

```
INVOX.OnClickHelp(function(){
    let helpURL = YOUR_HELP_URL;
    window.open(helpURL);
});
```

Local microphone driver

Microphone controls customization is only available for the Local Dictation Service and for a set of handheld microphones.

This functionality is supported by this handheld microphones:

- SpeechMike II
- SpeechMike III
- SpeechMike III Air
- Olympus DR-1200

- Olympus RM-4010P
- Nuance PowerMic II

INVOX Medical integrates with these handheld microphones allowing some functions:

- Control the activation and deactivation of voice recognition.
- Lock or unlock the write destination.
- Enable or disable capitalization.
- Etc.

In addition, **a mechanism is provided to customize both a specific button and the actions it performs.** Therefore, there are two possibilities:

- **Modify the default buttons action.** If this default action is changed, it will affect all the microphones that share it.
- **Modify the function that executes each button in particular, for each microphone.**

For example, an Olympus RM-4010P microphone has an F1 button that performs a certain action, but a SpeechMike III Pro microphone also has an F1 button that performs the same action. This action is responsible for enabling or disabling capitalization in the text box where it is dictated. With microphone control customization, the Olympus F1 button could execute a new function and the SpeechMike F1 button could still execute the default action or a new function as well.

Actions available

Action Identifier	Description	Implemented
INVOX.deviceActionId.DEVICE_ADDED	Microphone connected	
INVOX.deviceActionId.DEVICE_REMOVED	Microphone disconnected	
INVOX.deviceActionId.GO_TO_BEGINNING_DOC	Place the caret at the beginning of the current document	Yes
INVOX.deviceActionId.GO_TO_END_DOC	Place the caret at the end of the current document	Yes
INVOX.deviceActionId.GO_TO_BEGINNING_PARAGRAPH	Place the caret at the beginning of the current paragraph	
INVOX.deviceActionId.GO_TO_END_PARAGRAPH	Place the caret at the end of the current paragraph	
INVOX.deviceActionId.GO_TO_NEXT_PARAGRAPH	Place the caret at the beginning of the next paragraph	
INVOX.deviceActionId.GO_TO_PREV_PARAGRAPH	Place the caret at the beginning of the previous paragraph	
INVOX.deviceActionId.MOVE_BACKWARD	Place the caret at the beginning of the previous word	
INVOX.deviceActionId.MOVE_FORWARD	Place the caret at the beginning of the next word	
INVOX.deviceActionId.NEXT_FIELD	Select the next field	Yes
INVOX.deviceActionId.PREVIOUS_FIELD	Select the previous field	Yes
INVOX.deviceActionId.REDO	Redo the last change on the current document	
INVOX.deviceActionId.UNDO	Undo the last change on the current document	
INVOX.deviceActionId.START_DICTATION	Start the speech recognizer	Yes
INVOX.deviceActionId.STOP_DICTATION	Stop the speech recognizer	Yes
INVOX.deviceActionId.SWITCH_DICTATION	Start or stop the speech recognizer	Yes
INVOX.deviceActionId.SWITCH_CAPS_LOCK	Enable or disable caps lock	Yes

Action Identifier	Description	Implemented
INVOX.deviceActionId.SWITCH_DOC_LOCK	Lock or unlock writing to the current document	
INVOX.deviceActionId.SWITCH_WRITER	Switch between documents (writer targets)	

CustomizeDeviceAction

Function	Description	Parameters	Returns
CustomizeDeviceAction(deviceActionId, callback)	Allows you to customize the actions of the device buttons.	String, Function	void

Description:

The new function overwrites the default action.

The **INVOX.CustomizeDeviceAction** function allows you to customize the default action defined for handheld microphones.

To check the available device actions, use the INVOX.GetDeviceActions function.

Parameters:

- **deviceActionId**: Number. Identifier of the default action.
- **callback**: Function. Method defining the action to perform.
 - **session**: Object. Current user session.

Returns: No value returned.

Usage:

```
// Example to overwrites focus switching between two editors (TextArea).

// 1. Defining TextArea identifiers
const idTargetEditor1 = "invox-textarea-1";
const idTargetEditor2 = "invox-textarea-2";

// 2. Establish a writer for TextAreas
INVOX.SetTextWriter(INVOX.TextAreaTextWriter);

// 3. Implement the function to switch between editors (Writer Targets)
function switchFocus() {
```

```

const editor1 = document.getElementById(idTargetEditor1);
const editor2 = document.getElementById(idTargetEditor2);
if (editor1 === document.activeElement) {
    INVOX.SetWriterTarget(idTargetEditor2);
    editor2.focus();
    editor1.blur();
    return;
}
INVOX.SetWriterTarget(idTargetEditor1);
editor1.focus();
editor2.blur();
}

// 4. Set the new action
INVOX.CustomizeDeviceAction(INVOX.deviceActionId.SWITCH_WRITER, switchFocus);

```

This change affects all microphones that have this action assigned by default in any button.

GetDeviceActions

Function	Description	Parameters	Returns
GetDeviceActions()	Returns the list of action id and description in the current language.	-	Object []
GetDeviceActions(lang)	Returns the list of action id and description for a specific language.	String	Object []

Description: This function returns information about the device actions available in *INVOX.deviceActionId*.

Parameters:

- *lang*: String (Optional). Supported languages:
 - *INVOX.lang.en_US*: 'en-US'
 - *INVOX.lang.es_ES*: 'es-ES'
 - *INVOX.lang.pt_PT*: 'pt-PT'
 - *INVOX.lang.pt_BR*: 'pt-BR'

Returns:

- Available Action List. Object []. The format of the information provided for each action is as follows:
 - *deviceActionId*: *INVOX.deviceActionId*. Device action identifier.

- **description:** String. Action description.

Usage:

```
INVOX.GetDeviceActions(INVOX.lang.en_US);
```

Controls available

SpeechMike controls

- SpeechMike II
- SpeechMike III
- SpeechMike III Air

The INVOX.CustomizeSpeechMikeControl function is used as a mechanism to customize SpeechMike device buttons.

Button name	Button identifier	Identifier of the default action
<i>Record</i>	INVOX.SpeechMikeButton.RecPushed	INVOX.deviceActionId.SWITCH_DICTATION
<i>Play</i>	INVOX.SpeechMikeButton.PlayPushed	INVOX.deviceActionId.START_DICTATION
<i>Stop</i>	INVOX.SpeechMikeButton.StopPushed	INVOX.deviceActionId.STOP_DICTATION
<i>Play/Stop</i>	INVOX.SpeechMikeButton.PlayStopPushed	INVOX.deviceActionId.SWITCH_DICTATION
<i>End-of-letter (EOL)</i>	INVOX.SpeechMikeButton.EOLPushed	INVOX.deviceActionId.SWITCH_DOC_LOCK
<i>Insert/Overwrite</i>	INVOX.SpeechMikeButton.InsertPushed	Undefined
<i>F1</i>	INVOX.SpeechMikeButton.F1Pushed	INVOX.deviceActionId.SWITCH_CAPS_LOCK
<i>F2</i>	INVOX.SpeechMikeButton.F2Pushed	INVOX.deviceActionId.SWITCH_WRITER
<i>Fast Forward</i>	INVOX.SpeechMikeButton.FFPushed	INVOX.deviceActionId.NEXT_FIELD
<i>Rewind</i>	INVOX.SpeechMikeButton.RewPushed	INVOX.deviceActionId.PREVIOUS_FIELD

Olympus controls

Olympus RM-4010P

The INVOX.CustomizeOlympusRMControl function is used as a mechanism to customize the buttons of Olympus RM-4010P devices (RecMic II series).

Button name	Button identifier	Identifier of the default action
<i>Record</i>	INVOX.OlympusRMButton.RecPushed	INVOX.deviceActionId.SWITCH_DICTATION
<i>Play</i>	INVOX.OlympusRMButton.PlayPushed	INVOX.deviceActionId.SWITCH_DICTATION
<i>Stop</i>	INVOX.OlympusRMButton.StopPushed	INVOX.deviceActionId.STOP_DICTATION
<i>NEW</i>	INVOX.OlympusRMButton.NewPushed	INVOX.deviceActionId.SWITCH_DOC_LOCK

Button name	Button identifier	Identifier of the default action
<i>Insert/Overwrite</i>	INVOX.OlympusRMButton.InsertPushed	Undefined
<i>F1</i>	INVOX.OlympusRMButton.F1Pushed	INVOX.deviceActionId.SWITCH_CAPS_LOCK
<i>F2</i>	INVOX.OlympusRMButton.F2Pushed	INVOX.deviceActionId.SWITCH_WRITER
<i>Fast Forward</i>	INVOX.OlympusRMButton.FFPushed	INVOX.deviceActionId.NEXT_FIELD
<i>Rewind</i>	INVOX.OlympusRMButton.RewPushed	INVOX.deviceActionId.PREVIOUS_FIELD

Olympus DR-1200

The INVOX.CustomizeOlympusDRControl function is used as a mechanism to customize the buttons of Olympus DR-1200 devices (RecMic series).

Button name	Button identifier	Identifier of the default action
<i>Record</i>	INVOX.OlympusDRButton.RecPushed	INVOX.deviceActionId.SWITCH_DICTATION
<i>Play</i>	INVOX.OlympusDRButton.PlayPushed	INVOX.deviceActionId.SWITCH_DICTATION
<i>Stop</i>	INVOX.OlympusDRButton.StopPushed	INVOX.deviceActionId.STOP_DICTATION
<i>NEW</i>	INVOX.OlympusDRButton.NewPushed	INVOX.deviceActionId.SWITCH_DOC_LOCK
<i>Insert/Overwrite</i>	INVOX.OlympusDRButton.InsertPushed	Undefined
<i>F1</i>	INVOX.OlympusDRButton.F1Pushed	INVOX.deviceActionId.SWITCH_CAPS_LOCK
<i>F2</i>	INVOX.OlympusDRButton.F2Pushed	INVOX.deviceActionId.SWITCH_WRITER
<i>Fast Forward</i>	INVOX.OlympusDRButton.FFPushed	INVOX.deviceActionId.NEXT_FIELD
<i>Rewind</i>	INVOX.OlympusDRButton.RewPushed	INVOX.deviceActionId.PREVIOUS_FIELD

Nuance controls

- Nuance PowerMic II

The INVOX.CustomizeNuanceControl function is used as a mechanism to customize the buttons of Nuance devices.

Button name	Button identifier	Identifier of the default action
<i>Record</i>	INVOX.NuanceButton.RecPushed	INVOX.deviceActionId.SWITCH_DICTATION
<i>Play</i>	INVOX.NuanceButton.PlayPushed	INVOX.deviceActionId.SWITCH_DICTATION
<i>Tick</i>	INVOX.NuanceButton.TickPushed	Undefined
<i>Left of Tick</i>	INVOX.NuanceButton.LeftTickPushed	INVOX.deviceActionId.SWITCH_DOC_LOCK
<i>Right of Tick</i>	INVOX.NuanceButton.RightTickPushed	INVOX.deviceActionId.SWITCH_CAPS_LOCK
<i>Previous</i>	INVOX.NuanceButton.PrevPushed	INVOX.deviceActionId.PREVIOUS_FIELD
<i>Next</i>	INVOX.NuanceButton.NextPushed	INVOX.deviceActionId.NEXT_FIELD
<i>Fast Forward</i>	INVOX.NuanceButton.FFPushed	INVOX.deviceActionId.SWITCH_WRITER

Button name	Button identifier	Identifier of the default action
<i>Rewind</i>	INVOX.NuanceButton.RewPushed	INVOX.deviceActionId.SWITCH_WRITER

SpeechMike devices

CustomizeSpeechMikeControl

The new function overwrites the default action associated with a microphone button.

Function	Description	Parameters	Returns
CustomizeSpeechMikeControl (<i>buttonId</i> , <i>callback</i>)	Allows you to set a new control action.	Number, Function	void

Description: This function allows you to customize the action performed by a button on SpeechMike microphones.

Parameters:

- *buttonId*: Number. Code of the event produced by the microphone button.
- *callback*: Function. Function that overwrites the default action.
 - *session*: Object. Current user session.

Returns: No value returned.

Usage:

```
INVOX.CustomizeSpeechMikeControl(INVOX.SpeechMikeButton.EOLPushed, function (session) {
  console.log("Specific EOL button action changed for SpeechMike");
});
```

To check the list of available buttons, execute the function INVOX.GetSpeechMikeControls.

GetSpeechMikeControls

Function	Description	Parameters	Returns
GetSpeechMikeControls ()	Returns the list of available buttons for SpeechMike devices.	-	Object []

Description: This function returns the list of available buttons for SpeechMike devices. #### Parameters:
No parameters required.

Returns:

- Available Button List. `Object []`. The format of the information provided for each button is as follows:
 - **buttonId**: `String`. Identifier of the button present in *INVOX.SpeechMikeButton*.
 - **name**: `String`. Name of the button.

Usage:

```
INVOX.GetSpeechMikeControls();
```

Olympus devices

CustomizeOlympusDRControl

The new function overwrites the default action associated with a microphone button.

Function	Description	Parameters	Returns
CustomizeOlympusDRControl(buttonId, callback)	Allows you to set a new control action.	Number, Function	void

Description: This function allows you to customize the action performed by a button on Olympus DR-1200 microphones (RecMic series).

Parameters:

- **buttonId**: `Number`. Code of the event produced by the microphone button.
- **callback**: `Function`. Function that overwrites the default action.
 - **session**: `Object`. Current user session.

Returns: No value returned.

Usage:

```
INVOX.CustomizeOlympusDRControl(INVOX.OlympusDRButton.NewPushed, function (session) {  
    console.log("Specific NEW button action changed for Olympus DR-1200");  
});
```

To check the list of available buttons, execute the function `INVOX.GetOlympusDRControls`.

CustomizeOlympusRMControl

The new function overwrites the default action associated with a microphone button.

Function	Description	Parameters	Returns
CustomizeOlympusRMControl(<i>buttonId</i>, <i>callback</i>)	Allows you to set a new control action.	Number, Function	void

Description: This function allows you to customize the action performed by a button on Olympus RM-4010P microphones (RecMic II series).

Parameters:

- ***buttonId***: Number. Code of the event produced by the microphone button.
- ***callback***: Function. Function that overwrites the default action.
 - ***session***: Object. Current user session.

Returns: No value returned.

Usage:

```
INVOX.CustomizeOlympusRMControl(INVOX.OlympusRMButton.NewPushed, function (session) {  
    console.log("Specific NEW button action changed for Olympus RM-4010P");  
});
```

To check the list of available buttons, execute the function INVOX.GetOlympusRMControls.

GetOlympusDRControls

Function	Description	Parameters	Returns
GetOlympusDRControls()	Returns the list of available buttons for Olympus DR-1200 devices.	-	Object []

Description: This function returns the list of available buttons for Olympus DR-1200 devices (RecMic series). #### Parameters:

No parameters required.

Returns:

- Available Button List. `Object []`. The format of the information provided for each button is as follows:
 - **buttonId**: `String`. Identifier of the button present in *INVOX.OlympusDRButton*.
 - **name**: `String`. Name of the button.

Usage:

```
INVOX.GetOlympusDRControls();
```

GetOlympusRMControls

Function	Description	Parameters	Returns
GetOlympusRMControls()	Returns the list of available buttons for Olympus RM-4010P devices.	-	<code>Object []</code>

Description: This function returns the list of available buttons for Olympus RM-4010P devices (RecMic II series).

Parameters: No parameters required.

Returns:

- Available Button List. `Object []`. The format of the information provided for each button is as follows:
 - **buttonId**: `String`. Identifier of the button present in *INVOX.OlympusRMButton*.
 - **name**: `String`. Name of the button.

Usage:

```
INVOX.GetOlympusRMControls();
```

Nuance devices

CustomizeNuanceControl

The new function overwrites the default action associated with a microphone button.

Function	Description	Parameters	Returns
CustomizeNuanceControl(buttonId, callback)	Allows you to set a new control action.	Number, Function	void

Description: This function allows you to customize the action performed by a button on Nuance microphones.

Parameters:

- **buttonId:** Number. Code of the event produced by the microphone button.
- **callback.** Function. Function that overwrites the default action.
 - **session:** Object. Current user session.

Returns: No value returned.

Usage:

```
INVOX.CustomizeNuanceControl(INVOX.NuanceButton.TickPushed, function (session) {
  console.log("Specific Tick button action changed for Nuance PorwerMic II");
});
```

GetNuanceControls

Function	Description	Parameters	Returns
GetNuanceControls()	Returns the list of available buttons for Nuance devices.	-	Object []

Description: This function returns the list of available buttons for Nuance devices.

Parameters: No parameters required.

Returns:

- Available Button List. Object []. The format of the information provided for each button is as follows:
 - **buttonId:** String. Identifier of the button present in *INVOX.NuanceButton*.
 - **name:** String. Name of the button.

Usage:

```
INVOX.GetNuanceControls();
```

Voice commands

INVOX Medical provides a set of voice commands to work during dictation.

These commands range from starting or stopping speech recognition to working with text to apply formatting, delete or select text, among other functions.

In addition, **INVOX Medical offers the ability to customize the action that a command performs.**

Commands available

Command identifier	Description	Implemented
INVOX.commandId.ALL_CAPS	Capitalize a specific text in the current document	Yes
INVOX.commandId.CAPS	Capitalize the first letter of the next word	Yes
INVOX.commandId.NO_CAPS	Lowercase all letters in a specific text in the current document	Yes
INVOX.commandId.CAPS_ACTIVATE	Enable caps lock	Yes
INVOX.commandId.CAPS_DEACTIVATE	Disable caps lock	Yes
INVOX.commandId.CAPS_INLINE	Capitalize the first letter of the next word to be dictated	Yes
INVOX.commandId.INITIAL_CAPS_ACTIVATE	Enable writing all words with the first letter capitalized	Yes
INVOX.commandId.INITIAL_CAPS_DEACTIVATE	Disable writing all words with the first letter capitalized	Yes
INVOX.commandId.CAPS_SELECTION	Capitalize the selected text	Yes
INVOX.commandId.NORMALIZE_SELECTION	Lowercase the selected text	Yes
INVOX.commandId.DELETE	Delete a specific text from the current document	
INVOX.commandId.DELETE_SELECTION	Delete the current selection	Yes
INVOX.commandId.DELETE_PREV_PARAGRAPH	Delete the previous paragraph	
INVOX.commandId.DELETE_NEXT_PARAGRAPH	Delete the next paragraph	
INVOX.commandId.DELETE_PREV_SENTENCE	Delete the previous sentence	
INVOX.commandId.DELETE_NEXT_SENTENCE	Delete the next sentence	
INVOX.commandId.GO_TO	Place the caret before a specific text in the current document	Yes
INVOX.commandId.GO_TO_AFTER	Place the caret after a specific text of the current document	Yes
INVOX.commandId.GO_TO_BEGINNING_DOC	Place the caret at the beginning of the current document	Yes
INVOX.commandId.GO_TO_END_DOC	Place the caret at the end of the current document	Yes
INVOX.commandId.GO_TO_BEGINNING_PARAGRAPH	Place the caret at the beginning of the current paragraph	
INVOX.commandId.GO_TO_END_PARAGRAPH	Place the caret at the end of the current paragraph	
INVOX.commandId.GO_TO_PREV_PARAGRAPH	Place the caret at the beginning of the previous paragraph	
INVOX.commandId.GO_TO_NEXT_PARAGRAPH	Place the caret at the beginning of the next paragraph	
INVOX.commandId.GO_TO_BEGINNING_SENTENCE	Place the caret at the beginning of the current sentence	

Command identifier	Description	Implemented
INVOX.commandId.GO_TO_END_SENTENCE	Place the caret at the end of the current sentence	
INVOX.commandId.GO_TO_PREV_SENTENCE	Place the caret at the beginning of the previous sentence	
INVOX.commandId.GO_TO_NEXT_SENTENCE	Place the caret at the beginning of the next sentence	
INVOX.commandId.NEXT_FIELD	Select the next field	Yes
INVOX.commandId.PREVIOUS_FIELD	Select the previous field	Yes
INVOX.commandId.SELECT	Select a specific text of the current document	Yes
INVOX.commandId.UNSELECT	Disable the current selection	Yes
INVOX.commandId.SELECT_ALL	Select all text in the current document	Yes
INVOX.commandId.SELECT_NEXT_PARAGRAPH	Select the next paragraph	
INVOX.commandId.SELECT_NEXT_SENTENCE	Select the next sentence	
INVOX.commandId.SELECT_PREV_PARAGRAPH	Select the previous paragraph	
INVOX.commandId.SELECT_PREV_SENTENCE	Select the previous sentence	
INVOX.commandId.START_DICTATION	Start the speech recognizer	Yes
INVOX.commandId.STOP_DICTATION	Stop the speech recognizer	Yes
INVOX.commandId.SAVE_REPORT	Save the audio and text of the report so far and start a new one	
INVOX.commandId.REDO	Redo the last change on the current document	
INVOX.commandId.UNDO	Undo the last change made to the current document	
INVOX.commandId.TABULATION	Insert a tabulation	Yes

The INVOX.GetCommands function returns the list of available voice commands and their description.

GetCommands

Function	Description	Parameters	Returns
GetCommands()	Returns the list of command id and description in the current language.	-	Object []
GetCommands(lang)	Returns the list of command id and description for a specific language.	String	Object []

Description: This function returns information about the voice commands available in *INVOX.commandId*.

Parameters:

- **lang:** String (Optional). Indicates the language:
 - **INVOX.lang.en_US:** 'en-US'
 - **INVOX.lang.es_ES:** 'es-ES'
 - **INVOX.lang.pt_PT:** 'pt-PT'

- `INVOX.lang.pt_BR`: 'pt-BR'

Returns:

- Available Command List. `Object []`. The format of the information provided for each command is as follows:
 - **commandId**: `INVOX.commandId`. Command identifier.
 - **description**: `String`. Command description.

Usage:

```
INVOX.GetCommands(INVOX.lang.en_US);
```

For more information, see the list of commands available.

DisableCommand

Function	Description	Parameters	Returns
DisableCommand(commandId)	Disables the voice command passed as parameter.	Number	void

Description:

By default all voice commands are enabled.

This function allows you to disable a voice command to prevent it from executing its action when recognized.

Parameters:

- **commandId**: `Number`. Voice command identifier.

Returns: No value returned.

Usage:

```
// Disable a voice command
INVOX.DisableCommand(INVOX.commandId.CAPS_ACTIVATE);
```

```
// Disable a list of voice commands
const commandsToDisable = [
  INVOX.commandId.SELECT_ALL,
  INVOX.commandId.UNSELECT
```



```
];
```

```
commandsToDisable.forEach(commandId => INVOX.DisableCommand(commandId));
```

EnableCommand

Function	Description	Parameters	Returns
EnableCommand(commandId)	Enables the voice command passed as parameter.	Number	void

Description:

By default all voice commands are enabled.

This function allows you to enable a voice command previously disabled.

Parameters:

- *commandId*: Number. Voice command identifier.

Returns: No value returned.

Usage:

```
// Enable a voice command
INVOX.EnableCommand(INVOX.commandId.CAPS_ACTIVATE);

// Enable a list of voice commands
const commandsToEnable = [
  INVOX.commandId.SELECT_ALL,
  INVOX.commandId.UNSELECT
];

commandsToEnable.forEach(commandId => INVOX.EnableCommand(commandId));
```

GetDisabledCommands

Function	Description	Parameters	Returns
GetDisabledCommands()	Returns the list of disabled commands.	-	String []

Description:

By default all voice commands are enabled.

This function returns the list of disabled voice commands.

Parameters: No parameters required.

Returns:

- Disabled Command List: **String []**.

Usage:

```
const disabledCommands = INVOX.GetDisabledCommands();
console.log(disabledCommands);
```

```
//output example: ['INVOX.commandId.CAPS_ACTIVATE', 'INVOX.commandId.SELECT_ALL', 'INVOX.commandId.UNSELECT']
```

GetEnabledCommands

Function	Description	Parameters	Returns
GetEnabledCommands()	Returns the list of enabled commands.	-	String []

Description:

By default all voice commands are enabled.

This function returns the list of enabled voice commands.

Parameters: No parameters required.

Usage:

```
let enabledCommands = INVOX.GetEnabledCommands();
console.log(enabledCommands);

//output example: ['INVOX.commandId.ALL_CAPS', 'INVOX.commandId.CAPS', 'INVOX.commandId.CAPS_ACTIVATE', ...]
```

CustomizeCommand

The new function overwrites the default action of the voice command.

Function	Description	Parameters	Returns
CustomizeCommand(commandId, callback)	Allows you to customize an existing voice command.	String, Function	void

Description: The function allows you to customize an existing voice command overwriting its default action.

Parameters:

- **commandId**: Number. Voice command identifier.
- **callback**: Function. Method that receives the command triggered.
 - **command**: Object. Command information.
 - * **commandId**: Number. Voice command identifier.
 - * **hypothesisText**: String. Full text of the dictated command.
 - * **rightHandSide**: String. Right side of the dictation command.
 - **session**: Object. Current user session.

Returns: No value returned.

Usage:

```
INVOX.CustomizeCommand(INVOX.commandId.START_DICTATION, function (command, session) {
    toastr(`Voice command: ${command.hypothesisText}`);
    INVOX.SetDictationRunning();
});
```

Check the full voice commands list available with the INVOX.GetCommands function, or refer to the commands available section.

Capture internal events

INVOX Medical defines a set of events that are triggered at certain times, either to warn of an error during the session, or even a process that has been executed correctly. These events can be captured in order to define a behaviour.

Event	Description
INVOX.eventTypeReport.LOGIN_ERROR	Event reporting an error during login.
INVOX.eventTypeReport.LOGIN_SUCCESS	Event reporting a successful login.
INVOX.eventTypeReport.NOT_CUSTOMIZED_COMPONENTS	Event occurring during a login where customizable components have not been pre-specified.

The following example describes how to capture these events and how to define a specific action for each event:

```
document.addEventListener(INVOX.eventTypeReport.LOGIN_ERROR, e => {  
    ShowMessage(INVOX.MessageType.ERROR, e.detail);  
});  
document.addEventListener(INVOX.eventTypeReport.LOGIN_SUCCESS, e => {  
    let user = INVOX.GetUserInfo();  
    ShowMessage(INVOX.MessageType.SUCCESS, `Welcome, ${user.Name}!`);  
});  
document.addEventListener(INVOX.eventTypeReport.NOT_CUSTOMIZED_COMPONENTS, e => {  
    ShowMessage(INVOX.MessageType.WARNING, e.detail);  
});
```

The event has a *detail* field with information related to the occurrence of the event.

StartAdaptation

Function	Description	Parameters	Returns
<i>StartAdaptation()</i>	Starts the adaptation of words that are not included in the recognition model.	-	void

Description: The function allows you to start word adaptation in pending state. Once this process is finished, the words will be available for dictation.

If there are no words pending to be added to the recognition model, then no adaptation occurs.

Parameters: No parameters required.

Returns: No value returned.

Usage:

```
INVOX.StartsAdaptation();
```

OnStartAdaptation

Function	Description	Parameters	Returns
OnStartAdaptation(callback)	Allows subscription to adaptation start events.	Function	void

Description: The function allows you to handle adaptation start events. If the adaptation request triggers the addition of new words in the recognition model, this process sends a confirmation event.

Parameters:

- **callback: Function.** Method to handle adaptation start events. This function receives a message as parameter.
 - **message: String.** Message about the adaptation status.

Returns: No value returned.

Usage:

```
INVOX.OnStartAdaptation(function(message) {  
    // handle adaptation start event  
});
```

OnRejectAdaptation

Function	Description	Parameters	Returns
OnRejectAdaptation(callback)	Allows subscription to adaptation reject events.	Function	void

Description: The function allows you to handle adaptation reject events. An adaptation request is rejected when there are no words left to be added to the recognition model.

Parameters:

- **callback: Function.** Method to handle adaptation reject events. This function receives a message as parameter.
 - **message: String.** Message about the adaptation status.

Returns: No value returned.

Usage:

```
INVOX.OnRejectAdaptation(function(message) {  
    // handle adaptation reject event  
});
```

OnFinishAdaptation

Function	Description	Parameters	Returns
OnFinishAdaptation(callback)	Allows subscription to adaptation completion events.	Function	void

Description: The function allows you to handle adaptation completion events. If the adaptation request triggers the addition of new words in the recognition model, this process sends a confirmation event.

Parameters:

- **callback: Function.** Method to handle adaptation completion events. This function receives a message and information about the adaptation process as parameters.
 - **message: String.** Message about the adaptation status.
 - **adaptationInfo: Object.** Adaptation information.
 - * **duration: Number.** Time taken by the adaptation process in seconds.
 - * **success: Boolean.** Adaptation process success.

Returns: No value returned.

Usage:

```
INVOX.OnFinishAdaptation(function(message, adaptationInfo) {  
    // handle adaptation information  
    const { duration, success } = adaptationInfo;  
});
```

OnErrorAdaptation

Function	Description	Parameters	Returns
OnErrorAdaptation(callback)	Allows subscription to adaptation error events.	Function	void

Description: The function allows you to handle adaptation error events.

Parameters:

- **callback: Function.** Method to handle successful adaptation completion events. This function receives a message and the specific error as parameters.
 - **message: String.** Message about the adaptation status.
 - **error: String.** Specific error message.

Returns: No value returned.

Usage:

```
INVOX.OnErrorAdaptation(function(message, error) {
    // handle error
});
```

User Dictionary Repository

Get User Repository Instance:

Function	Description	Parameters	Returns
GetUserDictionary()	Returns the user Dictionary repository.	-	Object

Manage User Repository:

Function	Description	Parameters	Returns
add()	Add word to the user word list.	UserWord	Promise<WordOperationResult>
get()	Get user word list.	-	Promise<UserWord []>
remove()	Remove word from the user word list.	UserWord	Promise<WordOperationResult>
replace()	Replace word in the user word list.	UserWord, UserWord	Promise<WordOperationResult>

Usage summary:

```
// Get Repository
const userDictionary = INVOX.GetUserDictionary();

// Get Word List
const wordList = await userDictionary.get();

// Add Word
const shareWithSpecialtyUsers = true;
const word = INVOX.WordFactory.create("sharedWord", shareWithSpecialtyUsers);
const addResult = await userDictionary.add(word);

// Replace Word
const oldWord = word;
const newWord = INVOX.WordFactory.create("notSharedWord", !shareWithSpecialtyUsers);
const replaceResult = await userDictionary.replace(oldWord, newWord);

// Remove Word
const removeResult = await userDictionary.remove(newWord);
```

Returned values

UserWord

- text: **String**. Word text.
- pronunciations: **String []**. List of pronunciations.
- isShared: **Boolean**. If True the word is shared with other users of the same specialty.
- isAdapted: **Boolean**. If True the word was added to the recognition model, so it is ready to be dictated.

WordOperationResult

- Code: **Number**. Result operation code.
- Message: **String**. Result operation message.

Operation	Code	Message
NULL_OR_UNDEFINED	0	The word cannot be null or undefined
HAS_VALID_FORMAT	1	Valid format
EMPTY_NAME	2	The "Text" field cannot be empty
MISSING_FIELD_NAME	3	Missing "Text" field

Operation	Code	Message
MISSING_FIELD_PRONUNCIATIONS	4	Missing "Pronunciations" field
MISSING_FIELD_SHARED	5	Missing "Shared" field
NOT_ALLOWED_CHARACTERS	6	The "Text" field cannot contain blank spaces
SUCCESSFULLY_ADDED	7	Word added successfully
SUCCESSFULLY_REMOVED	8	Word removed successfully
SUCCESSFULLY_UPDATED	9	Word updated successfully
FAILED_ADD	11	An error occurred while saving the word
ALREADY_EXISTS	12	Word already exists
NOTHING_TO_REMOVE	13	The word you are trying to delete does not exist
NOT_EXIST	14	Word to replace does not exists
FAILED_GET	15	An error occurred while getting the words

Specialty Dictionary Repository

Get Specialty Repository Instance:

Function	Description	Parameters	Returns
GetSpecialtyDictionary()	Returns the specialty Dictionary repository.	-	Object

Manage Specialty Repository:

Function	Description	Parameters	Returns
get()	Get specialty word list.	-	Promise<SpecialtyWord []>

Usage summary:

```
// Get Repository
const specialtyDictionary = INVOX.GetSpecialtyDictionary();

// Get Word List
const wordList = await specialtyDictionary.get();
```

Returned values

SpecialtyWord

- text: **String**. Word text.
- pronunciations: **String []**. List of pronunciations.
- isSpecialtyWord: **Boolean**. If the word is added to the recognition model, then the word is ready to be dictated.
- isShared: **Boolean**. If the word is shared with other users of the same specialty.
- isAdapted: **Boolean**. If the word is added to the recognition model, then the word is ready to be dictated.

WordOperationResult

- Code: **Number**. Result operation code.
- Message: **String**. Result operation message.

Operation	Code	Message
FAILED_GET	15	An error occurred while getting the words

User Templates Repository

Get Repository Instance:

Function	Description	Parameters	Returns
GetUserTemplateRepository()	Returns the user Templates Repository.	-	Object

Manage Repository:

Function	Description	Parameters	Returns
add()	Add template to the user template list.	TextTemplate	Promise<TemplateOperationResult>
get()	Get user template list.	-	Promise<TextTemplate []>
remove()	Remove template from the user template list.	TextTemplate	Promise<TemplateOperationResult>
replace()	Replace template in the user template list.	TextTemplate, TextTemplate	Promise<TemplateOperationResult>

Usage summary:

```
// Get Repository
const templatesRepository = INVOX.GetUserTemplateRepository();
```

```

// Get Template List
const templateList = await templatesRepository.get();

// Add Template
const isInline = true;
const isShared = false;
const template = INVOX.TemplateFactory.create("Adenoma velloso", "Los cortes histológicos...", isShared, isInline);
const addResult = await templatesRepository.add(template);

// Replace Template
const oldTemplate = template;
const newTemplate = INVOX.TemplateFactory.create("ADENOMA VELLOSO", "Los cortes histológicos...", isShared, isInline);
const replaceResult = await templatesRepository.replace(oldTemplate, newTemplate);

// Remove Template
const removeResult = await templatesRepository.remove(newTemplate);

```

Returned values

TextTemplate

- name: **String**. What the user says.
- replacement: **String**. What the user expects instead.
- isShared: **Boolean**. If True the template is shared with other users of the same specialty.
- isInline: **Boolean**. If True the template does not need a pause during dictation to be triggered.
- isAdapted: **Boolean**. If True the template was added to the recognition model, so it is ready to be dictated.

TemplateOperationResult

- Code: **Number**. Result operation code.
- Message: **String**. Result operation message.

Operation	Code	Message
NULL_OR_UNDEFINED	0	The template cannot be null or undefined
HAS_VALID_FORMAT	1	Valid format
MISSING_FIELD_NAME	2	Missing "Name" field
MISSING_FIELD_REPLACEMENT	3	Missing "Replacement" field
MISSING_FIELD_SHARED	4	Missing "Shared" field
INVALID_REPLACEMENT_FORMAT	5	Invalid format in "Replacement" field
INVALID_NAME_FORMAT	6	Invalid format in "Name" field

Operation	Code	Message
EMPTY_NAME	7	The "Name" field cannot be empty
EMPTY_REPLACEMENT	8	The \"Replacement\" field cannot be empty
SUCCESSFULLY_ADDED	10	Template added successfully
SUCCESSFULLY_REMOVED	11	Template removed successfully
SUCCESSFULLY_UPDATED	12	Template updated successfully
FAILED_GET	13	An error occurred while getting the templates
ALREADY_EXISTS	14	Template already exists
FAILED_ADD	15	An error occurred while saving the template
NOTHING_TO_REMOVE	16	The template you are trying to delete does not exist
NOT_EXIST	17	Template to replace does not exists

Specialty Templates Repository

Get Specialty Repository Instance:

Function	Description	Parameters	Returns
GetSpecialtyTemplateRepository()	Returns the specialty Templates Repository.	-	Object

Manage Specialty Repository:

Function	Description	Parameters	Returns
get()	Get specialty templates list.	-	Promise<TextTemplate []>

Usage summary:

```
// Get Repository
const specialtyTemplates = INVOX.GetSpecialtyTemplateRepository();

// Get Templates List
const templatesList = await specialtyTemplates.get();
```

Returned values

TextTemplate

- name: **String**. What the user says.
- replacement: **String**. What the user expects instead.
- isShared: **Boolean**. If True the template is shared with other users of the same specialty.
- isInline: **Boolean**. If True the template does not need a pause during dictation to be triggered.
- isAdapted: **Boolean**. If True the template was added to the recognition model, so it is ready to be dictated.

TemplateOperationResult

- Code: **Number**. Result operation code.
- Message: **String**. Result operation message.

Operation	Code	Message
FAILED_GET	13	An error occurred while getting the templates

Transformations Repository

Transformations cannot be shared between users, so there is only one repository for each user.

Get Repository Instance:

Function	Description	Parameters	Returns
GetTransformationsRepository()	Returns the user Transformations Repository.	-	Object

Manage Repository:

Function	Description	Parameters	Returns
add()	Add transformation to the user transformation list.	Transformation	Promise<TransformationOperationResult>
get()	Get user transformation list.	-	Promise<Transformation []>
remove()	Remove transformation from the user transformation list.	Transformation	Promise<TransformationOperationResult>
replace()	Replace transformation in the user transformation list.	Transformation, Transformation	Promise<TransformationOperationResult>

Usage summary:

```
// Get Repository
const transformationsRepository = INVOX.GetTransformationsRepository();

// Get Transformation List
const transformationList = await transformationsRepository.get();

// Add Transformation
const transformation = INVOX.TransformationFactory.create("covid 19", "SARS-CoV-2");
const addResult = await transformationsRepository.add(transformation);

// Replace Transformation
const oldTransformation = transformation;
const newTransformation = INVOX.TransformationFactory.create("coronavirus", "SARS-CoV-2");
const replaceResult = await transformationsRepository.replace(oldTransformation, newTransformation);

// Remove Transformation
const removeResult = await transformationsRepository.remove(newTransformation);
```

Returned values

Transformation

- pattern: **String**. The recognized text that the user wants to transform.
- replace: **String**. The text that the user expects instead.

TransformationOperationResult

- Code: **Number**. Result operation code.
- Message: **String**. Result operation message.

Operation	Code	Message
NULL_OR_UNDEFINED	0	The transformation cannot be null or undefined
HAS_VALID_FORMAT	1	Valid format
EMPTY_PATTERN	2	The "Pattern" field cannot be empty
NOT_ALLOWED_CHARACTERS	3	The "Pattern" field has an incorrect format
MISSING_FIELD_PATTERN	4	Missing "Pattern" field
MISSING_FIELD_REPLACE	5	Missing "Replace" field

Operation	Code	Message
SUCCESSFULLY_ADDED	6	Transformation added successfully
SUCCESSFULLY_REMOVED	7	Transformation removed successfully
SUCCESSFULLY_UPDATED	8	Transformation updated successfully
FAILED_ADD	10	An error occurred while saving the transformation
ALREADY_EXISTS	11	Transformation already exists
NOTHING_TO_REMOVE	12	The transformation you are trying to delete does not exist
NOT_EXIST	13	Transformation to replace does not exists
FAILED_GET	14	An error occurred while getting the transformations

ChangeLogLevel

Function	Description	Parameters	Returns
ChangeLogLevel(newLevel)	Allows you to change the logging level of the INVOX Logger.	INVOX.LogLevel	void

Description: The function allows you to change the logging level of the INVOX Logger.

Parameters:

- *newLevel*: INVOX.LogLevel. Logging level. **Less value is higher priority.**

Level	Value
Fatal	1
Error	2
Warn	3
Info	4
Debug	5
Trace	6

Returns: No value returned.

Usage:

```
INVOX.ChangeLogLevel(INVOX.LogLevel.ERROR);
```

ChangeLogTarget

Function	Description	Parameters	Returns
ChangeLogTarget(newLogTargets)	Allows you to change the logging target of the INVOX Logger.	INVOX.LogTarget []	void

Description: The function allows you to change the logging messages destination of the INVOX Logger.

Parameters:

- **newLogTargets:** INVOX.LogTarget []. Set logging target list to store or display logging messages.

Logging Target	Value	Description
LOCALSTORAGE	1	Stores logging messages in the Local Storage.
CONSOLE	2	Displays logging messages in the Browser Console.
ONLY_SERVER	3	Stores logging messages only on Server side.

Returns: No value returned.

Usage:

The ONLY_SERVER target prevails over LOCALSTORAGE target.

```
const targets = [INVOX.LogTarget.CONSOLE, INVOX.LogTarget.LOCALSTORAGE];  
INVOX.ChangeLogTarget(targets);
```

ChangeLogKeywordsToIgnore

Function	Description	Parameters	Returns
ChangeLogKeywordsToIgnore(keywords)	Allows you to change the INVOX Logger keyword filter to be ignored.	string []	void

Description: This function allows you to change the keywords to be ignored from the INVOX Logger filter. This filter is used to indicate whether a log should be logged or not.

Parameters:

- **keywords:** string []. Set key words list to ignore the log line.

Returns: No value returned.

Usage:

```
const keywords = ['KeepAlive', 'Event'];  
INVOX.ChangeLogKeywordsToIgnore(keywords);
```

LogDebug

The logs are sent to the INVOX Medical Dictation Service for management.

Function	Description	Parameters	Returns
LogDebug(message)	Outputs a message in the Debug Level.	String	void

Description: The function allows you to write a message in the current Logging Target in the Debug Level.

Parameters

- message: **String**. A JavaScript string containing the message.

Returns: No value returned.

Usage:

```
INVOX.LogDebug("Here goes your message");
```

LogInfo

The logs are sent to the INVOX Medical Dictation Service for management.

Function	Description	Parameters	Returns
LogInfo(message)	Outputs a message in the Information Level.	String	void

Description: The function allows you to write a message in the current Logging Target in the Info Level.

Parameters:

- message: **String**. A JavaScript string containing the message.

Returns: No value returned.

Usage:

```
INVOX.LogInfo("Here goes your message");
```

LogWarn

The logs are sent to the INVOX Medical Dictation Service for management.

Function	Description	Parameters	Returns
LogWarn()	Outputs a message in the Warning Level.	String	void

Description: The function allows you to write a message in the current Logging Target in the Warn Level.

Parameters:

- message: **String**. A JavaScript string containing the message.

Returns: No value returned.

Usage:

```
INVOX.LogWarn("Here goes your message");
```

LogError

The logs are sent to the INVOX Medical Dictation Service for management.

Function	Description	Parameters	Returns
LogError(message)	Outputs a message in the Error Level.	String	void

Description: The function allows you to write a message in the current Logging Target in the Error Level.

Parameters:

- message: **String**. A JavaScript string containing the message.

Returns: No value returned.

Usage:

```
INVOX.LogError("Here goes your message");
```

LogFatal

The logs are sent to the INVOX Medical Dictation Service for management.

Function	Description	Parameters	Returns
LogFatal(message)	Outputs a message in the Fatal Level.	String	void

Description: The function allows you to write a message in the current Logging Target in the Fatal Level.

Parameters:

- message: **String**. A JavaScript string containing the message.

Returns: No value returned.

Usage:

```
INVOX.LogFatal("Here goes your message");
```

LogTrace

The logs are sent to the INVOX Medical Dictation Service for management.

Function	Description	Parameters	Returns
LogTrace(message)	Outputs a message in the Trace Level.	String	void

Description: The function allows you to write a message in the current Logging Target in the Trace Level.

Parameters:

- message: **String**. A JavaScript string containing the message.

Returns: No value returned.

Usage:

```
INVOX.LogTrace("Here goes your message");
```