Cisco Remote Expert Agent Desktop SDK Developer Guide

Release 1.9.5

May 24, 2015

Note All advertising materials mentioning features or use of this software must display the following acknowledgement: "*This product includes software developed by the University of California, Berkeley and its contributors.*"

Overview

This document provides information about the Cisco Remote Expert Agent Desktop (READ) software development kit (SDK). The READ SDK allows Cisco READ, which is the application that agents use to collaborate with customers, to be integrated into customer applications.

Topics in this guide include:

Acronyms

The following acronyms are used in this guide:

- API – Application Programming Interface
- CTI – Computer Telephony Integration
- DN – Directory Number
- eREAD – eRemote Expert Agent Desktop (uses Cisco Finesse)
- IEC – Interactive Experience Client
- MBR – Message Bus Receiver
- RE – Remote Expert
- REAC – Remote Expert Administration Console
- READ – Remote Expert Agent Desktop (uses Cisco Agent Desktop)
- REIC – Remote Expert Interactive Applications Control
- REM – Remote Expert Manager
- RESC – Remote Expert Session Controller
- SDK – Software Development Kit
- UI – User Interface
- VNC – Virtual Network Computing
- VTA – Virtual Teller Agent

Requirements

Libraries

Table 1 Libraries

Library	Comment
readsdk.min.js	Core library for READ SDK
minpubsub.js	Supporting library for pubsub mechanism

Library	Comment
jabberwerx.js	BOSH/XMPP client for Finesse notification server
finesse-rest.js	Client for Finesse REST API service
jQuery 1.9.x or above	Cross-platform JavaScript library that is designed to simplify the client-side scripting of HTML and is free, open-source software licensed under the MIT License

Client Requirements

Agents and administrators use web browsers on their desktops or laptops, referred here to as “clients”, to access Cisco Finesse/READ. No software is required to be installed on the clients.

Since READ SDK is built on top of Cisco Finesse technology, the agent’s desktop or laptop must meet the minimum requirement of Finesse:

1. The minimum supported screen resolution is 1280 x 1024.
2. The SDK and Call Control API are tested against the following operation systems and browsers for clients:

Table 2 Tested OS and Browsers

Operating System	Browser and Recommended Version
Windows 7	Firefox (version 30 or later) Internet Explorer 10.0 Internet Explorer 11.0 **Note** IE 11 requires Windows 7 SP1.

.Although there are no special requirements for the network, make sure that network characteristics should not exceed the following thresholds for optimal performance:

- Latency: 80 ms (round-trip) between Finesse servers and 200 ms (round-trip) from client to Finesse server
- Jitter: 2 ms
- Packet loss: 0.5%

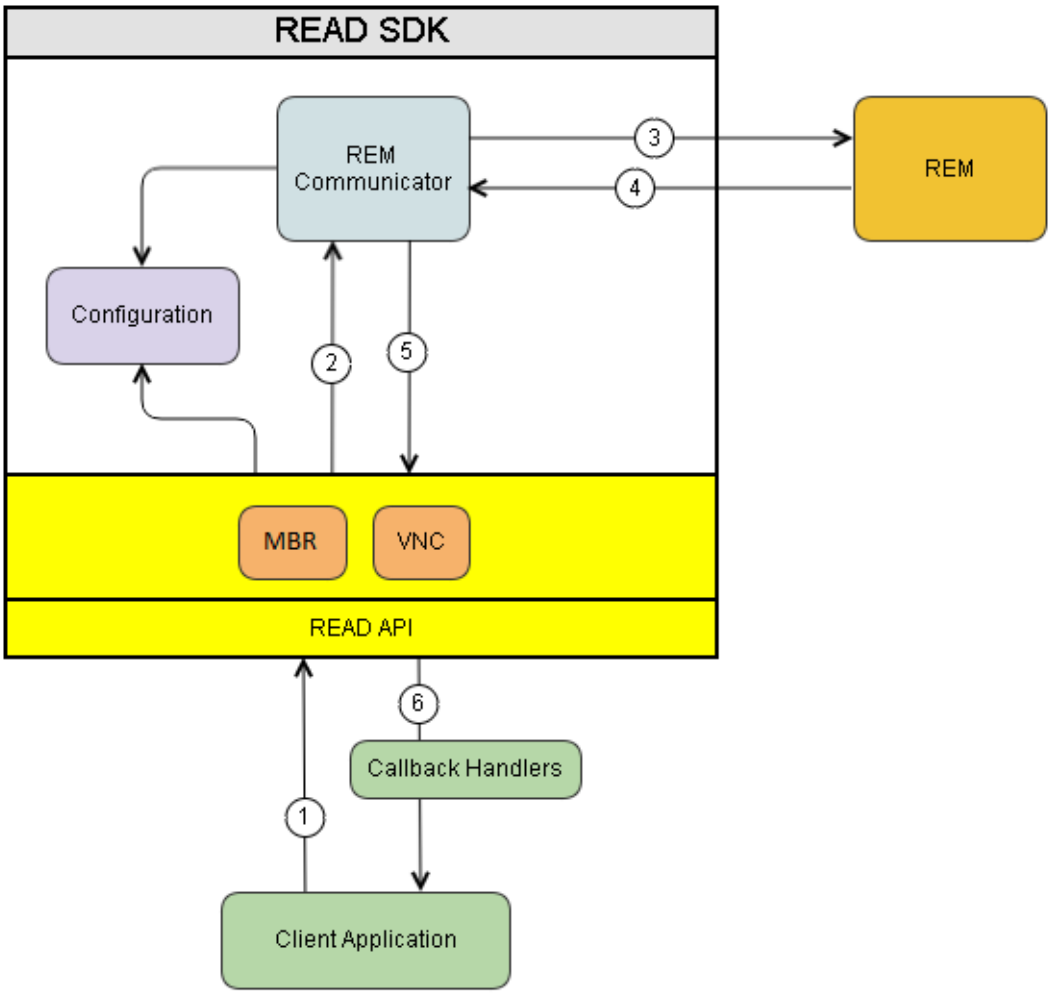
For more information, please refer to the official Finesse document at:
http://www.cisco.com/c/en/us/td/docs/voice_ip_comm/cust_contact/contact_center/finesse/finesse_1051/installation/guide/CFIN_BK_CA0E68AE_00_cisco-finesse-installation-and-upgrade-1051.html

Architecture

The purpose of READ SDK is to provide a layer of abstraction over the features provided by the READ application of the REM stack. Currently, READ SDK contains the control logic of READ for the co-browsing feature, which segregates controller code from the client UI code. With the help of this SDK client, UI code can be independent from the core control logic. As a result, the SDK can be easily integrated with client applications.

The figure below illustrates the architecture of READ SDK and interactions among components. Refer to the list below the figure for explanations of each interaction.

Figure 1 SDK Modules Interaction



1. Client application invokes the READ API (e.g., READ.Session.start(), READ.VNC.start(), etc.).
2. The relevant module of the READ API communicates with REM (READ server component) with the help of the REM Communicator (Ajax Module).
3. REM Communicator forwards the request to REM.

4. REM processes the request and sends the response to the REM Communicator via callback functions.
5. REM Communicator forwards the response to the READ API.
6. READ API invokes the corresponding call back handler function depending on the response (i.e. ‘success’ or ‘failure’).

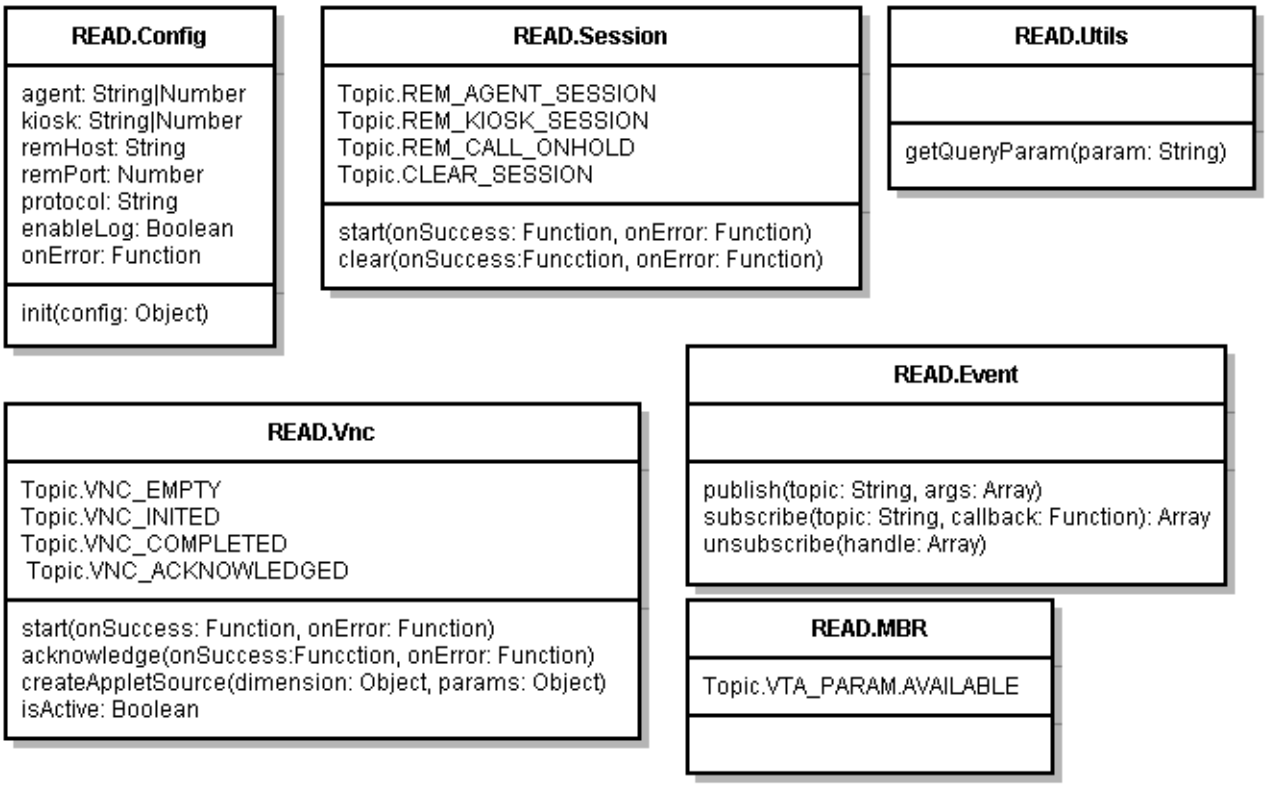
SDK Component Details

Introduction

The READ SDK is available under the namespace of READ. READ APIs are grouped into Configuration (Config), Session, Utility (Utils), Event, VNC and MBR modules. The Config API is used to configure READ SDK, and the Session API is used to manage REM sessions within client’s application. The VNC and MBR modules are used to manage VNC co-browsing and VTA message jobs respectively. The main classes to be used in construction of UI are Config and Session.

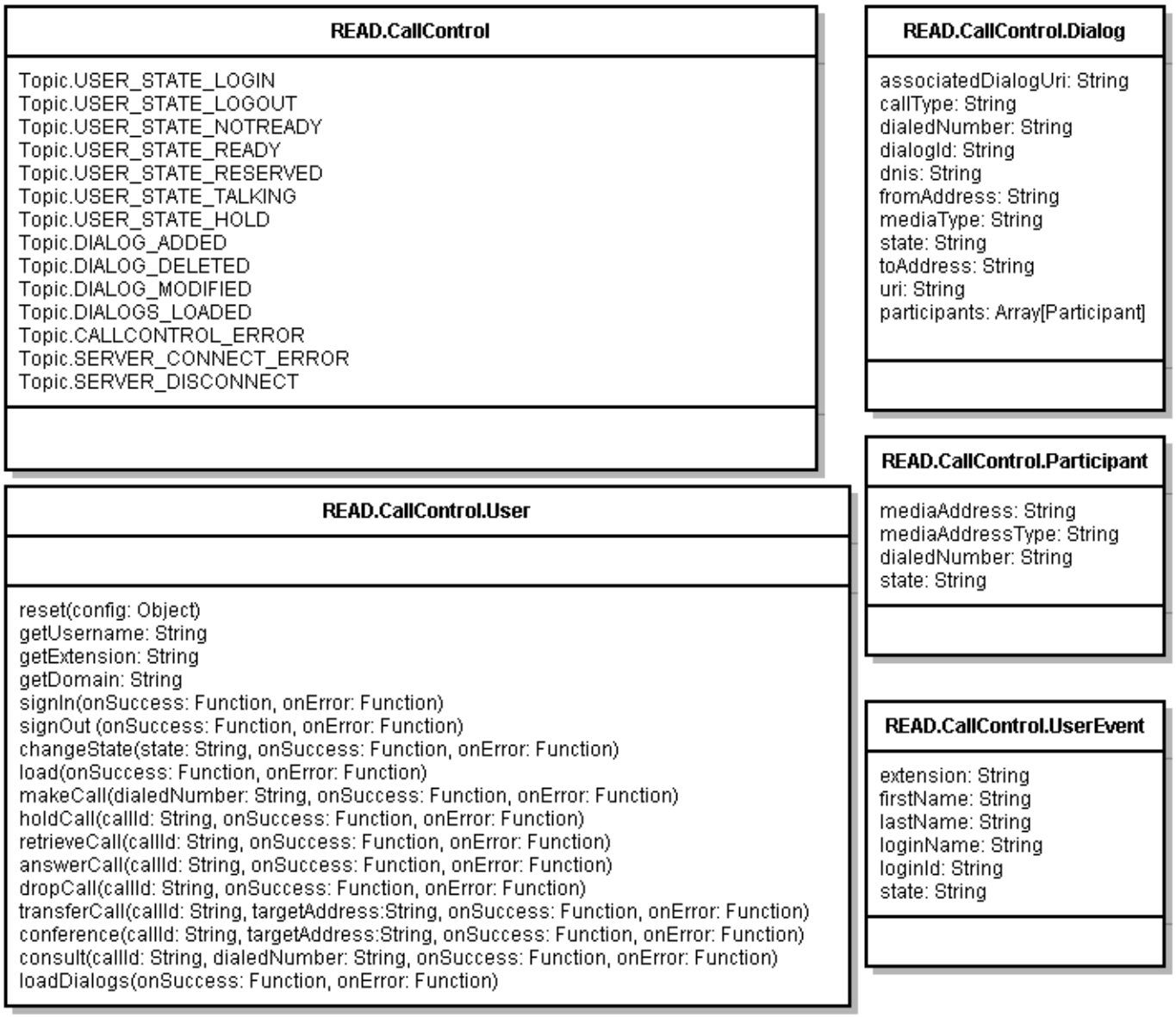
This below figure provides an overview of READ SDK modules.

Figure 2 Overview of READ SDK Modules



The Call Control API consists primarily of the User, Dialog and Participant classes. The User object is composed of Dialogs and each dialog in turn has a set of Participants. The User object represents an agent and includes information about the user such as the state, dialogs he/she is participating in. The class representation of these artifacts is presented below:

Figure 3 Overview of READ Call Control Module



Configuration Module

init (Initialize READ SDK in Client's Application)

- Prerequisite: The agent has received a call or made a call.
- API Description: This API is used to initializes the configuration for READ SDK.
- Syntax:

```
(static) READ.Config.init (config)
```

- Explanation of parameters:

Table 3 config Object

Name	Type	Description
config	Object	See Properties table below

Table 4 config Properties

Name	Type	Attributes	Description
agent	String | Number		The directory number of the agent/expert
kiosk	String | Number	<optional>	The directory number of kiosk or calling number
remHost	String	<optional>	The IP/host address of REM server
remPort	String | Number	<optional>	The port of REM server (80, 8080, 443 or 8443)
enableLog	Boolean	<optional>	true to enable console logging; false to disable logging. The default is false.
onError	function	<optional>	A generic callback to be invoked when any request to REM server fails

- Return Value: None
- Trigger Event: None
- Example:

```
READ.Config.init({
agent: agentDn, // Mandatory, agent directory number
kiosk: kioskDn, // Optional, kiosk directory number
remHost: 'rem.cisco.com', // Optional
remPort: '8443', // Optional
enableLog: true, // Optional
onError: function(error) {
// Inform user
},
});
```

Event Module

publish (Publish a REM event occurrence)

- Prerequisite: READ SDK is initialized successfully, and a call is made.
- API Description: This API is used to publish the occurrence of a REM event. This is used internally in the SDK to publish occurrences of key events related to session, call, and other jobs like co-browsing. However, this can be also be used by external developers to publish their events.
- Syntax:

```
(static) READ.Event.publish (topic, args)
```

- Explanation of parameters:

Table 5 publish Parameters

Name	Type	Description
topic	String	the topic to be published
args	Array	the arguments to be sent to the subscription callback

- Return Value: None
- Trigger Event: None
- Example:

```
// If it is detected that a REM session is not available during any point of time in a
call,
// the API can be used to indicate that a REM session is not available in the error
handler.
var details = error.details;
var msg = details.message;
var code = details.code;
if(code === READ.Constants.ErrorCode.NO_REM_SESSION) {
READ.Event.publish(READ.Session.Topic.REM_AGENT_SESSION, [false]);
}
```

subscribe (Subscribe to a REM Event Occurrence)

- Prerequisite: READ SDK is initialized successfully, and a call is made.
- API Description: This API is used to subscribe to be notified when a REM event is available.
- Syntax:

```
(static) READ.Event.subscribe (topic, callback)
```

- Explanation of parameters:

Table 6 subscribe Parameters

Name	Type	Description
topic	String	the topic to be subscribed
callback	function	the callback to be invoked when the event is published

- Return Value: The handler, which has type of “array”, can be used to disconnect this subscription.
- Trigger Event: None
- Example:

```
//Subscribe to be notified of agent availability
var sessionHandle = READ.Event.subscribe(READ.Session.Topic.REM_AGENT_SESSION,
function(isActive) {
If (!isActive) {
// Disable UI elements
}
});
```

unsubscribe (Unsubscribe to a REM Event Occurrence)

- Prerequisite: READ SDK is initialized successfully, and a call is made.
- API Description: This API is used to disconnect a subscription from a topic. Generally, its usage is not required.
- Syntax:

```
(static) READ.Event.unsubscribe (handle)
```

- Explanation of parameters:

Table 7 unsubscribe Parameters

Name	Type	Description
topic	String	the topic to be subscribed
callback	function	the callback to be invoked when the event is published

- Return Value: None
- Trigger Event: None
- Example:

```
READ.Event.unsubscribe(sessionHandle);
```

Session Module

start (Start Polling to Obtain REM Session Status)

- Prerequisite: READ SDK is initialized successfully, and a call is made.
- API Description: This API is used to inform READ SDK to start checking if agent session is established at REM server. This API is not required when integrated with the Call Control API.
- Syntax:

```
(static) READ.Session.start(onSuccess, onError)
```

- Explanation of parameters:

Table 8 start Parameters

Name	Type	Description
onSuccess	function	the callback to be invoked if the request succeeds
onError	function	the callback to be invoked if the request fails. The values returned include HTTP status code and response object from server. For example: { message="NO_ACTIVE_REM_SESSION", code=115, success=false}.

- Return Value: Please refer to the section of “Error Details” for more details.
- Trigger Event: None
- Example:

```
$(document).ready(function() {
READ.Session.start();
});
```

clear (Clear All Internal Polling for a Given REM Session)

- Prerequisite: READ SDK is initialized successfully, and a call is made.
- API Description: This API is used to clear all internal polling activities within READ SDK.
- Syntax:

```
(static) READ.Session.clear(onSuccess, onError)
```

- Explanation of parameters:

Table 9 clear Parameters

Name	Type	Description
onSuccess	function	the callback to be invoked if the request succeeds
onError	function	the callback to be invoked if the request fails. The values returned include HTTP status code and response object from server. For example: { message="NO_ACTIVE_REM_SESSION", code=115, success=false}.

- Return Value: Please refer to the section of “Error Details” for more details.
- Trigger Event: None
- Example:

```
// It is appropriate to be invoked when a call is dropped
$drop.click(function() {
READ.Session.clear();
// other activities
});
```

Subscription Topics of Session Class

- Prerequisite: READ SDK is initialized successfully.

- API Description: The topics of Session Class may be used via the subscribe API of READ.Event module to be notified of events related to REM session. The events are published internally by READ SDK. The user may choose to subscribe to these events in order to manage the UI state during those events.
- Syntax: Please refer the section of "Event Module" for more details.
- Explanation of parameters:

Table 10 topic Object

Name	Type	Description
topic	Object	Container for event topics/queue that can be subscribed

Table 11 Topic Properties

Name	Type	Description
REM_AGENT_SESSION	String	This is published when the agent part of session is available on REM server. The agent session becomes available when the agent has answered the call or made a call.
REM_CALL_ONHOLD	String	This is published when the call is placed on hold by agent.

- Return Value: None
- Trigger Event: None
- Examples:
  - Scenario 1: Subscribe to be notified when agent session is available or stopped

```
READ.Event.subscribe(READ.Session.Topic.REM_AGENT_SESSION, function(isActive) {
if(!isActive) {
if(READ.Vnc.isActive()) {
READ.Vnc.acknowledge();
}
document.getElementById('appletplace').innerHTML = "";
disableStartVncBtn();
disableStopVncBtn();
}
});
```

  - Scenario 2: Subscribe to be notified when call is placed on hold

```
READ.Event.subscribe(READ.Session.Topic.REM_CALL_ONHOLD, function(isOnHold) {
if(isOnHold) {
disableStartVncBtn();
disableStopVncBtn();
}
});
```

VNC Co-browsing Module

start (Start a VNC Co-browsing Job)

- Prerequisite: READ SDK is initialized successfully, and a call is made.
- API Description: This API is used to start a VNC co-browsing job in a READ session from client application.
- Syntax:

```
(static) READ.Vnc.start(onSuccess, onError)
```

- Explanation of parameters:

Table 12 start Parameters

Name	Type	Description
onSuccess	function	the callback to be invoked if the request succeeds
onError	function	the callback to be invoked if the request fails. The values returned include HTTP status code and response object from server. For example: { message="NO_ACTIVE_REM_SESSION", code=115, success=false}.

- Return Value: Please refer to the section of “Error Details” for more details.
- Trigger Event: None
- Example:

```
$startVncBtn.click(function() {
disableStartVncBtn();
READ.Vnc.start(function(data) {
console.info('Vnc request sent successfully');
}, function(error) {
console.error('Failed to start VNC. Status: ' + error.status + ', Cause: ' +
error.statusText);
alert("Co-browsing start failed. Reason: " + error.statusText);
enableStartVncBtn();
});
});
```

createAppletSource (Create an Applet Tag to Load VNC Client)

- Prerequisite: READ SDK is initialized, a call is answered, and the co-browsing job is in progress.
- API Description: This API is used to create an applet element to load VNC client as a Java applet from client application.
- Syntax:

```
(static) READ.Vnc. createAppletSource(dimension, params)
```

- Explanation of parameters:

Table 13 dimension Object

Name	Type	Description
dimension	Object	dimension of the applet

Table 14 dimension Properties

Name	Type	Description
width	Number	width of the applet
height	Number	height of the applet

Table 15 params Object

Name	Type	Description
params	Object	configuration parameters for the applet

Table 16 params Properties

Name	Type	Attribute	Description
codebase	String	<optional>	the URL to the location where the applet archive tightvnc-jviewer.jar is placed. If not provided, the default value will be “readsdk” relative to REM server.
iecIp	String		the IP address of IEC
iecPassword	String		the password to log into IEC
vncServerPort	String | Number		the port of VNC server in IEC

- Return Value: The html APPLET element, which has type of “string”
- Trigger Event: None
- Example:

```
READ.Event.subscribe(READ.Vnc.Topic.VNC_COMPLETED, function(job) {
var appletsource = READ.Vnc.createAppletSource({
width:100,
height:100
}, {
codebase: window.location.origin +'/FinesseAPIv3',
iecIp: job.IecIp,
iecPassword:job.IecPasswd,
vncServerPort:job.VncServerPort
} );
document.getElementById('appletplace').innerHTML = appletsource;
});
```

acknowledge (Start a VNC Co-browsing Job)

- Prerequisite: READ SDK is initialized, a call is answered, and the co-browsing job is in progress.
- API Description: This API is used to acknowledge the completion of VNC co-browsing job. Invoke the API to stop the job at REM server.

- Syntax:

```
(static) READ.Vnc. acknowledge(onSuccess, onError)
```

- Explanation of parameters:

Table 17 acknowledge Parameters

Name	Type	Description
onSuccess	function	the callback to be invoked if the request succeeds
onError	function	the callback to be invoked if the request fails. The values returned include HTTP status code and response object from server. For example: { message="NO_ACTIVE_REM_SESSION", code=115, success=false}.

- Return Value: Please refer to the section of “Error Details” for more details.
- Trigger Event: None
- Example:

```
READ.Event.subscribe(READ.Session.Topic.REM_AGENT_SESSION, function(isActive) {
if(!isActive) {
if(READ.Vnc.isActive()) {
READ.Vnc.acknowledge();
}
document.getElementById('appletplace').innerHTML = "";
disableStartVncBtn();
disableStopVncBtn();
}
});
```

Subscription Topics of VNC Class

- Prerequisite: READ SDK is initialized successfully.
- API Description: The topics of VNC class are the events published by the Vnc module that the user can subscribe to and be notified of events from Co-browsing job.
- Syntax: Please refer the section of “Event Module” for more details.
- Explanation of parameters:

Table 18 topic Object

Name	Type	Description
topic	Object	Container for event topics/queue to which can be subscribed

Table 19 topic Properties

Name	Type	Description
VNC_EMPTY	String	This is published when the SDK detects that there are no co-browsing jobs submitted to REM server.
VNC_INITED	String	This is published when the co-browsing job is accepted by REM server.

Name	Type	Description
VNC_COMPLETED	String	This is published when the VNC server has started
VNC_ACKNOWLEDGED	String	This is published when the Co-browsing job is stopped at REM server

- Return Value: None
- Trigger Event: None
- Example:

```
// Subscribe to co-browsing events
READ.Event.subscribe(READ.Vnc.Topic.VNC_INITED, function(job) {
//handle any UI state when job is accepted
});
READ.Event.subscribe(READ.Vnc.Topic.VNC_COMPLETED, function(job) {
var appletsource = READ.Vnc.createAppletSource({
width:100,
height:100
}, {
codebase: window.location.origin +'/FinesseAPI',
iecIp: job.IecIp,
iecPassword:job.IecPasswd,
vncServerPort:job.VncServerPort
} );
document.getElementById('appletplace').innerHTML = appletsource;
});
```

MBR Module

Subscription Topics of MBR Class

- Prerequisite: READ SDK is initialized successfully.
- API Description: This event is published when the parameters for VTA are available from REM server, and query string is constructed. This event publishes the Query String that can be used by event subscribers to customize behavior.
- Syntax: Please refer the section of “Event Module” for more details.
- Explanation of parameters:

Table 20 topic Object

Name	Type	Description
topic	Object	Container for event topics/queue to which can be subscribed

Table 21 topic Properties

Name	Type	Description
VTA_PARAM_AVAILABLE	String	the topic to subscribe to determine if the parameters to invoke VTA query string are available.

- Return Value: None
- Trigger Event: None
- Example:

```
// This event is not published till the query string parameters are not obtained from
REM
// server. Thereafter, it is notified regularly. To avoid repeated invocations of the
// functionality, the implementation needs to have a flag to verify that the query
string
// has been handled.
READ.Event.subscribe(READ.MBR.Topic.VTA_PARAM_AVAILABLE, function(querystring) {
console.info('VTA query string is available >> ', querystring);
// handle query string here
});
```

Error Details

The following table contains error codes for the Configuration (Config), Session, Utility (Utils), Event, VNC and MBR modules in READ APIs.

Table 22 Error Codes

Code	Message	Reason	SDK Constant
101	System error. Please try again later.	Internal REM error.	READ.Constants.ErrorCode. SYSTEM_ERROR
110	Illegal argument is passed.	Sent if agent directory number is missing in the parameter list sent to REM.	READ.Constants.ErrorCode. ILLEGAL_ARGUMENT
111	A collaboration request is still in progress.	Sent if an attempt is made by a collaborating agent to start a job that is already started by another agent.	READ.Constants.ErrorCode. REQUEST_AVAILABLE
114	Job request is already processed.		READ.Constants.ErrorCode. REQUEST_AVAILABLE
115	No active session was detected or available.	If a request is made without an active session in progress.	READ.Constants.ErrorCode. NO_REM_SESSION

Code	Message	Reason	SDK Constant
117	Command is invalid.		READ.Constants.ErrorCode. INVALID_COMMAND
118	Last command execution is not finished yet.		READ.Constants.ErrorCode. COMMAND_BARRED

Call Control Module

Introduction

The Call Control module has user object, which also includes callbacks for managing user state change and dialog event. The User object represents an agent and includes information about the user, such as roles, state of participating dialogs.

The Dialog object represents a dialog with participants. For the media type "voice", this object represents a call. A participant represents an internal or external user's Call Connection, or that user's leg of the call.

User—Sign In to Finesse

- Request Parameters:
  1. READ created User object.
  2. onSuccess(data): (Optional) Function callback to be invoked if request succeeds
  3. onError(error): (Optional) Function callback to be invoked if request fails. (Refer to the section Call Control API Error Format.).
- API Description: The User—Sign in to Finesse API allows a user to sign in to the CTI server. If the response is successful, the user is signed in to Finesse and is automatically placed in NOT_READY state.

  This API forces a sign-in. That is, if the user is already signed in, that user is authenticated via the sign-in process. If the user's credentials are correct, the user is signed in again but the user keeps the current state. For example, if a user signs in, changes state to Ready, and then signs in again, the user remains in Ready state.
- Response:
  - 202: Success
  - 400: Bad Request (for example, malformed or incomplete request, invalid extension)
  - 400: Parameter Missing
  - 401: Unauthorized (for example, the user is not authenticated in the Web Session)
  - 404: Not Found (for example, the user ID is not known)
  - 503: Service Unavailable (for example, the Notification Service is not running)
- Example Failure Response:

```
{
status: 404,
statusText: Not Found
details: [{
errorData: 4023,
```

```
errorType: User Not Found,
errorMessage: UNKNOWN_USER
}]
}
```

- Trigger Event: User notification.

User—Change Agent State

```
changeState(state: String, onSuccess: Function, onError: Function)
```

- Request Parameters:
  1. state (required): The new state that the user wants to be in (READY, NOT_READY)
  2. onSuccess(data): (Optional) Function callback to be invoked if request succeeds
  3. onError(error): (Optional) Function callback to be invoked if request fails. (Refer to the section Call Control API Error Format.)
- API Description: This API allows a user to change the state of an agent. Agents can change their own states. If the request to change an agent's state is successful, the response is sent as part of a User notification.
- Response:
  - 200: Success
  - 400: Bad Request
  - 401: Invalid Supervisor
  - 401: Unauthorized
  - 404: Not Found
  - 500: Internal Server Error
  - 503: Service Unavailable
- Example Failure Response:

```
{
status: 400,
statusText: Bad Request
details: [{
errorData: state,
errorType: Parameter Missing,
errorMessage: State Parameter missing
}]
}
```

- Trigger Event: User notification.

User—Sign Out of Finesse

```
signOut(onSuccess: Function, onError: Function)
```

- Request Parameters:
  1. onSuccess(data): (Optional) Function callback to be invoked if request succeeds
  2. onError(error): (Optional) Function callback to be invoked if request fails. (Refer to the section Call Control API Error Format.)

- API Description: This API allows a user to sign out of Finesse.
- HTTP Response:
  - 202: Success
  - 400: Bad Request (for example, malformed or incomplete request, invalid extension)
  - 401: Unauthorized (for example, the user is not authenticated in the Web Session)
  - 404: Not Found (for example, the user ID is not known)
  - 503: Service Unavailable (for example, the Notification Service is not running)
- Example Failure Response:

```
{
status: 400,
statusText: Bad Request
details: [{
32
errorData: state,
errorType: Invalid Input,
errorMessage: Invalid State specified for user
}]
}
```

- Trigger Event: User notification.

User—Load User

```
load(onSuccess: Function, onError: Function)
```

- Request Parameters:
  1. onSuccess(data): (Optional) Function callback to be invoked if request succeeds
  2. onError(error): (Optional) Function callback to be invoked if request fails. (Refer to the section Call Control API Error Format.)
- API Description: The load user API allows a user to get a copy of the User object.
- HTTP Response:
  - 200: Success
  - 401: Authorization Failure
  - 401: Invalid Authorization User Specified
  - 404: User Not Found
  - 500: Internal Server Error
  - 503: Service Unavailable
- Example Failure Response:

```
{
status: 404,
statusText: Not Found
details: [{
errorData: 4023,
errorType: User Not Found,
errorMessage: UNKNOWN_USER
}]
}
```

- Trigger Event: User notification.

Dialog—Create a New Dialog (Make a Call)

```
makeCall(dialledNumber:String, onSuccess:Function,onError:Function)
```

- Request Parameters:
  1. 1. dialledNumber:(required): the destination for the call
  2. 2. onSucess(data): (Optional) Function callback to be invoked if request succeeds
  3. 3. onError(error): (Optional) Function callback to be invoked if request fails. (Refer to the section Call Control API Error Format.)
- API Description: The Dialog object represents a dialog with participants. This API allows a user to make a call. The agent must be in NOT_READY state.
- HTTP Response:
  - 202: Successfully Accepted

    Note This response only indicates successful completion of the request. The request is processed and the actual response is sent as part of a dialog notification. This can be handled via the onDialogError callback.

  - 400: Bad Request (the request body is invalid)
  - 400: Parameter Missing
  - 400: Invalid Input
  - 400: Invalid Destination (the toAddress and fromAddress are the same)
  - 401: Authorization Failure
  - 401: Invalid Authorization
  - 500: Internal Server Error
- Example Failure Response:

```
{
status: 401,
statusText: Authorization Failure
details: [{
errorData: jsmith,
errorType: Authorization Failure,
errorMessage: Unauthorized
}]
}
```

- Trigger Event: Dialog notification.

Dialog—Answer an Incoming Call

```
answerCall(callId:String, onSuccess:Function,onError:Function)
```

- Request Parameters:
  1. callId: (Required) The ID of the dialog

2. onSuccess(data): (Optional) Function callback to be invoked if request succeed.
3. onError(error): (Optional) Function callback to be invoked if request fails. (Refer to the section Call Control API Error Format.)

- API Description: This API allows an agent to answer an incoming call.
- HTTP Response:
  - 202: Successfully Accepted
  - 400: Parameter Missing
  - 400: Invalid Input
  - 401: Authorization Failure
  - 401: Invalid Authorization User Specified
  - 404: Dialog Not Found
  - 500: Internal Server Error
- Example Failure Response:

```
{
status: 400,
statusText: Invalid Input
details: [{
errorData: ANSWER
errorType: Invalid Input,
errorMessage: Invalid action specified for dialog
}]
}
```

- Trigger Events:
  1. User notification
  2. Dialog notification
  3. Error notification

Dialog—Place a Call On Hold

```
holdCall(callId:String, targetAddress: String, onSuccess:Function,onError:Function)
```

- Request Parameters:
  1. callId: (Required) The ID of the dialog
  2. targetAddress: (Required) The extension of the agent placing the call on hold
  3. onSuccess(data): (Optional) Function callback to be invoked if request succeed.
  4. onError(error): (Optional) Function callback to be invoked if request fails. (Refer to the section Call Control API Error Format.)
- API Description: This API allows an agent to place a call on hold.
- HTTP Response:
  - 202: Successfully Accepted
  - 400: Parameter Missing
  - 400: Invalid Input
  - 401: Authorization Failure

  - 401: Invalid Authorization User Specified
  - 500: Internal Server Error
- Example Failure Response:

```
{
status: 400,
statusText: Invalid Input
details: [{
errorData: ANSWER,
errorType: Invalid Input,
errorMessage: Invalid action specified for dialog
}]
}
```

- Trigger Events:
  1. User notification
  2. Dialog notification
  3. Error notification

Dialog—Retrieve a Call

```
retrieveCall(callId:String, onSuccess:Function,onError:Function)
```

- Request Parameters:
  1. callId: (Required) The ID of the dialog
  2. onSuccess(data): (Optional) Function callback to be invoked if request succeed.
  3. onError(error): (Optional) Function callback to be invoked if request fails. (Refer to the section Call Control API Error Format.)
- API Description: This API allows the agent to retrieve a call placed on hold.
- HTTP Response:
  - 202: Successfully Accepted
  - 400: Parameter Missing
  - 400: Invalid Input
  - 401: Authorization Failure
  - 401: Invalid Authorization User Specified
  - 500: Internal Server Error
- Example Failure Response:

```
{
status: 400,
statusText: Invalid Input
details: [{
errorData: RETRIVE,
errorType: Invalid Input,
errorMessage: Invalid action specified for dialog
}]
}
```

- Trigger Events:
  1. User notification

2. Dialog notification
3. Error notification

Dialog—Make a Consult Call

```
consult(callId:String, dialedNumber:String, onSuccess:Function,onError:Function)
```

- Request Parameters:
  1. callId: (Required) The ID of the dialog
  2. dialedNumber: (Required) The destination for the call
  3. onSuccess(data): (Optional) Function callback to be invoked if request succeed.
  4. onError(error): (Optional) Function callback to be invoked if request fails. (Refer to the section Call Control API Error Format.)
- API Description: This API allows an agent to make a consult call request. After the request succeeds, the agent can complete the call as a conference or transfer. (The request is sent to the Dialog URL of an existing active call, from where the call is initiated.)

  Finesse supports the transfer or conference of any held call to the current active call, as long as the agent performing the transfer or conference is a participant in both the held and active call.
- HTTP Response:
  - 202: Successfully Accepted
  - 400: Parameter Missing
  - 400: Invalid Input
  - 401: Authorization Failure
  - 401: Invalid Authorization User Specified
  - 500: Internal Server Error
- Example Failure Response:

```
{
status: 400,
statusText: Invalid Input
details: [{
errorData: CONSULT_CALL,
errorType: Invalid Input,
errorMessage: Invalid action specified for dialog
}]
}
```

- Trigger Events:
  1. Dialog notification
  2. Error notification

Dialog—Conference a Consult Call

```
conference(callId:String, targetAddress:String, onSuccess:Function,onError:Function)
```

- Request Parameters:
  1. callId: (Required) The ID of the dialog

2. targetAddress: (Required) The extension of the agent who made the CONSULT_CALL
3. onSuccess(data): (Optional) Function callback to be invoked if request succeed.
4. onError(error): (Optional) Function callback to be invoked if request fails. (Refer to the section Call Control API Error Format.)

- API Description: This API allows an agent to conference a consult call into the user's active call. After a user makes a successful request, that user's consult call is removed and merged into the active call.
- Response:
  - 202: Successfully Accepted
  - 400: Parameter Missing
  - 400: Invalid Input
  - 401: Authorization Failure
  - 401: Invalid Authorization User Specified
  - 500: Internal Server Error
- Example Failure Response:

```
{
status: 400,
statusText: Invalid Input
details: [{
errorData: CONFERENCE,
errorType: Invalid Input,
errorMessage: Invalid action specified for dialog
}]
}
```

- Trigger Event:
- 1. Dialog notification
- 2. Error notification

Dialog—Transfer a Consult Call

```
transferCall(callId:String, targetAddress:String, onSuccess:Function,onError:Function)
```

- Request Parameters:
  1. callId: (Required) The ID of the dialog
  2. targetAddress: (Required) The extension of the agent who made the CONSULT_CALL
  3. onSuccess(data): (Optional) Function callback to be invoked if request succeed.
  4. onError(error): (Optional) Function callback to be invoked if request fails. (Refer to the section Call Control API Error Format.)
- API Description: This API allows a user to transfer a consult call. After a user makes a successful request, that user's active call is transferred to the destination.
- Response:
  - 202: Successfully Accepted
  - 400: Parameter Missing

  - 400: Invalid Input
  - 401: Authorization Failure
  - 401: Invalid Authorization User Specified
  - 500: Internal Server Error
- Example Failure Response:

```
{
status: 400,
statusText: Invalid Input
details: [{
errorData: TRANSFER,
errorType: Invalid Input,
errorMessage: Invalid action specified for dialog
}]
}
```

- Trigger Event:
  1. User notification
  2. Dialog notification
  3. Error notification

Dialog—Drop a Call

```
dropCall(callId:String, targetAddress:String, onSuccess:Function,onError:Function)
```

- Request Parameters:
  1. callId: (Required) The ID of the dialog
  2. targetAddress: (Required) The extension of the agent dropping the call
  3. onSuccess(data): (Optional) Function callback to be invoked if request succeed.
  4. onError(error): (Optional) Function callback to be invoked if request fails. (Refer to the section Call Control API Error Format.)
- API Description: This API allows the agent to drop (disconnect) a call.
- Response:
  - 202: Successfully Accepted
  - 400: Parameter Missing
  - 400: Invalid Input
  - 401: Authorization Failure
  - 401: Invalid Authorization User Specified
  - 500: Internal Server Error
- Example Failure Response:

```
{
status: 400,
statusText: Invalid Input
details: [{
41
errorData: DROP,
errorType: Invalid Input,
errorMessage: Invalid action specified for dialog
```

```
}]
}
```

- Trigger Event:
  1. User notification
  2. Dialog notification
  3. Error notification

Call Control API Error Format

All error arguments provided to the onError callback by the Call Control API have the format described below:

```
{
status: <http status>
statusText: <http status text>
details: [{
errorData: [Error Code],
errorType:[Error Category],
errorMessage: [Error Constant]
}]
}
```

User API Errors

Table 23 User API Errors

Status	Error Type	Description
400	Bad Request	The request is malformed or incomplete or the extension provided is invalid.
400	Generic Error	An unaccounted for error occurred. The root cause could not be determined.
400	Invalid Input	One of the parameters provided as part of the user input is invalid or not recognized (for example, the state for a user)
400	Invalid State	The requested state change is not allowed (for example, a user in LOGOUT state requests a state change to LOGOUT or a user tries move to LOGOUT state from READY state without going to NOT_READY state).
400	Parameter Missing	The extension, state, or requestedAction is not provided
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API.
401	Invalid Authorization User Specified	The authenticated user tried to make a request for another user
401	Invalid State	A user tried to change to a state that is not supported in the scenario
404	Not Found	The resource specified is invalid or does not exist
404	User Not Found	The user ID provided is invalid or is not recognized. No such user exists in CTI.

Status	Error Type	Description
500	Internal Server Error	Any runtime exception is caught and responded with this error.
503	Service Unavailable	A dependent service is down (for example, the Cisco Finesse Notification Service or Cisco Finesse Database). Finesse is OUT_OF_SERVICE.

Dialog API Errors

Table 24 Dialog API Errors

Status	Error Type	Description
400	Generic Error	An unaccounted for error occurred. The root cause could not be determined.
400	Invalid Destination	The toAddress and fromAddress are the same (if users attempt to call their own extension).
400	Invalid Input	One of the parameters provided as part of the user input is invalid or not recognized (for example, the fromAddress, toAddress, targetMediaAddress, requestedAction)
400	Parameter Missing	A required parameter was not provided in the request. For example, if creating a dialog, the fromAddess or toAddress was not provided.
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API.
401	Invalid Authorization User Specified	The authenticated user tried to make a request for another user . The authenticated user tried to use a fromAddress that does not belong to that user.
404	Not Found	The resource specified is invalid or does not exist
404	Dialog Not Found	The dialogId provided is invalid or no such dialog exists.
500	Internal Server Error	Any runtime exception is caught and responded with this error.

User and Dialog Notification Subscription

A user can subscribe to events to be notified of changes to agent state and dialog information. The topics that are available for subscription are described below:

Table 25 Subscriptions

Callback handler	Scenario	Notification Category
READ.CallControl.Topic.USER_STATE_LOGIN	This is published as the first event before the user transitions to any other state except LOGOUT	User
READ.CallControl.Topic.USER_STATE_LOGOUT	Will be published when the agent moves to LOGOUT state	User

Callback handler	Scenario	Notification Category
READ.CallControl.Topic.USER_STATE_READY	Will be published when the agent changes to READY state	User
READ.CallControl.Topic.USER_STATE_NOTREADY	Will be published when the agent changes to NOT_READY state	User
READ.CallControl.Topic.USER_STATE_RESERVED	Will be published when the agent is being alerted of a call. The internal state of the user is RESERVED.	User
READ.CallControl.Topic.USER_STATE_TALKING	Will be published when the agent answers the call and moves to TALKING state	User
READ.CallControl.Topic.USER_STATE_HOLD	Will be published when the call is placed on hold by the user	User
READ.CallControl.Topic.DIALOG_ADDED	Will be published a new dialog is created. The newly created dialog object is provided as a parameter to the callback.	Dialog
READ.CallControl.Topic.DIALOG_DELETED	Will be published when a dialog is removed Dialog from a user. The removed dialog is provided as a parameter to the callback.	Dialog
READ.CallControl.Topic.DIALOG_MODIFIED	Will be published when the state of the dialog or the state of any one of the participants change. The modified dialog object is provided as a parameter to the callback.	Dialog
READ.CallControl.Topic.DIALOGS_LOADED	This callback is useful to reconstruct the state of the page if the page is refreshed. It will be published when the dialogs are loaded during a page refresh. The argument is the array of dialogs created for the user.	Dialog

Callback handler	Scenario	Notification Category
READ.CallControl.Topic.CALLCONTROL_ERROR	This will be published when any call operation result in an error. The parameter is an array of error objects. See the section on Error Notification for more information.	Dialog
READ.CallControl.Topic.SERVER_CONNECT_ERROR	This will be published whenever BOSH connection to Finesse server fails. For example due to invalid user name or password.	
READ.CallControl.Topic.SERVER_DISCONNECT	This is published when the BOSH connection drops inadvertently.	

User and Dialog Error Notification Format

Call operations performed on a dialog may result in errors. These errors are sent as asynchronous updates. The parameter provided to the onDialogError callback method is an array of error objects with the following structure:

```
[{
errorData: [Error Code],
errorMessage: [Error Constant],
errorType:[Error Category]
}]
```

Dialog State Parameter Values

The following table describes possible values for the dialog parameter.

Table 26 Dialog State Parameter Values

Dialog State	Description
INITIATING	Indicates that the phone is off the hook at a device
INITIATED	Indicates that the phone is dialing at the device
ALERTING	Indicates that the call is ringing at a device
ACTIVE	Indicates that the dialog has at least one active participant
FAILED	Indicates that the dialog has failed
DROPPED	indicates that the dialog has no active participants

Participant State Parameter Values

The following table describes possible values for the participant of a dialog.

Table 27 Participant State Parameter Values

Participant State	Allowable Actions	Description
INITIATING	DROP	Indicates that an outgoing call, not yet active, exists on the device
INITIATED	DROP	Indicates that the phone is dialing at a device
ALERTING	ANSWER	Indicates that an incoming call is ringing on the device
ACTIVE	HOLD, DROP, CONSULT_CALL	Indicates that the participant is active on the call
FAILED	DROP	Indicates that the call failed (BUSY, BAD_DESTINATION, OTHER)

Integration of Call Control APIs with READ SDK

- Construct the user object by providing the username, password, extension and domain in the constructor configuration

```
var _user = new User({
username: 'agent23,
password: 'secret',
extension: '1023',
domain: 'www.myfinesse.com'
});
```

  These can also be changed dynamically at a later point of time with the 'reset' method.

- Subscribe listener methods to receive User and Dialog event notifications

```
onLogin = function(userevent) {
// Initialize UI state
};
READ.Event.subscribe(READ.CallControl.Topic.USER_STATE_LOGIN, onLogin);
onLogout = function(userevent) {
// Reset UI state
};
READ.Event.subscribe(READ.CallControl.Topic.USER_STATE_LOGOUT, onLogout);
onNotReady= function(userevent) {
// handle UI state
};
READ.Event.subscribe(READ.CallControl.Topic.USER_STATE_NOTREADY, onNotReady);
onReady = function(userevent) {
// handle UI state
};
READ.Event.subscribe(READ.CallControl.Topic.USER_STATE_READY, onReady);
```

Tip User events notifications such as READ.CallControl.Topic.USER_STATE_RESERVED, READ.CallControl.Topic.USER_STATE_TALKING, and READ.CallControl.Topic.USER_STATE_HOLD can be useful to handle any UI behavior specific to those states.

```
function onDialogAdded (dialog ) {
/*
* Dialogs will be created in these circumstances
* - The agent makes a call
* - Agent receives an inbound call from a kiosk
* - A Consult call is made or comes in
* - An inbound Consult call is transferred
* This is the right place to initialize READ SDK
* READ.Config.init({…});
*/
}
READ.Event.subscribe(READ.CallControl.Topic.DIALOG_ADDED, onDialogAdded);
function onDialogDeleted ( dialog ) {
/*
* Dialogs will be deleted in these circumstances
* - A Consult call is conferenced
* - A Consult call is transferred
*/
}
READ.Event.subscribe(READ.CallControl.Topic.DIALOG_DELETED, onDialogDeleted);
function onDialogModified (dialog ) {
/*
* Dialogs will be modified when the state of a call participant changes.
* Few such examples includes
* - The participant answers a call
* - The participant places a call on hold or retrieves a call
* - A participant drops off a conference
* - When a remote participant is conference, the call type changes to
From PREROUTE_ACD_IN to CONFERENCE and back to
PREROUTE_ACD_IN when the remote participant drops off the
conference
* - When a CONSULT call is TRANSFERred to another participant the
dialog call type of the transferred agent changes from TRANSFER to
CONSULT_OFFERED
*/
}
READ.Event.subscribe(READ.CallControl.Topic.DIALOG_MODIFIED, onDialogModified);
function onDialogsLoaded ( dialogs ) {
/*
* Implement this callback if page refresh is to be supported
* The argument provided to this callback is the array of dialogs
* created for this user. Its implementation is for most part
* a combination of the functionality of onDialogAdded and
* onDialogModified.
*/
}
READ.Event.subscribe(READ.CallControl.Topic.DIALOGS_LOADED, onDialogsLoaded);
function onCallControlError ( errors ) {
/*
* The argument is an array of error objects.
* Refer to the section on 'User and Dialog Error Notification Format'
*/
}
READ.Event.subscribe(READ.CallControl.Topic.CALLCONTROL_ERROR, onCallControlError);
```

- Handle any authentication errors during Sign-In

```
// To handle authentication errors and any inadvertent disconnection of the client
from
// Finesse BOSH notification server.
READ.Event.subscribe(READ.CallControl.Topic.SERVER_CONNECT_ERROR, function(error) {
alert('Failed to connect to finesse. Reason = ' + error);
});
```

- Handle any inadvertent disconnect of client from Finesse BOSH notification server

```
READ.Event.subscribe(READ.CallControl.Topic.FINESSE_DISCONNECT, onLogout);
```

- Load user and subsequently load associated dialogs for the user

```
_user.load(function() {
_user.loadDialogs(function(data){
_console.print2Console("LOAD DIALOGS RESPONSE ", data);
}, function(error) {
_console.print2Console("LOAD DIALOGS ERROR ", error);
});
}, function(error) {
console.error('User load error >>', error);
});
```

To handle authentication errors and any inadvertent disconnection of the client from Finesse BOSH notification server.

```
// Possible authentication error during sign-in
READ.Event.subscribe(READ.CallControl.Topic.FINESSE_CONNECT_ERROR, function(error) {
alert('Failed to connect to finesse. Reason = ' + error);
});
// Accidental disconnect of finesse
READ.Event.subscribe(READ.CallControl.Topic.FINESSE_DISCONNECT, onLogout);
```

- Subscribe to the Co-browsing and MBR events

```
READ.Event.subscribe(READ.Session.Topic.REM_AGENT_SESSION, function(isActive) {
// Toggle UI state based on agent session
});
READ.Event.subscribe(READ.Vnc.Topic.VNC_EMPTY, function() {
// Initialize or reset buttons if there are no co-browsing jobs
});
READ.Event.subscribe(READ.Vnc.Topic.VNC_INITED, function(job) {
// Handle UI state when the co-browsing job is accepted at REM server
});
READ.Event.subscribe(READ.Vnc.Topic.VNC_COMPLETED, function(job) {
// Handle UI state when the co-browsing job is started at REM server.
// It is here that applet element for the VNC client is created and loaded
});
READ.Event.subscribe(READ.Vnc.Topic.VNC_ACKNOWLEDGED, function(job) {
// Handle UI state when the co-browsing job is stopped at REM server
});
READ.Event.subscribe(READ.Session.Topic.REM_CALL_ONHOLD, function(isOnHold) {
// Handle UI state when if the call is placed on hold or retrieved
});
READ.Event.subscribe(READ.MBR.Topic.VTA_PARAM_AVAILABLE, function(querystring) {
// this event is published when the query string parameters for
// bringing up VTA are available. The query string provided to the listener
// callback will be in the form of ?param1=value1&param2=value2. For example
// ?id=<Chinese National ID>&name=<Customer Name>.
});
```

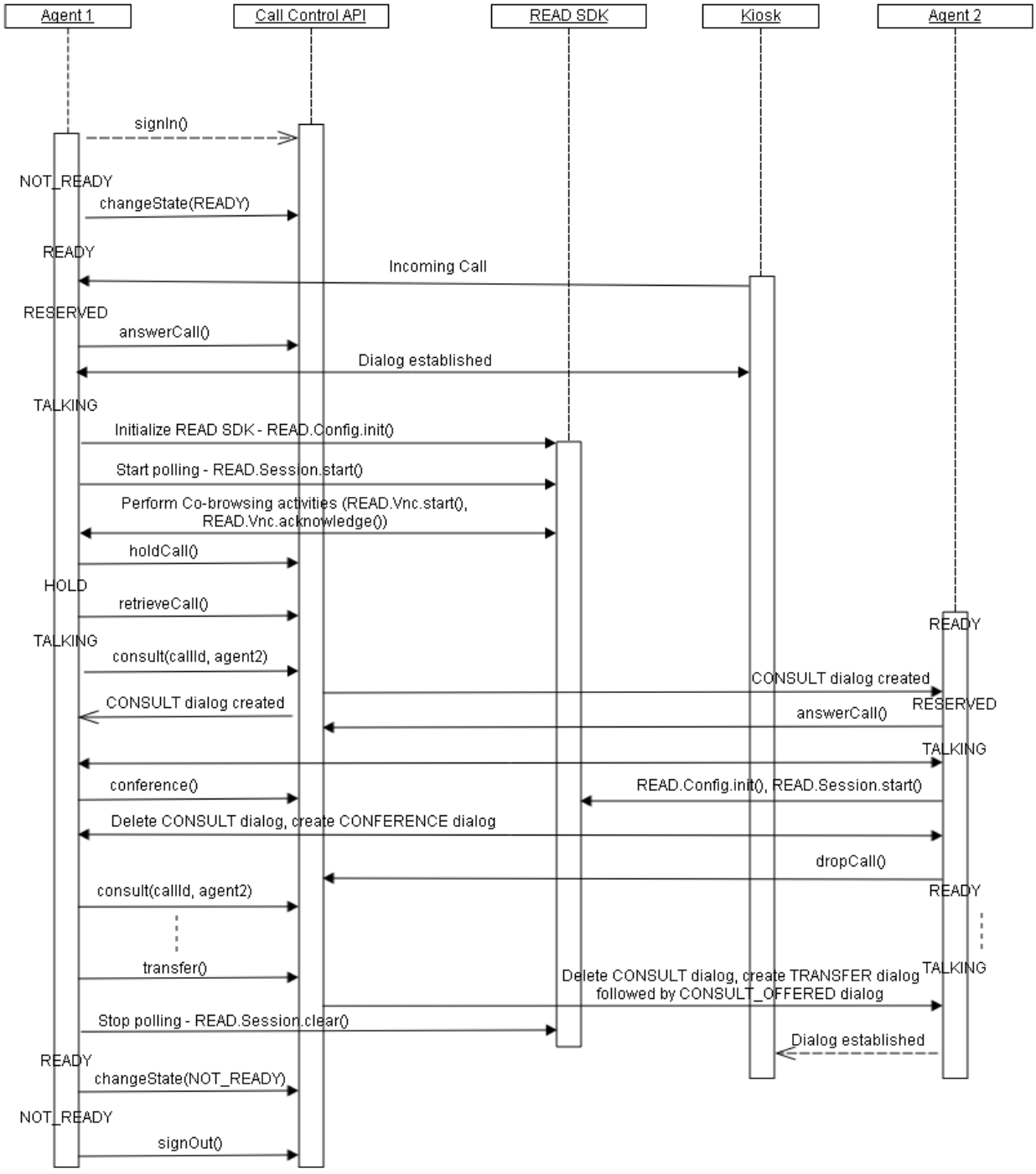
Sequence Diagrams for Components

API in Possible Use Case Scenarios

The APIs support the following capabilities:

- User Sign In/Sign Out
- Change Agent States
- Call Control
- Event Notification (call back handler to clients that subscribe to that class of resource)

Figure 4 Detailed Communication of Call Control Scenario



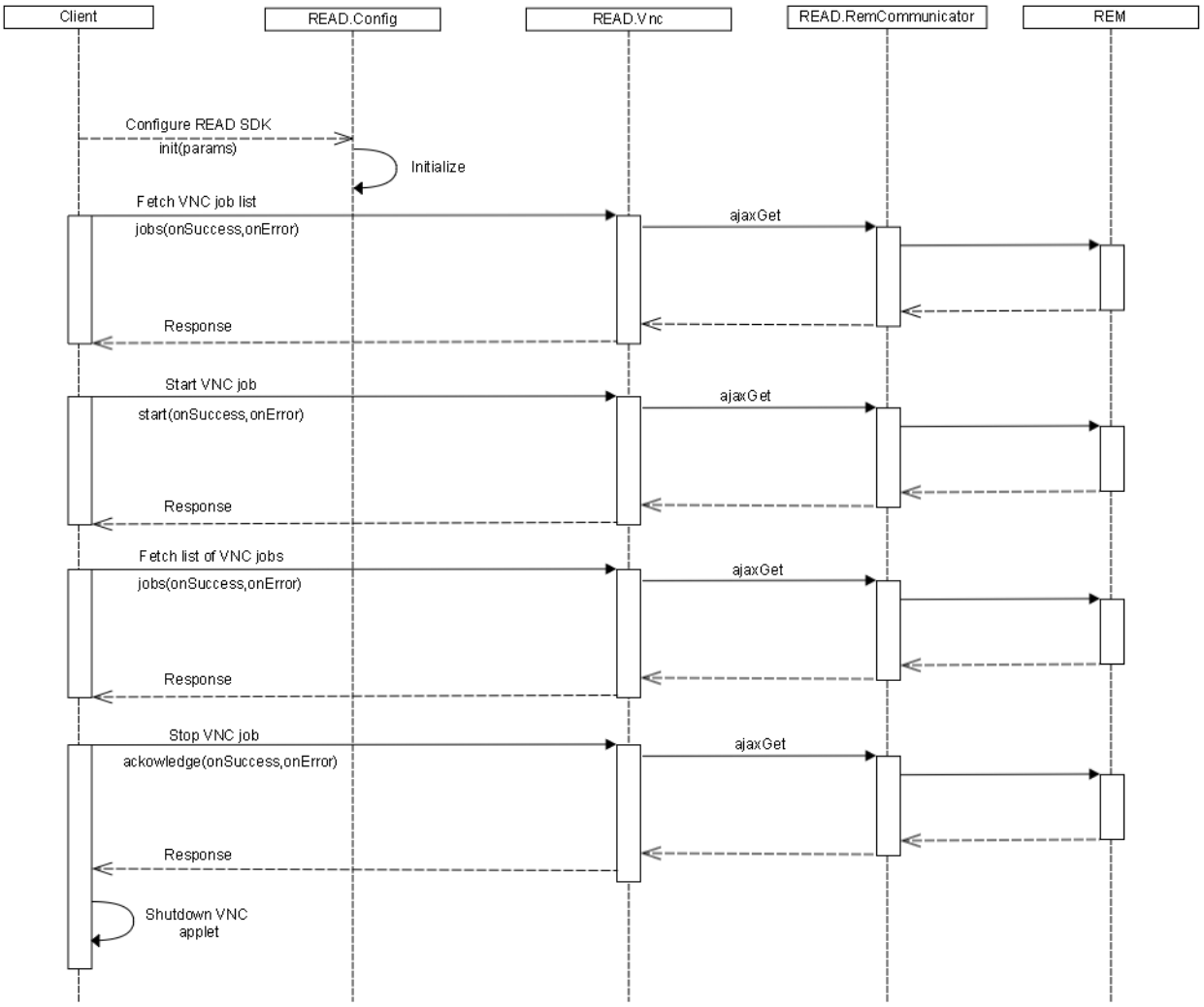
Supported use:

- Create a call (dialog) between VTM and VTA (agent)
- Put active call on hold
- Consult another remote agent and followed by conference or transfer the call

VNC Module

This section contains detail communication diagrams of VNC module.

Figure 5 Detailed Communication of VNC Module



Example Usage of READ SDK

This section provides some sample codes for using READ SDK in client application.

READ SDK Recommended Practices

- Implement the onError callback while configuring READ SDK through READ.Config.init() method. This will give an opportunity to handle accidental drop of REM session or HTTP connection:

```
READ.Config.init({
onError: function(error) {
    switch(error.status) {
    case 0:
    case 12029:
        console.error('Communication Failure.');
        break;
    case 420:
        var details = error.details;
        var msg = details.message;
        var code = details.code;
        console.error('Server reported an error. ', code, msg);
        if(code === READ.Constants.ErrorCode.NO_REM_SESSION) {
        READ.Session.clear();
        }
        break;
}
}.
});
```

- Take care to verify the correct URL is provided as codebase property while creating the APPLET element through the createAppletSource() method of VNC class.
- Stop Co-browsing job from the UI rather than disconnecting from VNC client.

Reference Implementation of VNC feature

- Define a placeholder element for the VNC co-browsing client applet:

```
<div id="appletplace"> </div>
```

- Start the Job at REM Server:

```
READ.Vnc.start(function(data) {
console.info('Vnc request sent successfully');
}, function(err, response) {
alert('Failed to start VNC. Status: ' + error.status + ', Cause: ' +
error.statusText);
});
```

- Poll for Job Status to Handle State of Application:

```
READ.Event.subscribe(READ.Session.Topic.REM_AGENT_SESSION, function(isActive) {
if(!isActive) {
if(READ.Vnc.isActive()) {
READ.Vnc.acknowledge();
}
document.getElementById('appletplace').innerHTML = "";
disableStartVncBtn();
disableStopVncBtn();
}
});
READ.Event.subscribe(READ.Vnc.Topic.VNC_EMPTY, function() {
enableStartVncBtn();
disableStopVncBtn();
```

```
});
READ.Event.subscribe(READ.Vnc.Topic.VNC_INITED, function(job) {
disableStartVncBtn();
if(job.Owner === READ.Config.agent) {
enableStopVncBtn();
} else {
disableStopVncBtn();
}
});
READ.Event.subscribe(READ.Vnc.Topic.VNC_COMPLETED, function(job) {
var appletPlace = document.getElementById('appletplace').innerHTML;
if(job.Owner === READ.Config.agent) {
disableStartVncBtn();
enableStopVncBtn();
// If applet is not already loaded, create an invisible
// APPLET by setting the width and height as 1px.
if(!appletPlace) {
var appletsource = READ.Vnc.createAppletSource( {width:1, height:1}, {codebase:
window.location.origin +'/FinesseAPI/readsdk', iecIp: job.IecIp,
iecPassword:job.IecPasswd, vncServerPort:job.VncServerPort } );
document.getElementById('appletplace').innerHTML = appletsource;
}
} else if(job.Owner !== READ.Config.agent) { // Probably a conference call
document.getElementById('appletplace').innerHTML = "";
disableStartVncBtn();
disableStopVncBtn();
}
});
READ.Event.subscribe(READ.Vnc.Topic.VNC_ACKNOWLEDGED, function(job) {
document.getElementById('appletplace').innerHTML = "";
enableStartVncBtn();
disableStopVncBtn();
});
```

- Acknowledge to Terminate the Job at REM:

```
READ.Vnc.acknowledge(function() {
document.getElementById('appletplace').innerHTML = "";
}, function(error) {
alert("Co-browsing stop failed. Reason: " + error.statusText);
});
```