Documentation Vocalcom

Hermes Universal CRM Connector

API guide

HUCC 6.3.2

Legal disclaimer

Documentation last update: March 10, 2025

Table of contents

Window custom events

VCUCCoreLoadDone Event (JavaScript API reference) for Vocalcom UCCore framework

The VCUCCoreLoadDone event is raised by the Vocalcom UCCore framework when the UCCore framework is loaded.

This event is used to determine whether the Vocalcom UCCore framework APIs are ready to be consumed.

Example

```
window.addEventListener("VCUCCoreLoadDone",() => {
  Vocalcom.UCCore.addHandler("OnCollapse", () => {
          this.isMinimized = true
  })
 })
```

Vocalcom.UCCore methods

Vocalcom UCCore Framework provides methods to use JavaScript API to build front-end CTI Adapter applications.

Vocalcom.UCCore.setHUCCSettings

This method sends the HUCC configuration of the agent to the CTI Adapter.

Syntax

Vocalcom.UCCore.setHUCCSettings(huccConfig);

Example

```
/**
* @param huccConfig The agent CTI Adapter Settings
* @param {string} huccConfig.proxyConfig - ProxyHermesConfig URL
* @param {string} huccConfig.configCode - ProxyHermesConfig Key
* @param {int} huccConfig.customerId - The Customer id (-1 for Hermes V5)
* @param {string} huccConfig.login - The agentId (if not SSO connection)
* @param {string} huccConfig.password - The agent password (if not SSO connection)
* @param {string} huccConfig.agentStation - The station of the agent
```

```
* @param {string} huccConfig.userIdentity - The user identity in the identity provider (for SSO connection)
* @param {string} huccConfig.jsFiles - 'url_file1,url_file2,...' A list of javascript files(for
customization,implementation of additional features...)
*/

var huccConfig = {
proxyConfig: configUrl, // ProxyHermesConfig URL
configCode: configcode, // ProxyHermesConfig Key
customerId: customerId, // The customer id (-1 for Hermes V5)
login: login, // The agent Id (if not SSO connection)
password: password, // The agent password (if not SSO connection)
agentStation: station, // The station of the agent
jsFiles: 'url_file1,url_file2,...' // A list of javascript files (for customization, implementation of
additional features ...)
};

Vocalcom.UCCore.setHUCCSettings(huccConfig);
```

Vocalcom.UCCore.addHandler

Adds the subscriber to the events.

Syntax

Vocalcom.UCCore.addHandler(eventName, handlerFunction);

Parameters

Name	Type	Requ-ired	Description
eventName	String	Yes	Name of the event for which the handler is set. The supported events are as follows: • **OnStartConnectionWithCTIServer:** The event is invoked when the CTI Connector starts the handshake with the CTI Server (use this event to implement a loading screen). • **OnConnectionWithCTIServerLost:** The event is invoked when the CTI Connector loses connection with the CTI Server (use this event to implement a warning screen).

Name	Type	Requ-ired	Description
			• **OnConnectionSlow:** The event is invoked when the network connection is slow (use this event to display a warning message). • **OnReconnecting:** The event is invoked when the network begins reconnecting. • **OnAgentReadyToConnect:** The event is invoked when the agent is ready to login (use this event to implement a login screen). • **OnAgentLogin:** This event is invoked when the agent is logged in (use this event to hide the login screen and display the main screen). • **OnAgentLogout:** This event is invoked when the agent is logged out (use this event to display the login screen, disable the click to call). • **OnAgentReady:** This event is invoked when the agent is ready to handle calls. • **OnAgentPause:** This event is invoked when the agent is on pause and not ready to handle calls.

Name	Type	Requ-ired	Description

```
Vocalcom.UCCore.addHandler("OnAgentPause", function (isPauseCodeSpecified) {
    if(!isPauseCodeSpecified) {
        displaySpecificPauseCodes();
    } else {
        hidePauseCodeList();
    }
});
```

- **OnUpdateAgentState:** This event is invoked when the agent state changes.

```
Vocalcom.UCCore.addHandler("OnUpdateAgentState", function () {
    var context = Vocalcom.UCCore.getGlobalContext();
    /**
    * context.agentStateText {string} the description of the state of agent
    * context.agentState {number} the agent state code
    */
    updateAgentState(context.agentStateText, context.agentState);
});
// context.agentState has one of the following values
// This code is from CtiConstants.js
var AgentStates = {
```

Name	Type	Requ-ired	Description

```
    INACTIVE: 0,   // No agent has logged in.
    WAITING: 1,   // The agent is waiting for a call.
    MANUALCALL: 2,   // The agent is generating a manual call.
    PAUSE: 3,   // The agent is in pause state.
    SUPERVISING: 6,   // The agent is a supervisor.
    WRAPUP: 7,   // The agent is in after call work.
    ALERTING: 8,   // The agent's phone is ringing.
    SEARCHMODE: 9,   // The agent is previewing an outbound automated call.
    CALLBACK: 10,  // A callback is dialed from the agent's phone.
    TEMPNOTREADY: 11,  // The agent is filling a phone number during
    a waiting state.
    PREVIEW: 96,  // The agent is consulting client's information before dialing.
    ONLINE: 100, // The agent is in a conversation. This state can be the result
    of an inbound or outbound call
    ONMAIL: 101, // The agent is handling an email. Deprecated, old Tools mode
    ONCHAT: 102, // The agent is chatting. Deprecated, old Tools mode
    VOICEMAIL: 103, // The agent is handling a voice mail.
    OUTBOUND: 104,  // The agent is handling an outbound call
    CONSULTATION: 1002,
    CONFERENCE: 1004,
    CONSULTATION_CLIENT_ONLINE: 1005

};
```

Name	Type	Requ-ired	Description
			• **OnInboundCallRinging:** This event is invoked on an incoming call (use this event to display incoming call ringing screen). `Vocalcom.UCCore.addHandler("OnInboundCallRinging", function () {` `  var context = Vocalcom.UCCore.getGlobalContext();` `  var callInfo = context.callInfo;` `  /**` `  * callInfo.caller  {string}  client phone number` `  * callInfo.campaignName {string}  inbound campaign name` `  */` `  onInboundCallRinging(callInfo.caller, callInfo.campaignName);` `});` • **OnOutboundCallRinging:** This event is invoked on an outgoing call that is ringing (use this event to display outbound call ringing screen). `Vocalcom.UCCore.addHandler("OnOutboundCallRinging", function () {` `  var context = Vocalcom.UCCore.getGlobalContext();` `  var callInfo = context.callInfo;` `  /**`

Name	Type	Requ-ired	Description

```
     * callInfo.caller       {string}   client phone number
     * callInfo.campaignName {string}   outbound campaign name
     */
    onOutboundCallRinging(callInfo.caller, callInfo.campaignName);
});
```

- **OnInternalCallRinging:** This event is invoked on an incoming internal call (use this event to display internal call ringing screen).

```
Vocalcom.UCCore.addHandler("OnInternalCallRinging", function () {
    var context = Vocalcom.UCCore.getGlobalContext();
    var callInfo = context.callInfo;
    /**
     * callInfo.caller       {string}   agent phone number
     * callInfo.campaignName {string}   campaign name
     */
    onInternalCallRinging(callInfo.caller, callInfo.campaignName);
});
```

- **OnCallOnline:** This event is invoked when the call is established between the agent

Name	Type	Requ-ired	Description
			and the client (use this event to display call information, call timer, etc.). • **OnCallWaiting:** This event is invoked when the client is hold (use this event to pause the call timer, display a hold indicator, etc.). • **OnCallFree:** This event is invoked when the call is finished (use this event to stop call timer, hide qualification panel, etc.). • **OnSecondCallRinging:** This event is invoked when the consult call rings. `Vocalcom.UCCore.addHandler("OnSecondCallRinging", function () {` `  var context = Vocalcom.UCCore.getGlobalContext();` `  var consultNumber = context.consultInfo.number;` `});` • **OnSecondCallOnline:** This event is invoked when the consult call is established. • **OnSecondCallFree:** This event is invoked when the consult call is finished.

Name	Type	Requ-ired	Description
			• **OnRestoreSession:** This event is invoked when the browser is refreshed or the network is reconnected. • **OnUpdateQueuesState:** This event is invoked when telephony queues states are updated. `Vocalcom.UCCore.addHandler("OnUpdateQueuesState", function () {` `  var context = Vocalcom.UCCore.getGlobalContext();` `  var personalQueue = context.queuesState.pers;` `  // The number of calls waiting in the agent's personal queues` `  var primaryQueue = context.queuesState.prim;` `  // The number of calls waiting in the agent's primary queues` `  var secondaryQueue = context.queuesState.sec;` `  // The number of calls waiting in the agent's secondary queues` `  onUpdateQueuesState(personalQueue, primaryQueue, secondaryQueue);` `});` • **OnUpdateAgentRights:** This event is invoked when the agent rights are updated (Use this event to show or to hide action components based on agent rights).

Name	Type	Requ-ired	Description

```
Vocalcom.UCCore.addHandler("OnUpdateAgentRights",() => {
    var context = Vocalcom.UCCore.getGlobalContext();
        UpdateAgentActions(context.agentRights);
});
```

- **OnSetCallDisposition:** This event is invoked when the agent qualifies the call (use this event to store the qualification of the agent on the CRM).

```
/**
* @param callStatusData - call disposition data
* @param {int} callStatusData.callStatusGroup - disposition group
* @param {int} callStatusData.callStatusNum - disposition code
* @param {int} callStatusData.callStatusDetail - disposition detail
* @param {string} callStatusData.callStatusDescription - disposition description
* @param {string} callStatusData.callStatusDetailDescription - disposition detail
description
* @param {string} callStatusData.comment - comment entered by the agent
* @param {string} callStatusData.callbackTime - callback date and time
* @param {string} callStatusData.phoneNumber - callback phone
*/
```

Name	Type	Requ-ired	Description
			`` Vocalcom.UCCore.addHandler("OnSetCallDisposition", function (callStatusData) { }); ``` • **OnConference:** This event is invoked when the conference is established between the client and the agents. • **OnUpdateCampaignName:** This event is invoked when the campaign name is updated (Use this event to display the campaign name of the current call). ``` Vocalcom.UCCore.addHandler("OnUpdateCampaignName",() => { var context = Vocalcom.UCCore.getGlobalContext(); var campaignName = context.callInfo.campaignName; UpdateCampaignName(campaignName); }); ``` • **OnSecondCallWaiting:** This event is invoked when the consult call is held (use this event to pause the consult call timer, display a hold indicator, etc.).

Name	Type	Requ-ired	Description
			• **OnAfterCall:** This event is invoked at the end of the call (Use this event to display call qualification panel, extend wrapup, etc.). • **OnDeleteSession:** This event is invoked at the end of the call session, the agent is ready to handle a new call.

```
Vocalcom.UCCore.addHandler("OnAfterCall", function () {
    var context = Vocalcom.UCCore.getGlobalContext();
    // if the agent has the right to qualify the call
    if(context.agentRights.SetCallStatus ) {
        if(context.qualificationGroup != -1) {
            var callQualifications = context.qualifs[context.qualificationGroup];
                DisplayQualificationPanel(callQualifications);
        }
    }
    // if the agent has the right to extend the wrapup
    if(context.agentRights.ExtendWrapup) {
        DisplayExtendWrapupButton();
    }
});
```

Name	Type	Requ-ired	Description
			• **OnSipHeaders:** This event gives the Sip Headers. `Vocalcom.UCCore.addHandler("OnSipHeaders", function (sipHeaders) {` `  console.log("UCCore sipHeaders ", sipHeaders);` `});` • **OnActivateClickToDial:** This event is invoked to activate the click to call in the CRM. `Vocalcom.UCCore.addHandler("OnActivateClickToDial", function (callback) {` `  // MS Dynamics` `  Microsoft.CIFramework.setClickToAct(true);` `  Microsoft.CIFramework.addHandler("onclicktoact", function(paramStr) {` `  let params = JSON.parse(paramStr);` `  var phoneNUmber = params.value;` `  callback({number: phoneNUmber});` `  // always execute the callback function` `  });` `});` • **OnSearchForCaller:** This event is invoked to search the client data in the CRM.

Name	Type	Requ-ired	Description

```
/**
* phoneNumber {object} {National: String, E164: String}
*/
Vocalcom.UCCore.addHandler("OnSearchForCaller", function (phoneNumber) {
    console.log("UCCore search for caller National Number", phoneNumber.National);
    console.log("UCCore search for caller International Number", phoneNumber.E164);
});
```

- **OnCallerSearchResult**: This event is invoked when the results are ready to be handled by the CTI Adapter.

```
/**
* results {Array} the list of results
*/
Vocalcom.UCCore.addHandler("OnCallerSearchResult", function (results) {
   // the results are also stored in the global context
   var context = Vocalcom.UCCore.getGlobalContext();
   var resultsFromLocalStorage =   context.callInfo.searchCallerResult;
});

The results are an array of objects:
```

Name	Type	Requ-ired	Description

```
[
    {
        objectId: value, // object ID in the CRM
        objectType: value, // object type in the CRM (contact, account, etc.),
        description: value // objet description (contact name and first name for
example)
    },
    ...
]
```

- **OnCallerSearchInputChange**: This event is invoked to search in the CRM the objects corresponding to the value entered by the user in the search feature.

```
Vocalcom.UCCore.addHandler("OnCallerSearchInputChange", function (val) {
   if (val.length > 3) {
      searchAndOpenRecords
(`?$select=fullname,contactid,mobilephone&$filter=contains
      (mobilephone, '${val}') or contains(firstname, '${val}') or contains
(lastname, '${val}')
      or contains(emailaddress1, '${val}')`, '', 'contact', false)
      .then(contacts => {
```

Name	Type	Requ-ired	Description

```
            tempContacts = contacts;
            searchAndOpenRecords(`?$select=name,accountid,telephone1&$filter=contains
(telephone1, '${val}')
            or contains(name, '${val}')`, '', 'account', false).then(accounts => {
               tempAccounts = accounts;
               setCallerResult();
                           });
                        });
                    }
  });
```

- **OnCampaignInfo:** This event gives the information about the current call campaign.

```
Vocalcom.UCCore.addHandler("OnCampaignInfo", function () {
    var context = Vocalcom.UCCore.getGlobalContext();
    var qualifGroup = context.qualificationGroup;
    var campaignType = context.callInfo.compaignType;
    // 0: manual campaign, 1: inbound campaign, 2: outbound campaign
});
```

- **OnManualCampaignChanged**: This event gives the information about the selected

Name	Type	Requ-ired	Description
			manual campaign. `/**` `* campaignID {string} - The selected campaign ID.` `*/` `Vocalcom.UCCore.addHandler("OnManualCampaignChanged", function (campaignID) {` `// The selected campaign ID is also stored in the global context. You can get the value using the following lines.` `var context = Vocalcom.UCCore.getGlobalContext();` `const currentManualCampaignId = context.selectedManualCampaign;` `});` • **OnQueueInfo:** This event indicates the status and information about a queue. `/**` `* queueId {Number} The queue identifier` `* description {String} The description of the queue` `* type {Number} 1 if this queue is of the telephone outgoing type, 0 otherwise`

Name	Type	Requ-ired	Description

```
* actor       {Number} The actor making this queue state change
*                       Stopped by Agent or Default = 0
*                       Stopped by supervisor = 1
*                       Started by Agent or Default = 2
*                       Started by supervisor = 3
*/
Vocalcom.UCCore.addHandler("OnQueueInfo", function (queueId, description, type,
actor) {

});

// you can access to the list of all queues as following
var context = JSON.parse(getFromStorage('CONTEXT'));
var queueList = context.queueList;
```

- **OnMessageInfo:** This event indicates that a message has occurred for the agent.

```
/**
* context {Number} 0: Global, 1: Telephony, 3: Voicemail, 4: Chat, 5: Mail,
6: Fax, 8: Incoming telephony, 9: Outgoing telephony, 10: Social network
* level   {Number} 0: Information, 1: Warning, 2: Error, 3: Fatal
* code    {Number} Possible value between 0 and 999
```

Name	Type	Requ-ired	Description

```
* message {String} The message
*/
Vocalcom.UCCore.addHandler("OnMessageInfo", function (context, level, code,
message) {

});
```

- **OnCallFailed**: This event indicates that a call has failed.

```
/**
* context {Number} 0: Global, 1: Telephony, 3: Voicemail, 4: Chat, 5: Mail, 6: Fax,
8: Incoming telephony, 9: Outgoing telephony, 10: Social network
* message {String} The message
* errorCode is the SIP error code or ISDN cause code
*/
Vocalcom.UCCore.addHandler("OnCallFailed", (context, errorCode, message) => {
})
```

- **OnRecordStarted**: This event indicates that a recording has started.

Name	Type	Requ-ired	Description
			`` /** * filePath {String} The relative path of the record */ Vocalcom.UCCore.addHandler("OnRecordStarted", function (filePath) { }); ``` • **OnRecordStopped:** This event indicates that a recording is stopped. • **OnShowWebRTCCallControlButtons**: This event is invoked to show WebRTC Accept call button and Reject call button. • **OnHideWebRTCCallControlButtons**: This event is invoked to hide WebRTC Accept call button and Reject call button. • **OnPhoneNumberError:** This event is invoked when the phone number to dial is not valid. • **OnWebRTCUpdateUI:** This event indicates that an agent uses a WebRTC station and

Name	Type	Requ-ired	Description
			its state: connected or disconnected. `/**` `* @param {boolean} connectionState - the connection state of the agent WebRTC station (true: station connected, false: station disconnected).` `*/` `Vocalcom.UCCore.addHandler("OnWebRTCUpdateUI", (connectionState: boolean) => {` `if(connectionState) {` `// display an icon to show that the agent is using a WebRTC station and the station is connected` `} else {` `// display an icon to show that the agent is using a WebRTC station and the station is disconnected` `}` `})`

Name	Type	Requ-ired	Description
			• **OnStationError**: This event indicates an error on the station used by the agent (not existing or wrong configuration). • **OnAgentConfigError**: This event indicates that the agent configuration is wrong. • **OnCRMObjectAttachedToCall**: This event indicates the CRM object attached to the ongoing call. `/**` `* @param {string} objectId` `* @param {string} description` `*/` `Vocalcom.UCCore.addHandler("OnCRMObjectAttachedToCall", function (objectId, description) {` `});` • **OnCollapse**: This event is invoked when the CTI Adapter panel is collapsed (minimized).

Name	Type	Requ-ired	Description
			• **OnExpand**: This event is invoked when the CTI Adapter panel is expanded (maximized). • **OnCreateCRMObject**: This event is invoked when the CTI Adapter is asking for the creation of a CRM object. • **OnRefreshSearchForMediaSession**: This event is invoked when the search results for a media session are refreshed. • **OnRestoreMediaSession**: This event is invoked when a specific media session is restored. • **OnSelectCurrentSessionForContext**: This event is invoked to select the current session for the context. • **OnMediaRelatedCRMObjects**: This event is invoked when media related CRM objects are emited. • **OnCRMObjectAttachedToMedia**: This event is invoked when the plugin asks the CTI to attach the CRM object to the media.

Name	Type	Requ-ired	Description
			• **OnAttachCRMObjectToMedia**: This event is invoked when the CTI asks the plugin to attach a CRM object to a media.
handlerFunction	Function	Yes	The handler function is invoked when any of the supported events are triggered.

Example

The sample code demonstrates setting the addHandler method for the **OnAgentReadyToConnect** event.

```
Vocalcom.UCCore.addHandler("OnAgentReadyToConnect", function () {
    $("#console_vc").hide();
    $("#login-panel").show();
});
```

Vocalcom.UCCore.removeHandler

Removes the subscriber from the events.

Syntax

Vocalcom.UCCore.removeHandler(eventName, handlerFunction);

Parameters

Name	Type	Required	Description
eventName	String	Yes	Name of the event for which the handler is set. The supported events are listed in "Vocalcom.UCCore.addHandler" on page 10.
handlerFunction	Function	Yes	The handler function that is to be removed.

Vocalcom.UCCore.removeAllHandlers

Removes the subscribers from the events.

Syntax

Vocalcom.UCCore.removeAllHandlers(eventName);

Parameters

Name	Type	Required	Description
eventName	String	Yes	Name of the event for which the handlers are set. The supported events are listed in "Vocalcom.UCCore.addHandler" on page 10.

Vocalcom.UCCore.addCustomHandler

Adds the subscriber to the events.

Syntax

Vocalcom.UCCore.addCustomHandler(eventName, executeFirst, handlerFunction);

Parameters

Name	Type	Required	Description
eventName	String	Yes	Name of the event for which the handler is set. The supported events are listed in "Vocalcom.UCCore.addHandler" on page 10.
executeFirst	Boolean	Yes	**true** to execute before the standard handlers, **false** to execute after the standard handlers.
handlerFunction	Function	Yes	The handler function is invoked when any of the supported events

Name	Type	Required	Description
			are triggered.

Vocalcom.UCCore.removeCustomHandler

Removes the subscriber from the events.

Syntax

Vocalcom.UCCore.removeCustomHandler(eventName, handlerFunction);

Parameters

Name	Type	Required	Description
eventName	String	Yes	Name of the event for which the handler is set. The supported events are listed in "Vocalcom.UCCore.addHandler" on page 10.
handlerFunction	Function	Yes	The handler function that is to removed.

Vocalcom.UCCore.removeAllCustomHandlers

Removes the subscribers from the events.

Syntax

Vocalcom.UCCore.removeAllCustomHandlers(eventName);

Parameters

Name	Type	Required	Description
eventName	String	Yes	Name of the event for which the handlers are set. The supported events are listed in "Vocalcom.UCCore.addHandler" on page 10.

Vocalcom.UCCore.init

Initializes the UCCore API with the provided parameters.

Syntax

Vocalcom.UCCore.init (params);

Parameters

Name	Type	Required	Description
params	Object	Yes	Initialization parameters.
params.proxyConfig	String	Yes	ProxyhermesConfig URL.
params.configCode	String	Yes	Proxy code located in **ProxyConfig.xml** file.
params.agentStation	String	No	Agent extension (WebRTC or free station).

Name	Type	Required	Description
params.phoneNumber	String	No	Agent phone number.
params.jsFiles	String	No	Comma-separated list of external JS files to load.
params.login	String	Yes (if SSO is disabled)	Hermes agent login.
params.password	String	Yes (if SSO is disabled)	Hermes agent password.
params.language	String	No	Language used by Hermes agent (by default: English). Either from the CRM plugin or default language.
params.customerId	Number	No (if SSO is enabled)	Hermes site ID.

Name	Type	Required	Description
		Otherwise Yes	

Vocalcom.UCCore.getGlobalContext

Returns a javascript object that contains all the data and the state of the agent.

You must call this method before each use, to avoid using old data.

Syntax

```
Vocalcom.UCCore.getGlobalContext();
```

Vocalcom.UCCore.loginAgent

Connects the agent to a station.

Syntax

Vocalcom.UCCore.loginAgent(station);

Parameters

Name	Type	Required	Description
station	Number	Yes	The telephony station of the agent.

Vocalcom.UCCore.logoutAgent

Disconnects the agent.

Syntax

```
Vocalcom.UCCore.logoutAgent();
```

Vocalcom.UCCore.requestReady

Request to make the agent available.

Syntax

```
Vocalcom.UCCore.requestReady();
```

Vocalcom.UCCore.requestPause

Request to make the agent unavailable.

Syntax

```
/**
 * request a pause
 * @param {number} pauseCode
 * @param {number} typeAction - optional (0 by default)
 */
Vocalcom.UCCore.requestPause(pauseCode, typeAction = 0)
```

Parameters

Name	Type	Required	Description
pauseCode	Number	No	The pause code. The pause codes are stored in the global context and accessible at any time.

Name	Type	Required	Description
			`var context = Vocalcom.UCCore.getGlobalContext();` `var pauseCodes = context.pauseCodes;`
typeAction	Number	No	The action type: one of the following values. `var ACTION_TYPES = { PAUSE: 0, DISCONNECT: 1, CHANGE_USER: 2, REFRESH: 3, EXIT: 4 }`

Vocalcom.UCCore.getPauseCodes

Returns the list of agent pause codes, grouped by categories.

Syntax

Vocalcom.UCCore.getPauseCodes()

Example

Example of a returned object:

```
{
        group1 : [
            {
                GroupName: group1,
                Duration: 0,
                Description: "Lunch",
                Code: 0,
                AllowedDuration: -1
            }, ...
        ],
        ...
}
```

Vocalcom.UCCore.Telephony.startQueue

Starts one or several telephony queues. The queue IDs are separated by ",".

Syntax

Vocalcom.UCCore.Telephony.startQueue(queueIds);

Parameters

Name	Type	Required	Description
queueIds	String	Yes	The queue IDs. The queues are stored in the global context and are accessible at any time. `var context = Vocalcom.UCCore.getGlobalContext();` `var queueList= context.queueList;` `for (var i = 0; i < queueList.length; i++) {` `if (Number(queueList[i].type) == 0) { // type = 0 : inbound queue, 1: outbound queue` `Vocalcom.UCCore.Telephony.startQueue(queueList[i].queueId);`

Name	Type	Required	Description
			<pre>} }</pre>

Vocalcom.UCCore.Telephony.stopQueue

Stops one or several telephony queues. The queue IDs are separated by ",".

Syntax

Vocalcom.UCCore.Telephony.stopQueue(queueIds);

Parameters

Name	Type	Required	Description
queueIds	String	Yes	The queue IDs. The queues are stored in the global context and are accessible at any time. `var context = Vocalcom.UCCore.getGlobalContext();` `var queueList= context.queueList;` `for (var i = 0; i < queueList.length; i++) {` `if (Number(queueList[i].type) == 0) { // type = 0 : inbound queue, 1: outbound queue` `Vocalcom.UCCore.Telephony.stopQueue(queueList[i].queueId);`

Name	Type	Required	Description
			<pre> } }</pre>

Vocalcom.UCCore.Telephony.selectManualCampaign

Tells the system which manual campaign to use for manual calls and CRM click to call.

Syntax

Vocalcom.UCCore.Telephony.selectManualCampaign(campaignId);

Parameters

Name	Type	Required	Description
campaignId	String	Yes	The manual campaign ID. The manual campaigns are stored in the global context and are accessible at any time. `var context = Vocalcom.UCCore.getGlobalContext();` `var manualCampaigns = context.manualCampaigns;` `for (var i = 0; i < manualCampaigns.length; i++) {` `console.log("CampaignId = " + manualCampaigns[i].campId + ",`

Name	Type	Required	Description
			`Campaign Name = " + manualCampaigns[i].description + ", Country Code = " + manualCampaigns[i].countryCode); }`

Vocalcom.UCCore.Telephony.manualCall

Makes a manual call.

Syntax

```
Vocalcom.UCCore.Telephony.manualCall(phoneNumber);
```

Parameters

Name	Type	Required	Description
phoneNumber	Number	Yes	The phone number to call.

Vocalcom.UCCore.Telephony.internalCall

Makes an internal call (Call an agent).

Syntax

Vocalcom.UCCore.Telephony.internalCall(agentId);

Parameters

Name	Type	Required	Description
agentId	Number	Yes	The agent code.

Vocalcom.UCCore.Telephony.holdCall

Puts the call on hold.

Syntax

```
Vocalcom.UCCore.Telephony.holdCall();
```

Vocalcom.UCCore.Telephony.retrieveCall

Resumes the call.

Syntax

```
Vocalcom.UCCore.Telephony.retrieveCall();
```

Vocalcom.UCCore.Telephony.hangupCall

Hangs up the call.

Syntax

```
Vocalcom.UCCore.Telephony.hangupCall();
```

Vocalcom.UCCore.Telephony.setCallDisposition

Qualifies the call.

Syntax

```
/**
 * @param {string} timezone - timezone value
 */

Vocalcom.UCCore.Telephony.setCallDisposition(dispositionCode, dispositionDetailCode, comment, callbackTime,
callbackNumber, callbackValidity, timezone)
```

Parameters

Name	Type	Required	Description
dispositionCode	Number	Yes	Status code.
dispositionDetailCode	Number	No	Status detailed code.
comment	String	No	Free agent comment saved with the session qualification.

Name	Type	Required	Description
callbackTime	Number	No	For callback and personal callback code, specify the datetime for callback (format yyyyMMddHHmm).
callbackNumber	Number	No	Phone number to dial on callback.
callbackValidity	Number	No	Validity of the callback from its scheduled time.
timezone	String	No	Time zone value (one of the supported time zone values). For more information, you can refer to "Vocalcom.UCCore.getTimeZonesInfo" on page 103.

Vocalcom.UCCore.Telephony.blindTransferToPhoneNumber

Makes a blind transfer to a phone number.

Syntax

Vocalcom.UCCore.Telephony.blindTransferToPhoneNumber(phoneNumber);

Parameters

Name	Type	Required	Description
phoneNumber	Number	Yes	The phone number to transfer the call to.

Vocalcom.UCCore.Telephony.blindTransferToAgent

Makes a blind transfer to an agent.

Syntax

Vocalcom.UCCore.Telephony.blindTransferToAgent(agentId);

Parameters

Name	Type	Required	Description
agentId	Number	Yes	The agent ID to transfer the call to.

Vocalcom.UCCore.Telephony.transferToCampaign

Makes transfer to a campaign.

Syntax

Vocalcom.UCCore.Telephony.transferToCampaign(campaignId);

Parameters

Name	Type	Required	Description
campaignId	String	Yes	The campaign ID to transfer the call to. `var context = Vocalcom.UCCore.getGlobalContext();` `var manualCampaigns = context.manualCampaigns; // manual campaigns` `var inboundCamps = context.inboundCamps;  // inbound campaigns` `var agentCampaigns = context.agentCampaigns; // all other campaigns`

Vocalcom.UCCore.Telephony.consultPhoneNumber

Consults a phone number.

Syntax

```
Vocalcom.UCCore.Telephony.consultPhoneNumber(phoneNumber);
```

Parameters

Name	Type	Required	Description
phoneNumber	Number	Yes	The phone number to consult.

Vocalcom.UCCore.Telephony.consultAgent

Consult an agent.

Syntax

Vocalcom.UCCore.Telephony.consultAgent(agentId);

Parameters

Name	Type	Required	Description
agentId	Number	Yes	The agent ID to consult.

Vocalcom.UCCore.Telephony.alternate

During a consult, alternates the active agent line.

Syntax

```
Vocalcom.UCCore.Telephony.alternate();
```

Vocalcom.UCCore.Telephony.conference

Initiates a conference between the three parts.

Syntax

Vocalcom.UCCore.Telephony.conference();

Vocalcom.UCCore.Telephony.cancelConference

Cancels the current conference.

Syntax

```
Vocalcom.UCCore.Telephony.cancelConference();
```

Vocalcom.UCCore.Telephony.cancelConsult

Cancels the current consult. Hangups the third party.

Syntax

```
Vocalcom.UCCore.Telephony.cancelConsult();
```

Vocalcom.UCCore.Telephony.callAgain

Allows to redial the same customer (a new session will not be created).

Syntax

Vocalcom.UCCore.Telephony.callAgain(phoneNumber);

Parameters

Name	Type	Required	Description
phoneNumber	Number	No	The phone number to call.

Vocalcom.UCCore.Telephony.extendWrapup

Extend the wrap up time when the agent configuration is set as automatic ready.

Syntax

```
Vocalcom.UCCore.Telephony.extendWrapup();
```

Vocalcom.UCCore.Telephony.playDTMF

Send the DTMF sequence on the line.

Syntax

Vocalcom.UCCore.Telephony.playDTMF(sequence);

Parameters

Name	Type	Required	Description
sequence	String	Yes	The DTMF string to send.

Vocalcom.UCCore.Telephony.record

Starts recording the agent call that is in progress.

Syntax

```
Vocalcom.UCCore.Telephony.record();
```

Vocalcom.UCCore.Telephony.stopRecord

Stops the record in progress.

Syntax

```
Vocalcom.UCCore.Telephony.stopRecord();
```

Vocalcom.UCCore.Telephony.callPreview

Calls the current previewed customer.

Syntax

```
Vocalcom.UCCore.Telephony.callPreview();
```

Vocalcom.UCCore.Telephony.stopOutboundContext

Stops the outbound context.

Syntax

```
Vocalcom.UCCore.Telephony.stopOutboundContext();
```

Vocalcom.UCCore.Telephony.stopInboundContext

Stops the inbound context

Syntax

```
Vocalcom.UCCore.Telephony.stopInboundContext();
```

Vocalcom.UCCore.Telephony.softphoneAccept

Accepts a WebRTC call.

Syntax

```
Vocalcom.UCCore.Telephony.softphoneAccept();
```

Vocalcom.UCCore.Telephony.softphoneReject

Rejects a WebRTC call.

Syntax

```
Vocalcom.UCCore.Telephony.softphoneReject();
```

Vocalcom.UCCore.Telephony.callFromHistory

Starts a new call from the history of the recent call (the attached CRM record is poped-up and attached to the new call).

It populates the **context.callInfo.objectFromHistory**

Syntax

Vocalcom.UCCore.Telephony.callFromHistory(phoneNumber, objectId, objectType, objectDescription)

objectFromHistory object

```
context.callInfo.objectFromHistory = {
        objectId,
        objectType: ,
        description
    };
```

Vocalcom.UCCore.Telephony.transfer

Transfers the call during consultation.

Syntax

Vocalcom.UCCore.Telephony.transfer()

Vocalcom.UCCore.Telephony.getFirstCallStartTime

Gives the start time of the ongoing call.

Syntax

```
/**
 * Gives the start time of the ongoing call
 * @returns {Date}
 */
Vocalcom.UCCore.Telephony.getFirstCallStartTime()
```

Vocalcom.UCCore.Telephony.getFirstCallEndTime

Gives the end time of the finished call.

Syntax

```
/**
 * Gives the end time of the finished call
 * @returns {Date}
 */
Vocalcom.UCCore.Telephony.getFirstCallEndTime()
```

Vocalcom.UCCore.Telephony.closeSession

Closes the session of the current call.

Syntax

```
/**
 * Closes the session of the current call
 */
Vocalcom.UCCore.Telephony.closeSession()
```

Vocalcom.UCCore.Telephony.isRecordingAllowed

Indicates if the recording is allowed for the current state of the ongoing call.

Syntax

```
/**
 * indicates if the recording is allowed for the current state of the ongoing call
 * @returns {boolean}
 */
Vocalcom.UCCore.Telephony.isRecordingAllowed()
```

Vocalcom.UCCore.Telephony.onBeforeSetCallDisposition

This method can be implemented to execute custom code when the agent qualifies a call.

Example

To ensure data integrity, a company wants their agents to fill a specific field in the CRM before qualifying a call in HUCC CTI.

Syntax

```
Vocalcom.UCCore.Telephony.onBeforeSetCallDisposition = function () {
  return new Promise((resolve, reject) => {

    if( My logic is ok )
      resolve(true);
    else
     resolve(false);
     // if exception
        reject(error);
 });
};
```

Result

Returns a promise.

If it is evaluated to true, the agent can qualify the call. If not, the agent will stay in after call status.

Vocalcom.UCCore.emitCallerSearchResult

Communicates a list of records to the CTI Adapter.

Syntax

Vocalcom.UCCore.emitCallerSearchResult(results);

Result

The result is an array of objects:

```
[
    {
objectId: value, // object ID in the CRM
objectType: value, // object type in the CRM (contact, account, etc.),
description: value // objet description (contact name and first name for example)
    },
...
]
```

The client records search results are available at any time in the **localStorage**:

```
var context = Vocalcom.UCCore.getGlobalContext();
var results =   context.callInfo.searchCallerResult;
```

Vocalcom.UCCore.attachCRMObjectToCall

Attaches the record chosen by the user in the CTI Adapter to the current call, when several records match the number.

Syntax

```
Vocalcom.UCCore.attachCRMObjectToCall(objectId);
```

Vocalcom.UCCore.openCRMObject

Previews a record in the CRM.

Syntax

```
/**
 *
 * @param {string} objectId
 * @param {string} objectType
 */
Vocalcom.UCCore.openCRMObject(objectId, objectType);
```

Vocalcom.UCCore.searchForCaller

Asks the CRM plugin to search and return the CRM records matching the caller phone number (triggers **OnSearchForCaller** event).

Syntax

```
Vocalcom.UCCore.searchForCaller();
```

Vocalcom.UCCore.refreshSearchForCaller

Asks the CRM to refresh the search results (triggers **OnSearchForCaller** event)

Syntax

Vocalcom.UCCore.refreshSearchForCaller()

Vocalcom.UCCore.getAgents

Returns the list of connected agents (promise).

Syntax

Vocalcom.UCCore.getAgents()

Example

```
Vocalcom.UCCore.getAgents()
 .then((res) => {
   console.log(res);
   this.agentList = res
 }).catch((error: any) => { console.log(error); })
```

Vocalcom.UCCore.selectInboundContext

Selects the inbound context.

Syntax

Vocalcom.UCCore.selectInboundContext()

Vocalcom.UCCore.selectOutboundContext

Selects the outbound context.

Syntax

Vocalcom.UCCore.selectOutboundContext()

Vocalcom.UCCore.getRecentCalls

To be implemented in the CRM plugin to return a list of the recent calls for the current agent.

Syntax

Vocalcom.UCCore.getRecentCalls()

Result

Returns a promise:

```
// MS Dynamics example

    var getPhoneCalls = function () {
        const userid = localStorage.USERID;
        return Microsoft.CIFramework.searchAndOpenRecords("phonecall",
`?$select=phonenumber,actualdurationminutes,_regardingobjectid_value&$filter=_createdby_value eq
'${userid}'&$orderby=actualend desc&$top=20`, true);
    };

    Vocalcom.UCCore.getRecentCalls = function () {
        return new Promise((resolve, reject) => {
                getPhoneCalls().then(
```

```
                                function success(result) {
                                    res = JSON.parse(result);
                                    var calls = [];
                                    for (const [key, call] of Object.entries(res)) {
                                        var d = new Date(call["actualend"]);
                                        calls.push({
                                            callerName: call["_regardingobjectid_
value@OData.Community.Display.V1.FormattedValue"],
                                            objectId: call["_regardingobjectid_value"],
                                            objectType: call["_regardingobjectid_
value@Microsoft.Dynamics.CRM.lookuplogicalname"],
                                            callerPhoneNumber: call["phonenumber"],
                                            startTime: call['huuc_hermes_call_starttime'],
                                            endTime: call["huuc_hermes_call_stoptime"],
                                            callDirection: call['directioncode'] ? 1: 0 // 0: inbound, 1:
outbound
                                        });
                                    }
                                    resolve(calls);
                                },
                                function (error) {
                                    reject(error);
                                }
                            );
                    });
            };
```

Vocalcom.UCCore.expand

Expands (maximize) the CTI Adapter widget.

Syntax

Vocalcom.UCCore.expand()

Vocalcom.UCCore.collapse

Collapses (minimize) the CTI Adapter widget.

Syntax

Vocalcom.UCCore.collapse()

Vocalcom.UCCore.emitCRMObjectAttachedToCall

Sends the CRM object attached to the call to the CTI Adapter.

Syntax

```
/**
 * Sends to the CTI Adapter the CRM object attached to the call
 * @param {string} objectId
 * @param {string} description
 */
Vocalcom.UCCore.emitCRMObjectAttachedToCall(objectId, description)
```

Vocalcom.UCCore.getTimeZonesInfo

Gets the list of timezones supported by the CTI Adapter.

Syntax

```
/**
 * get the list of timezones supported by the CTI Adapter
 * @returns {Promise} - {timezoneList: array, defaultTimezone: object}
 * */

Vocalcom.UCCore.getTimeZonesInfo()
```

Vocalcom.UCCore.getCallbackTime

Calculates the callback time.

Syntax

```
/**
 * Calculate the callback time
 *
 * @param {Date} date
 * @param {String} timezone - ex : "Romance Standard Time
 * @returns {Promise} - callback time ex: "202211071248"
 */
Vocalcom.UCCore.getCallbackTime(date, timezone)
```

Vocalcom.UCCore.getListContact

Gets the list of contacts, grouped together. By default, returns the contacts declared in Hermes.

This method can be overridden.

Syntax

```
/**
 *  Get the list of contacts grouped (By default returns the contacts declared in Hermes )
 *  @returns {Promise} - an array of contact
 */
Vocalcom.UCCore.getListContact()
```

Example

Example of implementation:

```
Vocalcom.UCCore.getListContact = function () {
    return new Promise((resolve, reject) => {
        var listContact = [];
        listContact["group 1"] = [
                            {
```

```
                        Name: "contact name",
                        Phone: "contact number",
                        Email: "agent@vocalcom.com"
                            }, ...
                       ];
            resolve(listContact);
        });
    };
```

Vocalcom.UCCore.isMediaActive

Indicates if the agent is processing a media (call, chat, etc.).

Syntax

```
/**
 *  indicates if the agent is processing a media (call, chat, ...)
 *  @returns {boolean}
 */
Vocalcom.UCCore.isMediaActive()
```

Vocalcom.UCCore.getVersion

Returns the HUCC version number in the CRM CTI log-in screen.

Syntax

```
Vocalcom.UCCore.getVersion();
```

Vocalcom.UCCore.requestTempNotReady

Requests the CTI to set the agent as not ready.

It is used to give the agent time for manual dialing.

Syntax

```
Vocalcom.UCCore.requestTempNotReady();
```

Vocalcom.UCCore.cancelTempNotReady

Cancels the previous **TempNotReady** request (see "Vocalcom.UCCore.requestTempNotReady" on the previous page).

Syntax

```
Vocalcom.UCCore.cancelTempNotReady();
```

Vocalcom.UCCore.isMaster

Allows to identify the master window: returns **true** when the current window is the master, **false** otherwise. Avoids executing several times automatic processes that must be executed once: automatic processes will be executed only from the master window.

For example, if, in a CRM, the CTI is open in two windows at the same time, and a call object is created, only one object will be created (from the master window).

Syntax

```
/**
 *
 * @returns true when the current window is the master, false otherwise
 */
Vocalcom.UCCore.isMaster()
```

Vocalcom.UCCore.restoreContext

Restores all conversations (sessions) for all medias (for example after refreshing the browser).

Syntax

```
Vocalcom.UCCore.restoreContext();
```

Vocalcom.UCCore.getStatusDescription

Gets status description for specific call status codes.

Syntax

```
Vocalcom.UCCore.getStatusDescription(group, code, detail);
```

Parameters

Name	Type	Required	Description
group	Number	Yes	Call status group
code	Number	Yes	Call status code

Name	Type	Required	Description
detail	Number	No	Call status detail

Result

It returns Array<string>: [statusDescription, statusDetailDescription]

Vocalcom.UCCore.createCRMObject

Creates a CRM object: contact, account, etc. (triggers the **OnCreateCRMObject** event).

Syntax

Vocalcom.UCCore.createCRMObject(options);

Parameters

Name	Type	Required	Description
options	Object	Yes	The options for creating the CRM object.
options.contextType	Number	Yes	Context type.
options.sessionId	String	Yes	Session ID.
options.objectType	String	Yes	Object type.
options.params	Object	No	Parameters for creating the CRM object.

Vocalcom.UCCore.refreshSearchForMediaSession

Refreshes the search results for a media session, based on the context type and session ID (triggers the **OnRefreshSearchForMediaSession** event).

Syntax

Vocalcom.UCCore.refreshSearchForMediaSession(contextType, sessionId);

Parameters

Name	Type	Required	Description
contextType	Number	Yes	Type of context.
sessionId	String	Yes	Media session ID.

Vocalcom.UCCore.searchForContactByInputValue

Searches for a contact depending on the input (triggers **OnCallerSearchInputChange** event).

Syntax

Vocalcom.UCCore.searchForContactByInputValue(options);

Parameters

Name	Type	Required	Description
options	Object	Yes	Search options.
options.contextType	Number	Yes	Type of context.
options.sessionId	String	Yes	Session ID.
options.inputValue	String	Yes	Input value to search for.

Vocalcom.UCCore.restoreMediaSession

Restores a specific media session (triggers the **OnRestoreMediaSession** event).

Syntax

```
Vocalcom.UCCore.restoreMediaSession(contextType, sessionId);
```

Parameters

Name	Type	Required	Description
contextType	Number	Yes	Type of context.
sessionId	String	Yes	Session ID.

Vocalcom.UCCore.selectCurrentSessionForContext

Used by UI to select the current context session (triggers the **OnSelectCurrentSessionForContext** event).

Plugins listen to this event to perform actions on the current context session. For example: it allows an agent to switch from one conversation to another and retrieves their respective session.

Syntax

Vocalcom.UCCore.selectCurrentSessionForContext(contextType, sessionId);

Parameters

Name	Type	Required	Description
contextType	Number	Yes	Type of context.
sessionId	String	Yes	Session ID.

Vocalcom.UCCore.emitMediaRelatedCRMObjects

Emits media related CRM objects (triggers the **OnMediaRelatedCRMObjects** event).

Syntax

Vocalcom.UCCore.emitMediaRelatedCRMObjects(contextType, sessionId, listCRMObjects);

Parameters

Name	Type	Required	Description
contextType	Number	Yes	Type of context.
sessionId	String	Yes	Session ID.
listCRMObjects	Array	Yes	List of the CRM objects.

Vocalcom.UCCore.emitCRMObjectAttachedToMedia

Allows the plugin to ask the CTI to attach the CRM object to the media (triggers the **OnCRMObjectAttachedToMedia** event).

Syntax

Vocalcom.UCCore.emitCRMObjectAttachedToMedia(contextType, sessionId, CRMObject);

Parameters

Name	Type	Required	Description
contextType	Number	Yes	Type of context.
sessionId	String	Yes	Session ID.
CRMObject	Object	Yes	The CRM object attached to the media.

Vocalcom.UCCore.attachCRMObjectToMedia

Allows the CTI to ask the plugin to attach a CRM object to a media (triggers the **OnAttachCRMObjectToMedia** event).

Syntax

Vocalcom.UCCore.attachCRMObjectToMedia(contextType, sessionId, objectId);

Parameters

Name	Type	Required	Description
contextType	Number	Yes	Type of the media context.
sessionId	String	Yes	Media session ID.
objectId	String	Yes	ID of the CRM object to attach

Vocalcom.UCCore.getMediaChannels

Retrieves all available contexts (telephony, social networks, etc.).

Syntax

```
Vocalcom.UCCore.getMediaChannels{};
```

Result

Returns an array of the available contexts.