Cisco Remote Expert Agent Desktop SDK Developer Guide

Release 1.9.4

March 5, 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:

Introduction

READ SDK contains the control logic of READ for Video and VNC modules, which segregates controller code from the client user interface (UI) code. Previously, control logic code was integrated with the client UI code written in ExtJS. With the help of this SDK client, UI code can be independent from the core control logic. As a result, the SDK can be integrated with client applications easily without the need of Flash.

This document describes the high-level design diagrams and application programming interface (API) details for each operation performed in READ.

Acronyms

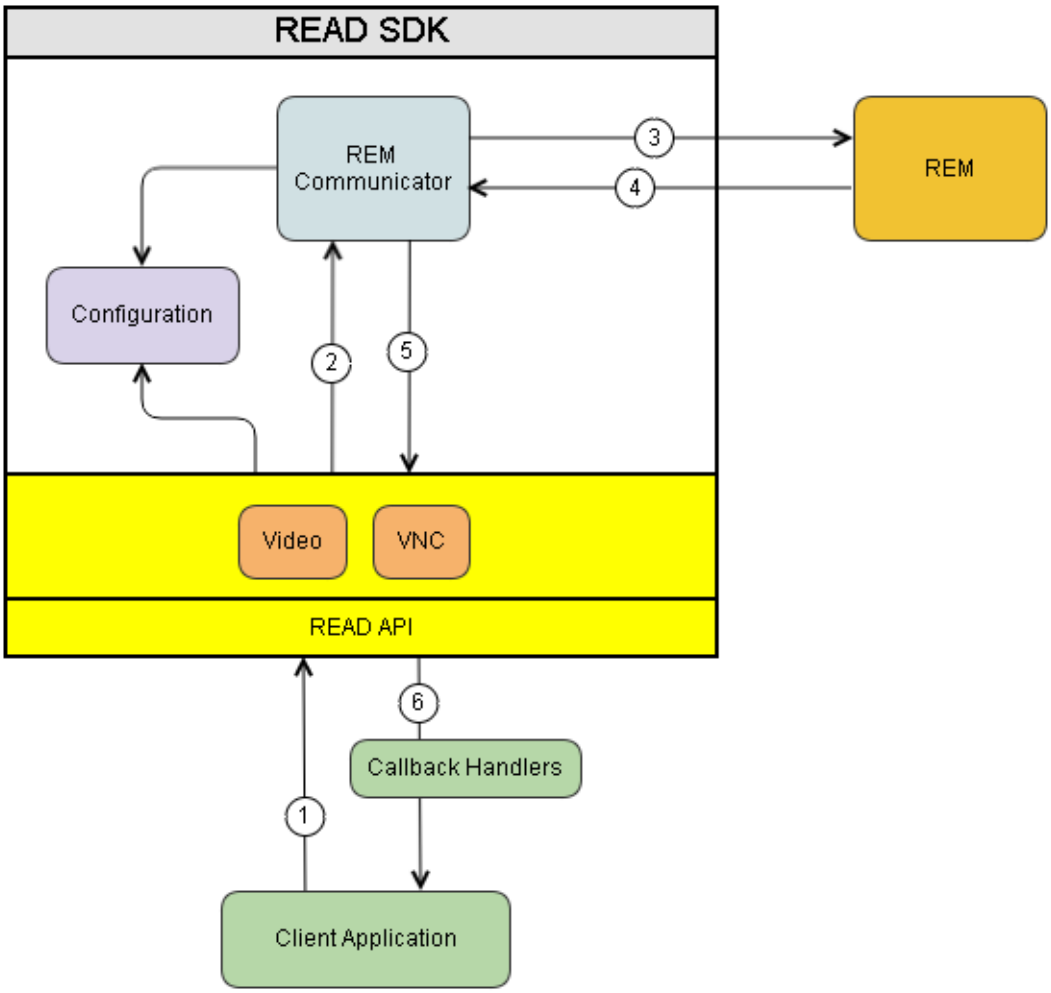
The following acronyms are used in this guide:

- API – Application Programming Interface
- eREAD – eRemote Expert Agent Desktop (uses Cisco Finesse)
- IEC – Interactive Experience Client
- 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

High Level Component Interaction

The figure below illustrates the internal communication between 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., getVideoList(), streamVideo(), stopVideo(), 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 (e.g., video list).
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

The main class for accessing all the APIs is the ‘READ’.

READ APIs are grouped into Configuration, Video, and VNC modules. The Configuration API is used to configure READ SDK and the AJAX utility API within it is used to make AJAX calls to the REM server. The Video and VNC modules are used to manage Video and VNC jobs respectively.

1. READ Configuration module – The 'READ' class has a 'Configuration' property, which is an object containing the `init` method. The signature of the method is shown below:

   ```
   READ.Configuration.init (/*object*/ params)
   ```

   - `agent` – [mandatory] Agent's directory number
   - `kiosk` – [optional] Kiosk's or caller's directory number
   - `hostname` – [optional] Hostname or IP address of the REM server. If a value is not provided, the default value will be the current host on the browser.
   - `port` – [optional] Port of the REM server. If a value is not provided, the default value will be the current port of the browser.
   - `protocol` – [optional] Scheme of the REM server, whether it be HTTP or HTTPS. If a value is not provided, it will be inferred by the port value.
   - `onAjaxStart()` – [optional] Callback to be invoked when an Ajax call is made. The callback can bring up a loading mask or any useful message to the user.
   - `onAjaxComplete()` – [optional] Callback to be invoked when the Ajax call completes.
   - `onAjaxError(error)` – [optional] Callback to be invoked when an error occurs during the Ajax call. The error parameter contains responseJSON, responseText, status, and statusText.

2. READ VNC module – The VNC module is available under the 'READ.Vnc' namespace. The 'Vnc' class has the following methods:

   a. `start(onSuccess, onError)` – This sends a request to create the VNC job on the REM server.
      - `onSuccess(data)` – This is the callback that will be invoked when the job is successfully created by the REM server.
      - `onError(error)` – This is the callback that will be invoked during an error.

   b. `acknowledge(onSuccess, onError)` – This is sent to indicate to the REM server to stop the job and to clear the command queue.
      - `onSuccess(data)` – This is the callback that will be invoked when the request is successfully executed.
      - `onError(error)` – This is the callback that will be invoked during an error.

   c. `jobs(onSuccess, onError)` – This fetches the list of VNC jobs submitted to REM in the current session. The response of this request can be used to determine the current state of the job and handle the UI elements accordingly.
      - `onSuccess(data)` – This is the callback that will be invoked when the request for jobs is successfully received by the server.
      - `onError(error)` – This is the callback that will be invoked during an error.

   This method must be invoked in a polling cycle to fetch the status of a VNC job periodically. The jobs method emits various events to which the client has to subscribe if it is interested in handling them. The events emitted are VNC_INITED, VNC_COMPLETED, VNC_ACKNOWLEDGED, and VNC_EMPTY.

3. READ Video module – The video module is available under the 'READ.Video' namespace. The 'Video' class has the following methods:

a. `list(onSuccess, onError)` – This fetches the list of videos configured by the administrator in REAC.
   - `onSuccess(data)`– This is the callback that will be invoked when the list of videos is successfully received by the method.
   - `onError(error)`– This is the callback that will be invoked during an error.

b. assisgnOnHold(id, onSuccess, onError) – This assigns a video to be played on the kiosk when the call is on hold during a session.
   - `id` – This is the identification of the video record to be assigned for on hold playback
   - `onSuccess(data)` – This is the callback that will be invoked when the command is successfully executed.
   - `onError(error)`– This is the callback that will be invoked during an error.

c. `stream(id, onSuccess, onError)`– This sends a request to queue a video to be played back on the kiosk to the customer.
   - `id` – This is the identification of the video record to be assigned.
   - `onSuccess(data)`– This is the callback that will be invoked when the command is successfully executed.
   - `onError(error)`– This is the callback that will be invoked during an error.

   The `stream()` function internally initiates polling to determine the states of video sharing jobs submitted and emits events for the client to handle. It continues polling as long as there is at-least one shared video in viewing state. If there are no pending jobs, it stops polling. The events published are: VIDEO_TO_BE_VIEWED, VIDEO_VIEWING, VIDEO_STOPPED, VIDEO_VIEWED, VIDEO_ACKNOWLEDGED, NO_VIDEO_JOBS.

d. `stop(onSuccess, onError)`– This sends a request to queue a video to be played back on the kiosk to the customer.
   - `id` – This is the identification of the video record to be assigned
   - `onSuccess(data)`– This is the callback that will be invoked when the command is successfully executed.
   - `onError(error)`– This is the callback that will be invoked during an error.

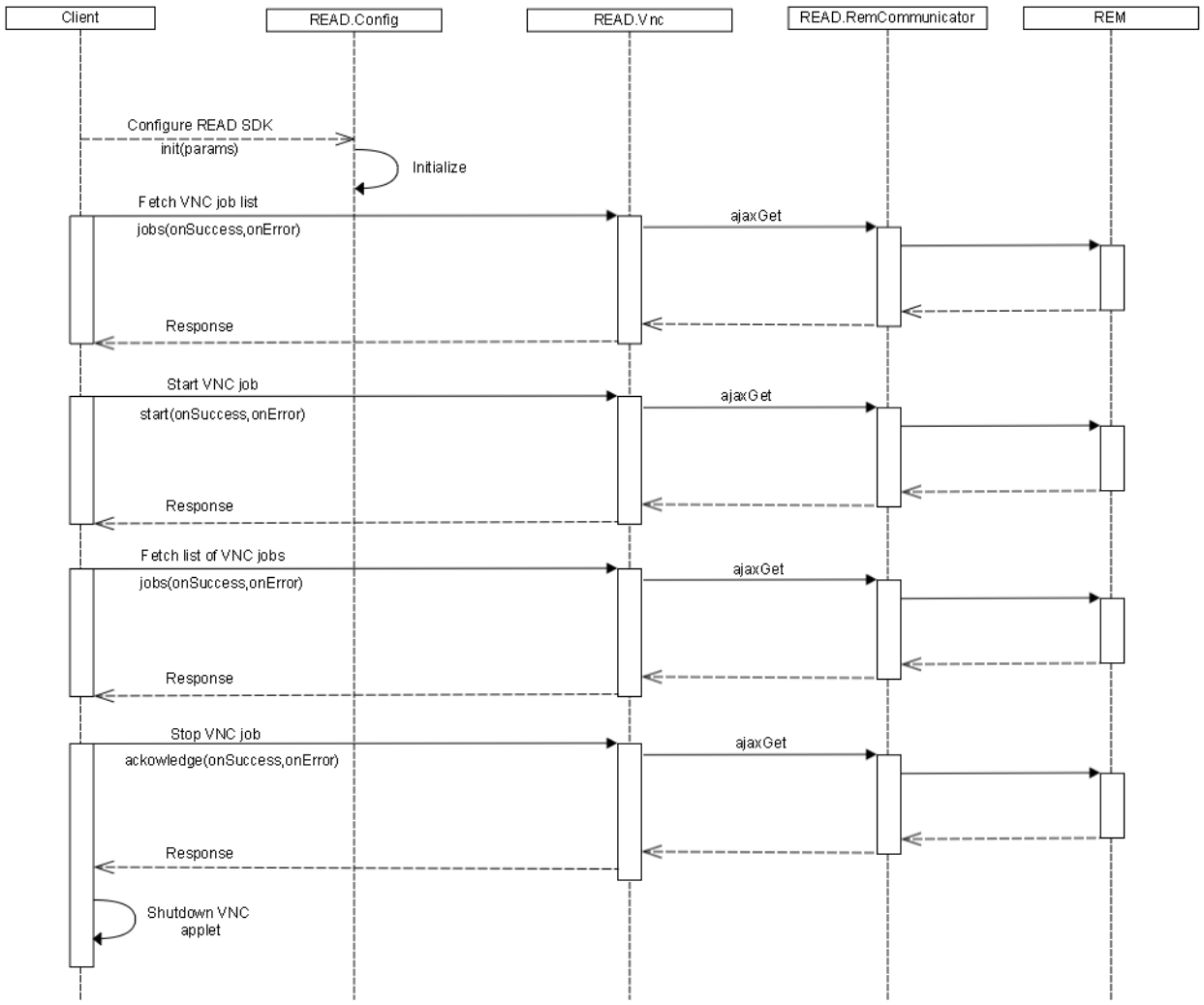
Sequence Diagrams for Components

This section contains detail communication diagrams of the following:

- VNC module
- Video module

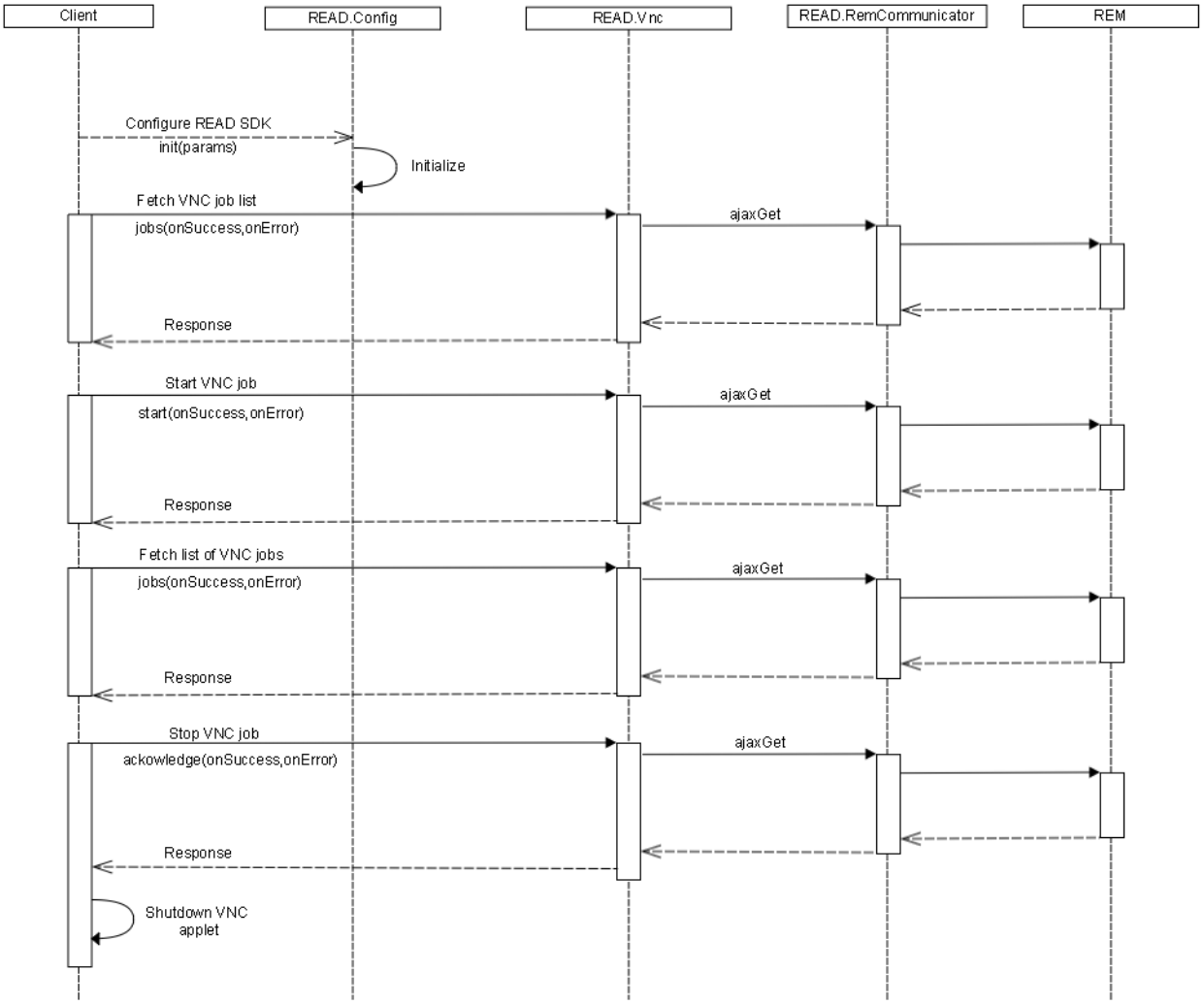
VNC Module

Figure 2 Detailed Communication of VNC Module



Video Module

Figure 3 Detailed Communication of Video Module



Error Details

The following table contains error codes for the READ APIs.

Table 1 Error Codes

Code	Message	Comment
101	System error. Please try again later.	Internal REM error.
110	Illegal argument is passed.	Sent if agent directory number is missing in the parameter list sent to REM.
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.
114	Job request is already processed.	
115	No active session was detected or available.	If a request is made without an active session in progress.
117	Command is invalid.	
118	Last command execution is not finished yet.	

Example Usage of READ SDK

Configure READ SDK

```
READ.Config.init({
    agent:'2519',
    kiosk: '2504',
    hostname: '172.25.26.125',
    port: '80',
    onAjaxStart: function() {
        console.log('XHR request sent. Please wait...');
    },
    onAjaxComplete: function() {
        console.log('XHR Completed');
    },
    onAjaxError: function(xhr) {
        console.error('XHR request failed :: ', xhr.statusText);
    }
});
```

Reference Implementation of VNC feature

Define the VNC Applet to Run

```
function startApplet(iecIp, iecPwd, vncPort) {
    appletsource='<applet code="com.glavsoft.viewer.Viewer"
        archive="tightvnc-jviewer.jar"'+'height=500 width=800>\n';
```

```
    appletsource+='<PARAM name="ipAddress" value="' + iecIp + '">\n';
    appletsource+='<PARAM name="password" value="' + iecPwd + '">\n';
    appletsource+='<PARAM name="port" value="' + vncPort  + '">\n';
    appletsource += '<embed type="application/x-java-applet;version=1.6"
        code="com.glavsoft.viewer.Viewer" archive="tightvnc-jviewer.jar" width="800"
        height="500" ipAddress="' + iecIp + '" password="' + iecPwd + '"></embed>';
    appletsource+='</applet>\n';

    document.getElementById('appletplace').innerHTML = appletsource;
}

<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: ' + err.status + ', Cause: ' + response.message);
});
```

Poll for Job Status to Handle State of Application

```
var vncpoll = setInterval(function() { fetchJobInfo(); }, 5000);

READ.Event.subscribe(READ.Event.Topic.VNC_INITED, function(job) {
    if(job && job.Status === READ.Constants.JobStatus.INITIATED) {
        disableStartVncBtn();
        enableStopVncBtn();

        if(!activeJob) activeJob = job;
    }
});

READ.Event.subscribe(READ.Event.Topic.VNC_COMPLETED, function(job) {
    if(job && job.Status === READ.Constants.JobStatus.COMPLETED) {
        enableStopVncBtn();
        disableStartVncBtn();

    if(!activeJob) activeJob = job;

    if(job.Owner === READ.Config.agent) {
        if(!jobInProgress) {
        jobInProgress = true;
        startApplet(job.IecIp, job.IecPasswd, job.VncServerPort || '5980');
        }
        } else if(job.Owner !== READ.Config.agent) {
        document.getElementById('appletplace').innerHTML = "";

        disableStartVncBtn();
        disableStopVncBtn();
        }
    }
});

READ.Event.subscribe(READ.Event.Topic.VNC_ACKNOWLEDGED, function(job) {
    if(job && job.Status === READ.Constants.JobStatus.ACKNOWLEDGED
        && activeJob && activeJob.JobId === job.JobId) {
    activeJob = null;
```

```
    jobInProgress = false;

    document.getElementById('appletplace').innerHTML = "";

    enableStartVncBtn();
    disableStopVncBtn();

});

READ.Event.subscribe(READ.Event.Topic.VNC_EMPTY, function() {
    activeJob = null;

    enableStartVncBtn();
    disableStopVncBtn();
});
```

Acknowledge to Terminate the Job at REM

```
READ.Vnc.acknowledge(function() {
    jobInProgress = false;
    document.getElementById('appletplace').innerHTML = "";
}, function(err, response) {
    alert('Job acknowledgement failed. Status: ' +err.status + ', Cause: ' +
    response.message);
});
```

Example Usage of Video APIs

Fetch List of Video and Display on the UI

```
READ.Video.list(function(data) {
    console.log(data);

    $.each(data, function(idx, video) {
        // Construct data table
    });
}, function(status, error) {
    console.info(status, error);
});
```

Assign a Video to be Played During Call On-Hold

```
READ.Video.assignOnHold(1, function(data) {
    console.info('video stop response::', data);
}, function(status, error) {
    console.error('video stop error::', status, error);
});
```

Subscribe to Video State Events

```
READ.Event.subscribe(READ.Event.Topic.VIDEO_TO_BE_VIEWED, function(job) {
    // Update UI state here
});
```

```
READ.Event.subscribe(READ.Event.Topic.VIDEO_VIEWING, function(job) {
    // Update UI state here
});

READ.Event.subscribe(READ.Event.Topic.VIDEO_STOPPED, function(job) {
    // Update UI state here
});

READ.Event.subscribe(READ.Event.Topic.VIDEO_VIEWED, function(job) {
    // Update UI state here
});

READ.Event.subscribe(READ.Event.Topic.VIDEO_ACKNOWLEDGED, function(job) {
    // Update UI state here
});
```

Stream a Video to the Kisok

```
$('#video-start').click(function() {
    READ.Video.stream(1, function() { // READ does not send any response
        // inform user
    }, function(status, error) {
        // inform user
    });
});
```

Stop Video Streaming

```
READ.Video.stop(1, function() { // READ does not send any response
    // inform user
}, function(status, error) {
    // inform user
});
```