Co-Browsing

Co-browsing is achieved via VNC sharing. The VNC server must be started in order to start co-browsing. To enable co-browsing, the client application needs to include Vnc.js in the HTML page.

The following are the APIs for co-browsing:

1. Start VNC server – The following API starts the VNC sever:

   `REIC.vnc.start(password, host, port, readOnly)`

   - `password` – IEC's device maintenance code as the password (optional)
   - `host` – IP address of the IEC device (optional)
   - `port` – Port for VNC server (by default 5980)
   - `readOnly` – Read only access ('true' or 'false')

   After starting the VNC server successfully, the VNC job must be updated with the 'Completed' status (2) using the following API:

   ```
   REIC.common.updateJobStatus(sessionId,REIC.common.jobStatus.COMPLE
   TED, REIC.common.jobType.VNC, '', '')
   ```

   - `sessionId` – Session identification
2. Stop VNC server – To stop the VNC server, invoke the following API:

   `REIC.vnc.stop()`
3. Check Server Status – Invoke the following API to check running status of the VNC server:

   `REIC.vnc.isRunning()`

   - return – `true/false`
4. Read Only Check – Whether the VNC server is accessed as read only can be checked by the following API:

   `REIC.vnc.isReadOnly()`

   - return – `true/false`

SIP Call Broadcaster

Callback functions invoked by the Cobra API for different SIP call statuses (e.g. established, disconnected, error) are defined under the namespace 'REIC.sipBroadcaster'. Callbacks are accessed by invoking the following APIs:

1. Call Established – This method internally starts polling for kiosk notification status. If the poll returns a session identification, another poll for session status is started and kiosk notification poll is stopped.

   `REIC.sipBroadcaster.onEstablished()`

2. Call Disconnected – The following API disconnects and stops any active polling when invoked:

   `REIC.sipBroadcaster.onDisconnected()`

3. Call Error – The following API stops any active polling if there is an error:

   `REIC.sipBroadcaster.onError(code, explanation)`

   - `code` – Error code
   - `explanation` – Reason of the error

IEC Device API

All IEC related APIs are available under the 'REIC.iec' namespace.

1. Initialization – An IEC device can be initialized by invoking the following API:

   `REIC.iec.init(baudRate, dataBits, flowControl, parity, stopBits, failCallBackFn)`

   - `baudRate` – Baud rate
   - `dataBits` – Data bits
   - `flowControl` – Flow control
   - `parity` – Parity
   - `stopBits` – Stop bits
   - `failCallBackFn` - Callback function provided by the client application for failure

   Example:

   ```
   baudRate=BaudRate19200
   dataBits=DataBits8
   flowControl=FlowControlOff
   parity=ParityNonef
   stopBits=StopBits1
   ```

2. IEC Serial – The following API returns the serial number of the IEC device:

   `REIC.iec.getSerial()`

3. Send Command – The following API sends commands to the IEC device:

   `REIC.iec.sendCommand(command)`

   - `command` – The command that needs to be sent

4. Get Screen Index – To get the screen index use the following API:

   `REIC.iec.getScreenIndex()`

- `return` – Index of the screen

5. Post IEC Reboot – The IEC can be rebooted by invoking the following API:

   `REIC.iec.postIecReboot(successCallBackFn, failCallBackFn)`

   - `successCallBackFn` – Callback function provided by the client application for success
   - `failCallBackFn` - Callback function provided by the client application for failure

Note The IEC must be rebooted after Cisco TelePresence (TP) comes alive.

6. Get Kiosk Notification Information – To get the kiosk notification information, call the following API:

   `REIC.iec.getKioskNotificationInfo(iecSerial, successCallBackFn, failCallBackFn)`

   - `iecSerial` – IEC's serial number
   - `successCallBackFn` – Callback function provided by the client application for success
   - `failCallBackFn` - Callback function provided by the client application for failure

SDK Configuration

REIC SDK must be initialized after loading the SDK. This step is necessary to load the REIC property file from the REM server or any other server. Once the initialization has completed, the IEC's device serial number and the RESC service URL can be accessed. To initialize SDK, use the following API:

`REIC.config.init(config, successCallBackFn, failCallBackFn)`

- `successCallBackFn` – Callback function provided by the client application for success
- `failCallBackFn` - Callback function provided by the client application for failure
- `config` – Optional configuration object (see example below)

```
config = {
hostname: '10.76.8.190',
port: '8443',
path: 'reic'
}
```

For the above configuration, the REIC property path is: https://10.76.8.190:8443/reic/conf/reic.properties.

The following API can access the IEC's device serial number:

`REIC.config.IEC_SERIAL`

The following API can access the RESC service URL:

`REIC.config.RESC_SERVICE_URL`

The whole property file is stored as an object with key-value pairs describing each property. To access the object invoke the following API:

`REIC.config.getProperty()`

Notification and Polling

Notification service is invoked to check for any active REM sessions and TP aliveness once the REIC SDK is loaded. A call to the kiosk notification service returns an active session identification if there is a REM call established. Once the valid session identification is found, polling for session or call status is automatically started to fetch the job details. Intervals for various polling can be configured in the reic.properties file. For detailed information on configuring the property file, please refer to the *REIC Properties* section of this document. The TP aliveness check returns 'true' or 'false'. Once TP comes alive (i.e. it transitions from 'false' to 'true' status), an IEC reboot is performed to restart the IEC device.

Note JavaScript for notification APIs must be loaded after initializing the REIC SDK (`REIC.config.init(config)`).

Polling APIs are defined under the 'REIC.polling' namespace.

1. Kiosk Notification Polling –

   `REIC.polling.startKioskNotificationInfoPoll(interval)`

   `interval` – Interval for polling

2. Session Polling –

   `REIC.polling.startSessionPoll(sessionId)`

   `sessionId` – Session identification

3. TP Aliveness Check Polling –

   `REIC.polling.startTPALiveCheckPoll(interval)`

   `interval` – Interval for polling

4. Notification Polling – Kiosk notification and TP aliveness check polling can be started together by invoking the following API:

   `REIC.polling.startNotificationPoll()`

REIC Properties

The REIC property file is loaded by invoking `REIC.config.init(successCallBackFn, failCallBackFn, config)`. If no config property is provided, the default property file is loaded from the 'conf/reic.properties' location relative to the parent folder of REIC SDK.

The REIC property file contains the following properties:

```
resc.url=https://10.196.97.170:8443/resc/services/
command.kiosk=1!
command.tp=2!

baudRate=BaudRate19200
dataBits=DataBits8
flowControl=FlowControlOff
parity=ParityNonef
stopBits=StopBits1


# Document camera property
camera.serialPort.device=S0
camera.serialPort.baudRate=9600
```

```
camera.serialPort.dataBits=8
camera.serialPort.flowControl=0
camera.serialPort.parity=0
camera.serialPort.stopBits=1

# poll property start
# poll frequency
session.poll=7000
kiosk.notification.poll=5000
tp.poll=10000

# poll flags
kiosk.notification.poll.enable=true
tp.poll.enable=true
# poll property end

#VNC Server
#text | image | none | RE
vnc.server.readonly=false
vnc.server.port=5980
```

Logging

Device level logging (i.e. IEC event logs) is enabled in the API. In case of error in device level logging, the browser level log is enabled automatically. Different types of logging are supported: info, warn, error, and debug. For logging access use the following APIs:

- `REIC.logger.info(text, title)`
- `REIC.logger.warn(text, title)`
- `REIC.logger.error(text, title)`
- `REIC.logger.debug(text, title)`
  - `text` – Message
  - `title` – Title for log context (e.g., logging for scan operation uses ‘SCAN’ as title)

SDK Folder Structure

SDK parent directory is ‘reic_sdk’. The directory contains the following folders:

- ‘src’ for all the source files
- ‘lib’ for libraries used in SDK
- ‘dist’ for minified version of SDK
- ‘doc’ for JS documentation
- ‘conf’ for reic.properties file
- ‘test’ for unit test case code

Minify REIC SDK

REIC SDK is minified using Google Closure Compiler. Since REIC must be initialized manually in the client application before notification poll starts, SDK is minified without the Notification.js file. Notification.js is minified separately. Minified REIC SDK version without notification polling is 'reic.sdk.min.js and notification.min.js' for Notification.js.

Note The 'notification.min.js' must be loaded after invoking `REIC.config.init([args])`.

To minify the SDK invoke the '`ant`' command in the parent directory of REIC SDK. Minified version of the SDK is placed under the 'dist' directory. Build script also creates JS Documentation under the 'doc' folder.

Minimum Requirements

The minimum requirements are the following:

- jQuery 1.8
- JSDoc 3
- QUnit 1.14
- Ant 1.9.4