Logging

Default logger, enabling and disabling

By default, WebSDK functions log will be displayed to the browser console with INFO level

```
09:46:56 INFO webrtc -  Initialized                    flashphoner.js:35631
09:46:56 INFO websocket -  Initialized                 flashphoner.js:35631
09:46:56 INFO core -  Initialized                      flashphoner.js:35631
```

Since build 0.5.28.2753.131 (the source code is available on GitHub by tag 05cb5bd), logging can be fully disabled while API initializing

```
Flashphoner.init({flashMediaProviderSwfLocation: '../../../../media-
provider.swf', logger: null});
```

or in an application code by the following function call

```
Flashphoner.getLogger().setEnableLogs(false);
```

Then, logging can be enabled again if necessary

```
Flashphoner.getLogger().setEnableLogs(true);
```

Log level adjusting

Logging level can be changed while API initializing

```
Flashphoner.init({flashMediaProviderSwfLocation: '../../../../media-
provider.swf', logger: {severity: "WARN"}});
```

or in an application code by the following function call

```
Flashphoner.getLogger().setLevel("WARN");
```

The following logging levels are supported:

Level text constant	Description
ERROR	Errors only

Level text constant	Description
WARN	Errors and warnings
INFO	WebSDK functions normal log (default)
DEBUG	Debug info
TRACE	Execution trace

Push logs to the server

By default, client logs will no be pushed to the server. This feature can be enabled if necessary while API initializing

```
Flashphoner.init({flashMediaProviderSwfLocation: '../../../../media-provider.swf', logger: {push: true}});
```

or in an application code by the following function call

```
Flashphoner.getLogger().setPushLogs(true);
```

In this case client logs will be sent to WCS server via Websocket connection and will be passed to backend using REST hook /pushLogs:

```
10:16:03,335 INFO            RestClient - API-ASYNC-pool-12-thread-5 SEND REST OBJECT ==>
URL:http://localhost:8081/apps/EchoApp/pushLogs
OBJECT:
{
  "nodeId" : "vdUfWbDQUa9TIFfYSwGmXhDs3zp1vH4p@192.168.0.111",
  "appKey" : "defaultApp",
  "sessionId" : "/192.168.0.100:50627/192.168.0.111:8443-966a2082-8033-4982-9bcb-fecc1bd46169",
  "logs" : "10:15:55 INFO webrtc - \"Initialized\"\n",
  "origin" : "https://test.flashphoner.com:8888"
}
```

On Websocket connection closing, client logs will be written to server logs:

Client log example in server logs

Using custom logger

Since build 0.5.28.2753.131 (the source code is available on GitHub by tag 05cb5bd), a custom logger can be defined:

```
var customLogger = {
   error: function (text) {
       console.log("custom logger: ERROR:",text);
   },
   warn: function (text) {
       console.log("custom logger: WARN:",text);
   },
   info: function (text) {
       console.log("custom logger: INFO:",text);
   },
   debug: function (text) {
       console.log("custom logger: DEBUG:",text);
   },
   trace: function (text) {
       console.log("custom logger: TRACE:",text);
   }
};
```

and can be enabled while API initializing

```
Flashphoner.init({flashMediaProviderSwfLocation: '../../../../media-
provider.swf', logger: {customLogger: customLogger}});
```

or in an application code by the following function call

```
Flashphoner.getLogger().setCustomLogger(customLogger);
```

The custom logger example above will display to the browser console

```
custom logger: INFO: Initialized                                    two_way_streaming.js:15
custom logger: INFO: Initialized                                    two_way_streaming.js:15
custom logger: INFO: Initialized                                    two_way_streaming.js:15
Create new session with url wss://test1.flashphoner.com:8443        two_way_streaming.js:54
custom logger: INFO:                                                two_way_streaming.js:15
▶ {audio: true, video: {…}, customStream: undefined}
custom logger: INFO: FOUND WEBRTC CACHED INSTANCE, id               two_way_streaming.js:15
8185c8c0-2081-11ea-b1ef-9de17262c752-LOCAL_CACHED_VIDEO
```

Logging enabling, disabling, level adjusting and pushing logs to the server work for custom logger as well as for default logger.

A separate logger parameters for session, stream or call

Since WebSDK build 2.0.215 it is possible to set a separate logger parameters not only for application, but also for every session, stream or call. In this case, every object uses its own logger instance. For example, let's create a separate custom loggers for stream publishing and playback in Two Way Streaming application:

1. Define custom loggers for publishing

```
var publishCustomLogger = {
   error: function (text) {
       console.log("publish: ERROR:",text);
   },
   warn: function (text) {
       console.log("publish: WARN:",text);
   },
   info: function (text) {
       console.log("publish: INFO:",text);
   },
   debug: function (text) {
       console.log("publish: DEBUG:",text);
   },
   trace: function (text) {
       console.log("publish: TRACE:",text);
   }
};
```

and playback

```
var playCustomLogger = {
   error: function (text) {
       console.log("play: ERROR:",text);
   },
   warn: function (text) {
       console.log("play: WARN:",text);
   },
   info: function (text) {
       console.log("play: INFO:",text);
   },
   debug: function (text) {
       console.log("play: DEBUG:",text);
   },
   trace: function (text) {
       console.log("play: TRACE:",text);
   }
};
```

2. Set custom logger option when creating Stream objects for publishing

```
function publishStream() {
    ...
    session.createStream({
        name: streamName,
        display: localVideo,
        ...,
        logger: {customLogger: publishCustomLogger}
        ...
    }).publish();
}
```

and for playback

```
function playStream() {
    ...
    session.createStream({
        name: streamName,
        display: remoteVideo,
        logger: {customLogger: playCustomLogger}
        ...
    }).play();
}
```

3. The following messages will be displayed in browser xconsole while publishing and playing a stream in modified Two Way Streaming example