XTRMX SDK

Like being in the same place

September 2017

Contents

Introduction

What is XTRMX SDK

XTRMX SDK is a unique platform that enables the creation of collaborative applications through a simple API.

Togetherness: XTRMX SDK enables the creation of applications, for users who wish to co-work, together and simultaneously on the same data, especially with media content involved.

Write for single, deploy for many: In effect, XTRMX SDK is a collection of APIs, that serve as a platform to build collaborative applications. While you concentrate on building a great app, XTRMX SDK manages the users' concurrency “under the hood”.

Realtime Simultaneous Media Manipulation: Although XTRMX SDK provides the means to manipulate any kind of data concurrently, it specializes in manipulating digital media concurrently. With XTRMX SDK, multiple users can manipulate the same media-content simultaneously, and get updated with modifications applied by other users in realtime.

Media Anywhere: XTRMX SDK allows the modification of media, regardless of its location. The media may reside on a single user’s host, yet any user would be able to manipulate it transparently, without the need to upload it to a shared repository. As long as one of the users has access to the media (by network or on his local drive), any colleague would be able to work on it.

Media Format Abstraction: XTRMX SDK manages an abstraction layer on top of the media, so whatever the format, host, or protocol required for streaming the media – it's available for manipulation via the very same API.

Processing Load Balancing: XTRMX SDK dynamically load balances the processing work across the users' devices. Stronger devices are assigned to more processing work, while weaker devices are aided by their more capable partners. The load is dynamically tuned based on network throughput, required quality, user-workflow and other available resources. Over all, the system trades-off high quality, high responsiveness and resource-consumption, to enable weak devices to work beyond their strength, maintain user experience at realtime responsiveness, and optimize quality.

Develop with speed: XTRMX SDK is a JavaScript-based SDK. As such, it can work from any browser or on a node server. No installation required.

Ready for integration: XTRMX SDK also has a native extension in C & Java, to connect the JavaScript engine with the native platform resources – transcoders, GPU, camera, files etc. The native extension enables integrating XTRMX SDK into native applications that expose their own extension SDK.

Audience

XTRMX SDK is intended for anyone who's writing an application or a plugin, that will be used by multiple users simultaneously or when browsing of remote content is required. In addition, it suits applications that require media-processing load balancing, or mixing media content that is hosted on different sources, all in a multi-platform manner. For example, if you need:

- An app for browsing / manipulating media-content on a remote source
- An app that requires abstraction for management variety of media sources (local, web, etc.).
- A Multi-platform application that requires heavy media-processing on one hand, and needs to support weak devices on the other hand.
- Software that manages the same digital data (media-less data included, whatever this data may represent), by multiple concurrent users.
- Browsing the same media (images, videos, audios) from multiple terminals.
- Low-level drivers that require real-time media and data sharing.
- An application for editing any media by multiple users, wherever the media is located – on a local storage, on the cloud or anywhere available on the web.
- Extending an application so it can be used by multiple users at the same time (including applications that internally deal with media)

Then XTRMX SDK is your solution.

Document Overview: How is this document structured?

The first part (that is, this one) is "introduction".

The "Getting Started" part give you a quick review of the API scope. That would be the right place to start with, review the sample code, and try to write something similar yourself.

Then we drill down:

- MX Session: How to create, destroy, join and leave a session, either from server side (using node-js) or client side (using either javascript or C/C++ API)
- MX Streaming API: How to stream your media using a dedicated C/C++ API

- Modus API: How to control collaboratively the stream – play, stop, random access control and more
- Users API: Get notified when users join and leave the session
- MX MVC API: Create your own collaborative data-models (schemes) and control them simultaneously using a realtime collaborated MVC-like system

Getting Started

The following chapter show shortly a full, end to end, XTRMX workflow. The steps shown are:

1) Create a new session from the server
2) Join that session on a C/C++ machine and start streaming
3) Join that session on a JS machine and receive streaming

The first step is to create a session. This step is done from the node server. To authenticate yourself, you should use the authentication-key provided by XTRMX during the registration process (aka “auth”).

When the session is created, a unique session token is generated and returned in the *outSessionData*. This token should be later used by other clients to authenticate themselves when joining to the session.

```
//Import MX SDK
var MXSDK = require('./MXSDK');

//Load MX SDK
var MX = MXSDK("eu.xtrmx.com", 8090);

//Start a new session
MX.newSession(
    "UNBREAKABLEAUTHORIZATIONSTRING",//auth
    //inUserData:
    {
        name: "John", id: "UniqueUserId"
    },
    //inSessionData:
    {
        name: "Session#1"
    },
    //MXOption:
    null,
    //onNewSession callback:
    function (err, outSessionData, MXAPI) {
        //session created !
    });

//Stop the session
MX.stopSession(
    //sessionData:
    {
        id: "ABCD",
        token:"UNBREAKABLEAUTHORIZATIONSTRING"
    },
    //onSessionEnded callback:
    function (err) {

    });

```

Next, we'll join the session from the C/C++ API:

```
#include <MXAPI.h>

//get session API
MXAPI* mx = MXAPI();
IMXSession* mxSessionAPI = mx->getMXSessionAPI();

//register a session callback:
class MyMXSessionCb : public IMXSessionCb
{
public:
	virtual ~MyMXSessionCb() {}

	virtual void OnSessionJoined(const MXStr& err, const MXStr& userId) override {
		//todo
	}
	virtual void OnLeaveSession(const MXStr& err) override {
		//todo
	}

};
MyMXSessionCb cb;
mxSessionAPI->SetSessionCB(&cb);

//join a session:
uint32_t handle = mxSessionAPI->JoinSession("TheSessionId", "TheSessionToken");
```

Once joined into the session, we'll start streaming:

```
IMXStreamingAPI* mxStreamingAPI = mx->getStreamingAPI(handler);

//configure the streaming API to get its data from a third party and send it to the session's
collaborators
mxStreamingAPI->setFlags(MXAPI_SOURCE_FROM_THIRD_PARTY | MXAPI_SEND_BINARY_DATA_TO_MODUS);

//register the IMXStreaming callback
class MyMXStreamingCb : public IMXStreamingCb
{
public:
	virtual IMXStreamingCb() {}
	virtual OnTransportChanged(uint32_t state) {
		//todo
	}
	virtual OnSetPos(uint64_t position, uint32_t version) {
		//todo
	}

};
MyMXStreamingCb cb;
mxStreamingAPI->RegisterStreamingCB(&cb);

//set input audio format
mxStreamingAPI->SetAudioInputFormat(48000, 2, 2);

//set inuput video format
mxStreamingAPI->SetVideoInputFormat(25, 1, 1920, 1080, MAKEFOURCC('R', 'G', 'B', 'A'));

mxStreamingAPI->SubmitAudioBuffer(data, size, 0);
mxStreamingAPI->SubmitVideoBuffer(data, 0, 0)
```

The next step would be to join from the JS-client side, in order to get the stream and control it

```
<script src="https://eu.xtrmx.com:8080/mxsdk.js">
var mxOptions = {
    display:{
        dom:$("#display"),//#display is a DIV where the canvas will be generated inside to display the image
        backgroundColor:0//background color of the displaying canvas (when no image is displayed, e.g. before the media is loaded)
    }
};

MX.joinSession ( inUserData, inSessionData, mxOptions, function(err, MXAPI){
    //get the ModusAPI:
    var modusAPI = MXAPI.ModusAPI;

    //get notified when the stream is ready:
    modusAPI.onStreamReady(function(err){



    //get notified when there is a new stream position
    modusAPI.onNewStreamPosition(function(streamPosData) {
            console.log("The new stream pos: " + streamPosData.currentFrame);
        });
        //and when starting to play
        modusAPI.play(function(){
            conseol.log("Start playing !");
        });
        //and when paused
        modusAPI.pause(function(){
            conseol.log("Stopped playing !");
        });
        //and when muted/unmuted
        modusAPI.mute(function(muted){
            conseol.log("The audio is " + (muted ? "muted" : "unmuted"));
        });

        //what is the current frame ?
        conseol.log("The current frame is: " + modusAPI.currentFrame());

        //modify the transport
        modusAPI.transportLock();//lock the transport before modifying it
        modusAPI.play();//start playing
        modusAPI.transportUnlock();//unlock when done so other can modify the transport as well
        45
    });

});
```

There is much more, of course. There is the users API for the users management, the MX MVC API to create API that match your own customized data model and more.

MX Session: The beginning

MX Session: Server side

MX Session is the container of all the realtime, collaborative activity provided under the MX SDK. Users may join to the same session, and simultaneously control the session attributes (the session's data as described in the MVC chapter, and the session's media as described in the Media chapter) in a synchronized manner.

On the server-side, you one can manage an MX session by following those steps:

1) Import MX SDK
2) Load MX SDK
3) Start a new session
4) Work with MX SDK to manage the session
5) Stop the session

```
//Import MX SDK
var MXSDK = require('./MXSDK');

//Load MX SDK
var MX = MXSDK("eu.xtrmx.com", 8090);

//Start a new session
MX.newSession(
    "UNBREAKABLEAUTHORIZATIONSTRING",//auth
    //inUserData:
    {
        name: "John",id: "UniqueUserId"
    },
    //inSessionData:
    {
        name: "Session#1"
    },
    //MXOption:
    null,
    //onNewSession callback:
    function (err, outSessionData, MXAPI) {
        //session created !
    });

//Stop the session
MX.stopSession(
    //sessionData:
    {
        id: "ABCD",
        token:"UNBREAKABLEAUTHORIZATIONSTRING"
    },
    //onSessionEnded callback:
    function (err) {

    });

```

We'll dive into those steps one by one:

Import MX SDK

The first step to use XTRMX SDK is to import it. On your node server implementation, write:

```
1 var MXSDK = require('./MXSDK');
```

Load MX SDK

MXSDK (address, port)

Load MX SDK on the server side

address
Type: String The address of the XTRMX Server, such as https://eu.xtrmx.com

port
Type: Integer The port used by the XTRMX server to serve the API

```
1          var MX = MXSDK("eu.xtrmx.com", 8090);
```

Start a new session

MX.newSession (auth, [inUserData], [inSessionData], [MXOptions], onNewSession)

Start a new MX Session

Auth
Type: String
An authentication string provided by XTRMX to authenticate the session-initializer. This authentication is your account descriptor, and your account settings are encrypted within. If you don't have your auth-string, please contact XTRMX support

-

[inUserData]
Type: Plain Object
Input information about the session-generating user

> **[Name]**
> **Type:** String
> Optional user name
>
> **UserId**
> **Type:** String
> Unique user id

[inSessionData]
Type: Plain Object
Input information about the new session

> **[Name]**
> **Type:** String
> Optional session name

MXOptions
Type: Plain object
An object describes the supported MVC-data-models in this session. See the MVC chapter for additional details

onNewSession(err, outSessionData, MXAPI)
Type: Function
Callback function invoked when the new session is ready (or upon failure)

> **Err**
> **Type:** String
> A string representation of the error in case of failure, or null in case of success
>
> **outSessionData**

Type: Plain object
Representing the session (a reflection of the inSessionData with the addition of a session-id

name
Type: String
The session name (if it was submitted in the inSessionData)

id
Type: String
A unique session-id

token
Type: String
A unique, JWT encrypted session token. This is a top security issue: Using that token (and this token only) other clients can join the session. Be careful regarding how and to whom this token is handed

MXAPI
Type: Plain object
A collection of APIs

ModusAPI
Type: Object
API for realtime collaboration transport (SetPos, Play, Stop etc)
See the Modus chapter for additional details

UsersAPI
Type: Object
API for realtime activities of the users (whos in, whose out etc)

```
MX.newSession(
    "UNBREAKABLEAUTHORIZATIONSTRING",//auth
    //inUserData:
    {
        name: "John", id: "UniqeUserId",
    },
    //inSessionData:
    {
        name: "Session#1"
    },
    //MXOption:
    null,
    //onNewSession callback:
    function (err, outSessionData, MXAPI) {
      //session created !
    });
```

Stop the session

MX.stopSession ([sessionData], onSessionEnded)

Stop the session

[sessionData]
Type: Plain Object
The session to be stopped

name
Type: String
session name

Id
Type: String
A unique user-id for that session

token
Type: String
A unique, JWT encrypted session token

onSessionEndedSession(err)
Type: Function
Callback function invoked when the session ended (or upon failure)

err
Type: String
A string representation of the error in case of failure, or null in case of success

```
MX.stopSession(
    //sessionData:
    {
        id: "ABCD",
        token:"UNBREAKABLEAUTHORIZATIONSTRING"
    },
    //onSessionEnded callback:
    function (err) {

    });
```

MX Session: JS Client side

A JS Client may join and leave the session using the session id and the session token. When a user joined the session, it gets a unique user id for that session which is used as this user identifier (for that session). More details can be added regarding that user – its name, email etc – allowing to easily track and manage the session's user.

MX.joinSession ([inUserData], [inSessionData], [MXOptions], onJoinSession)

Join an existing MX Session

[inUserData]
Type: Plain Object
Input information about the session-generating user

[Name]
Type: String
Optional user name

userId
Type: String
unique user id

inSessionData
Type: Plain Object
Input information about the new session

[Name]
Type: String
Optional session name

id
Type: String
A unique session-id (received when the session was generated)

token
Type: String
A unique, JWT encrypted session token (received when the session was generated)

MXOptions
Type: Plain object
An object describes the supported MVC-data-models in this session. See the MVC chapter for additional details

onJoinSession(err, MXAPI)

Type: Function
Callback function invoked when the new session is ready (or upon failure)

err
Type: String
A string representation of the error in case of failure, or null in case of success

MXAPI
Type: Plain object
A collection of APIs

ModusAPI
Type: Object
API for realtime collaboration transport (SetPos, Play, Stop etc)
See the “Modus” chapter for additional details

UsersAPI
Type: Object
API for realtime activities of the users (whos in, whose out etc)

```
MX.joinSession(
    //inUSerData:
    {
        name: "John", id: "UniqueUserId"
    },
    {
        sessionId: "ABCD",
        token:"UNBREAKABLEAUTHORIZATIONSTRING"
    },
    //mx options
    null,
    //onJoinSession callback:
    function (err, MXAPI) {

    });
```

MX.leaveSession ([onLevaeSession])

Leave the current MX Session

[onLeaveSession(err)]
Type: Function
Callback function invoked when left the session

err
Type: String
A string representation of the error in case of failure, or null in case of success

```
MX.leaveSession(
    function (err) {

    });
```

MX Session: C Client side

A C Client may join and leave the session using the session id and the session token. When a user joined the session, it gets a unique user id for that session which is used as this user identifier (for that session). More details can be added regarding that user – its name, email etc – allowing to easily track and manage the session's user.

The MX C-API contains the following steps to manage a session:

1. Get the session API
2. Register a session-callback
3. Join a session
4. Leave a session

The following example shows a sample code of those steps:

```
#include <MXAPI.h>

//get session API
MXAPI* mx = MXAPI();
IMXSession* mxSessionAPI = mx->getMXSessionAPI();

//register a session callback:
class MyMXSessionCb : public IMXSessionCb
{
public:
          virtual ~MyMXSessionCb() {}

          virtual void OnSessionJoined(const MXStr& err, const MXStr& userId) override {
                    //todo
          }
          virtual void OnLeaveSession(const MXStr& err) override {
                    //todo
          }

};
MyMXSessionCb cb;
mxSessionAPI->SetSessionCB(&cb);

//join a session:
uint32_t handle = mxSessionAPI->JoinSession("TheSessionId", "TheSessionToken");
//...
//leave a session:
mxSessionAPI->LeaveSession(handle);
```

Get the session API

The session API is done by including the C-MXAPI header, generating the MXAPI object (which will internally dynamically load the MXAPI library) and extracting the IMXSessionAPI

```
#include <MXAPI.h>


MXAPI* mx = MXAPI();
IMXSession* mxSessionAPI = mx->getMXSessionAPI();
```

IMXSessionCb

The MXSessionCb allows the user to implement and register a callback to be invoked upon session joining/leaving

IMXSessionCb

Session callback

Notify when joining/leaving a sesion

```
class IMXSessionCb
{
public:
	virtual ~IMXSessionCb() {}

	//! Invoked when joined the session
	/*!
	\param err: In case of error, the err string will be none-empty
	\param userId: A unique identifier to the user
	*/
	virtual void OnSessionJoined(const MXStr& err, const MXStr& userId) = 0;

	//! Invoked after the session was left
	/*!
	\param err: In case of error, the err string will be none-empty
	*/
	virtual void OnLeaveSession(const MXStr& err) = 0;
};
```

IMXSession

The IMXSession interface allows the user to register the IMSessionCb callback, and to join/leave sessions

IMXSession header

```
class IMXSession
{
public:
	virtual ~IMXSession() {}

	//! Set the MX Session callback
	/*!
	\param cb: Sessoion callback
	*/
	virtual void SetSessionCB(IMXSessionCb* cb) = 0;

	//! Join an existing session
	/*!
	\param sessionId: The unique session id (received when creating the
session)
	\param sessionToken: The session token (received when creating the
session)
	\return: A handle to that session
	*/
	virtual uint32_t JoinSession(const MXStr& sessionId, const MXStr&
sessionToken) = 0;

	//! Leave the session
	/*!
	\param handle: A handle to the session to leave
	*/
	virtual void LeaveSession(uint32_t handle) = 0;
};
```

IMXSession::SetSessionCB(IMXSessionCb cb)*

Register an IMXSessionCb

cb
Type: IMXSessionCb A Session callback object

IMXSession::JoinSession(const MXStr& sessionId, const MStr& token)

Join an existing session

sessionId
Type: MXStr The unique session id (that was received when generating that session)

token
Type: MXStr The unique session's token (that was received when generating that session)

return
Type: uint32_t A handler to that session

IMXSession::LeaveSession(uint32_t handler)

Leave an existing session

handler
Type: uint32_t A handler to that session

MX Streaming API

Streaming is generated by the “streamer” side. There is a JS and C APIs to initiate streams, which might be either:

1) Streams generated out of files
2) Streams generated out of capture device
3) Stream generated out of a third party

Streaming: Streamer C side

Given that the client had join a session, The Streaming C API allows the streamer to stream, in the following steps:

1. Get the streaming API
2. Configure the C-API to stream
3. Register the transport callback (to be notified on a request to change the transport state or a set-position request)
4. Set Audio/Video input formats: Configure the formats to be submitted (those routines may be called again each time the input format changes)
5. Optionally, set the audio/video output (streaming) format. If those routines aren’t called, the default settings are used
6. Streaming: Submit video/audio data

```
#include <MXAPI.h>

//get the streaming API using a session handler
MXAPI* mx = MXAPI();
IMXStreamingAPI* mxStreamingAPI = mx->getStreamingAPI(handler);

//configure the streaming API to get its data from a third party and send it to the session's
collaborators
mxStreamingAPI->setFlags(MXAPI_SOURCE_FROM_THIRD_PARTY | MXAPI_SEND_BINARY_DATA_TO_MODUS);

//register the IMXStreaming callback
class MyMXStreamingCb : public IMXStreamingCb
{
public:
        virtual IMXStreamingCb() {}
        virtual OnTransportChanged(uint32_t state) {
                //todo
        }
        virtual OnSetPos(uint64_t position, uint32_t version) {
                //todo
        }

};
MyMXStreamingCb cb;
mxStreamingAPI->RegisterStreamingCB(&cb);

//set input audio format
mxStreamingAPI->SetAudioInputFormat(48000, 2, 2);

//set inuput video format
mxStreamingAPI->SetVideoInputFormat(25, 1, 1920, 1080, MAKEFOURCC('R', 'G', 'B', 'A'));

//....
//submit data (over and over)
mxStreamingAPI->SubmitAudioBuffer(data, size, 0);
mxStreamingAPI->SubmitVideoBuffer(data, 0, 0);
```

IMXStreaminCb

Used as a callback to notify the user about changed in the transport state or in case the position had changed

IMXStreamingCb

```
class IMXStreamingCb
{
public:
        virtual ~IMXStreamingCb() {}

#define MX_TRANSPORT_STATE_PLAYING                0x0001
#define MX_TRANSPORT_STATE_SCRUBBING      0x0002
#define MX_TRANSPORT_STATE_MUTED          0x0004

        //! Invoked when the transport changed
        /*!
        \param state: A bitwise of MX_TRANSPORT_STATE_X
        */
        virtual OnTransportChanged(uint32_t state) = 0;

        //! Invoked when the position is modified
        /*!
        \param position: The position (in frames)
        \param version: An ever incrementing version that identify that set-position call (to
be associated with the buffer submitting, see SubmitVideoBuffer)
        */
        virtual OnSetPos(uint64_t position, uint32_t version) = 0;

};
```

IMXStreamingAPI

Allows streaming video and audio data by transcoding them on the fly. The audio and video input formats has to be pre-configured, and may be modified in case the input format changes.

IMXStreamingAPI Header

```
class IMXStreamingAPI
{
public:
        virtual ~IMXStreamingAPI() {}

        //! Configure the C-streaming-API role (streaming from file ? from third party ?
receiving? displaying?)
        /*!
        \param handle: the session handle
        \param flags: configuration for streaming/recieving/displaying.
        */
#define MXAPI_SOURCE_FROM_FILE                          0x00000001/*read from a file*/
#define MXAPI_SOURCE_FROM_STREAM                0x00000002/*read from an incoming stream*/
#define MXAPI_SOURCE_FROM_THIRD_PARTY           0x00000004/*media from a third party software
(such as NLE)*/
#define MXAPI_RENDER_ON_LOCAL_WINDOW            0x00000008/*render source onto local window*/
#define MXAPI_RENDER_ON_LOCAL_CLIENT            0x00000010/*render source via the local
client's canvas*/
#define MXAPI_SEND_BINARY_DATA_TO_MODUS         0x00000020/*send source to modus server*/
        typedef uint64_t MXAPIFlags;
        virtual void SetFlags(MXAPIFlags flags) = 0;

        //! Register a callback to be invoked when the streaming state modified
        /*!
        \param cb: The streaming callback
        */
        virtual void RegisterStreamingCB(IMXStreamingCb* cb) = 0;

        //! Declare the audio input format
        /*!
        \param sampleRate: 8000-48000
        \param channel: number of channels (note: Currently downmixed to mono or stereo when
streamed)
        \param sampleType: 8bit-32bit, or float at range [-1,1)
        \return true in case of success. Call MXAPI::GetLastError to get a description of
error in case of failure
        */
        virtual bool SetAudioInputFormat(
                uint32_t sampleRate,
                uint32_t channels,
                int sampleType) = 0;//ST_PCM_U8 = 1,ST_PCM_S16 = 2,ST_PCM_S24 = 3,ST_PCM_S32 =
4,ST_PCM_U32 = 5,ST_PCM_FLOAT = 6

        //! Declare the video input format
        /*!
        \param sampleRate : 8000 - 48000
        \param rate: The rate of the frame-rate
        \param scale: The scale of the frame rate
        \param width: width in pixels
        \param height: height in pixels
        \param colorModel: Either RGB or RGBA, with optionally flipped R and B
        \param bitsPerChannels: Only 8 bits per channel are currently supported
        \param pitch: number of bytes per row
        \param angle: Flipped angle (or 0)
        \return true in case of success.Call MXAPI::GetLastError to get a description of error
in case of failure
        */
```

```
        typedef unsigned long FOURCC;/* a four character code */
        virtual bool SetVideoInputFormat(
                uint32_t rate,
                uint32_t scale,
                uint32_t width,//in pixels
                uint32_t height,//in pixels
                FOURCC colorModel,//Currently supported: RGB3,BGR3,RGBA,BGRA
                uint32_t bitsPerChannels = 8,//currently supported: 8
                uint32_t pitch = 0,//number of bytes per row. value <= 0 means:
width*(bitsPerChannel/8)*channels
                int angle = 0);//if flipped vertically, indicate 180. Currently supported: 0,
90, 180, 270).


        //! Set the audio output format (will use the default if not invoked)
        /*!
        \param codec: only '.mp3' is currently supported
        \param channel: number of channels to stream (only 1 or 2 downmix)
        \param sampleRate: 8000-48000Hz
        \param bitRate: 64000-256000
        \return true in case of success. Call MXAPI::GetLastError to get a description of
error in case of failure
        */
        typedef unsigned long FOURCC;/* a four character code */
        virtual bool SetAudioOutputFormat(
                FOURCC codec,//only '.mp3' is currently supported
                uint32_t channels,
                uint32_t sampleRate,
                uint32_t bitRate) = 0;

        //! Set the video output format (will use the default if not invoked)
        /*!
        \param codec: only 'hevc' is currently supported
        \param width: output streaming width dimension
        \param height: output streaming height dimension
        \return true in case of success. Call MXAPI::GetLastError to get a description of
error in case of failure
        */
        virtual bool SetVideoOutputFormat(
                FOURCC codec,
                uint32_t width,
                uint32_t height) = 0;


        //! Submit audio data to stream according to the declared input audio format
        /*!
        \param data: array of bytes per each channel
        \param dataSize: size of each channel in bytes (size must be equal for all channels)
        \param posInSamples: relative to stream
        \return true in case of success. Call MXAPI::GetLastError to get a description of
error in case of failure
        */
        virtual bool SubmitAudioBuffer(
                uint8_t** data,
                uint32_t dataSize,
                uint64_t posInSamples) = 0;

        //! Submit video data to stream according to the declared input video format
        /*!
        \param data: a linear image
        \param posInFrames: relative to stream
        \param versopm: An ever incrementing version that identify the originating set-
position call (see IMXStreamingCb::OnSetPos)
        \return true in case of success.Call MXAPI::GetLastError to get a description of error
in case of failure
        */
        virtual bool SubmitVideoBuffer(
                uint8_t* data,
                uint64_t posInFrames,
```

```
                uint32_t version) = 0;

};
```

IMXSession::SetFlags(MXAPIFlags flags)

Configure the C-Streaming-API roles: To stream from a file / third party, to receive or to display

Flags

Type: MXAPIFlags (typedef uint64_t)

A bitwise flags to configure the MX session, one of the following values ORed:

```
#define MXAPI_SOURCE_FROM_FILE                0x00000001/*read from a file*/
#define MXAPI_SOURCE_FROM_STREAM              0x00000002/*read from an incoming stream*/
#define MXAPI_SOURCE_FROM_THIRD_PARTY 0x00000004/*media from a third party software (such as NLE)*/
#define MXAPI_RENDER_ON_LOCAL_WINDOW  0x00000008/*render source onto local window*/
#define MXAPI_RENDER_ON_LOCAL_CLIENT  0x00000010/*render source via the local client's canvas*/
#define MXAPI_SEND_BINARY_DATA_TO_MODUS       0x00000020/*send source to modus server*/
```

IMXSession::RegisterStreamingCB(IMXStreamingCB cb)*

Register a callback to be notified when the streaming transport changes, or when a new position is requested

cb

Type: IMXStreamingCb

IMXSession::SetAudioInputFormat(uint32_t sampleRate, uin32_t channels, int sampleType)

Set the input audio format to be sumibtted

sampleRate

Type: Integer

8000-48000Hz

Channels

Type: Integer

One (mono) or two (stereo)

SampleType

Type: Integer

One of the following enum:

```
{ST_PCM_U8 = 1,//8 bit unsigned
ST_PCM_S16 = 2,//16 bit signed
ST_PCM_S24 = 3,//25 bit signed
ST_PCM_S32 = 4,//32 but signed
ST_PCM_U32 = 5,//32 bit unsigned
```

```
ST_PCM_FLOAT = 6}//float at range [0,1)
```

IMXSession::SetVideoInputFormat(uint32_t rate, uint32_t scale, uint32_t width, uint32_t height, FOURCC colorModel, unit32_t bitPerChannel, uint32_t pitch, int angle)

Set the input audio format to be sumibtted

Rate
Type: Integer
The rate of the frame

Scale
Type: Integer
The scale of the frame rate

width
Type: Integer
Width in pixels

height
Type: Integer
Height in pixels

colorModel
Type: FOURCC (typedef unsigned long)
Currently Supported: RGB3, BGR3, RGBA and BGRA

bitsPerChannel
Type: Integer
Bit depth per channel. Currently Supported: 8 bits

pitch
Type: Integer
Number of bytes per pixels-raw. value <= 0 means: width*(bitsPerChannel/8)*channels

angle
Type: Integer
Flipped angle in degrees (0 for no flipping). Currently supported: 0, 90, 270

IMXSession::SetAudioOutputFormat(FOPURCC codec, uint32_t channels, uint32_t sampleRate, uint32_t bitRate)

Set the output audio format. If not set, the default will be used (Default audio/video formats are configured through the admin API/Portal)

codec
Type: FOURCC (typedef unsigned long)
Currently only '.mp3' is supported

channels
Type: Integer
Number of streamed channels (currently only 1 or 2 is supported)

sampleRate
Type: Integer
[8000-48000] Hz

bitRate
Type: Integer
[64000-256000] bps

IMXSession::SetVideoOutputFormat(FOPURCC codec, uint32_t width, uint32_t height)

Set the output video format. If not set, the default will be used (Default audio/video formats are configured through the admin API/Portal)

codec
Type: FOURCC (typedef unsigned long)
Currently only 'hevc' is supported

width
Type: Integer
Width of image in pixels

height
Type: Integer
Height of image in pixels

*IMXSession::SubmitAudioBuffer(uint8_t** data, uint32_t size, uint64_t posInSamples)*

Submit the next audio buffer to stream

data
Type: Array of bytes-array
Each array represents a channel. The array must be of the same size. The format must complies with the format submitted to the 'setAudioInputFormat'

size
Type: Integer
Size in bytes of each array

posInSamples
Type: Integer
Relative to the source zero-position

IMXSession::SubmitVideoBuffer(uint8_t data, uint64_t posInFrames, uint32_t version)*

Submit the next audio buffer to stream

data
Type: Bytes array
Representing an image with the same format submitted to the 'setInputVideoFormat'

posInFrames
Type: Integer
Relative to the source zero position

version
Type: Integer
An ever-incrementing version, identify the set-pos call that triggered that position-modification

Modus API: Media for all

XTRMX media framework and its related components are lovingly named “Modus”, because it changes so fundamentally the way you can work with media. Using Modus, multiple users can work on multiple media content, regardless of where the content is hosted or where the users are. The content might be on one of the users’ local devices, on the network or in the cloud – yet all users of the same session will be able to manipulate that content, and in a simultaneous fashion.

The API allows different controlling types: Transport (Play/Pause/Set-position), Transformations (Visual and auditory effects), analysis (such as object detection) and more.

The following API details the transport control of a single asset. Multiple assets control, effects and analyze APIs are currently in validation process and will be released in the next SDK releases.

In addition to being a media realtime simultaneous engine, one of the unique features of Modus is its so-called “fragmentation”. Although transparent to the API-user, Modus can apply many of the operations in multiple hosts. Using that capability, many of the media processing operations might be executed on a particular client, even if the original media isn’t hosted on that client. With this capability, Modus can compromise between quality – whenever the processing is applied on the original media, responsiveness – whenever the processing is done on the modification-triggering client, and resource-balancing – apply the processing on the machine with the appropriate resources (CPU/GPU etc). The compromise between the three – quality, responsiveness, resource-balancing – is changed dynamically, with a change in the content, the users' scenarios and environment settings (such as the machines' resources-consumption or the dynamically available bandwidth).

Transport control – JS Client side

Once a user joined a session, it can easily get a notification when a stream is ready, display it on a DOM canvas, and control / modify the transport state (play/stop/position) in a collaborative manner.

The ModusAPI is part of the MXAPI returned on the sessionJoined callback (see the “MX Session – JS Client Side” chapter).

In order to manage a stream from a client js-side, one should follow those steps:

1. Join a session. The MXOptions may contain the ‘display’ attribute with a nested ‘dom’ attribute, that points to a div element. In that case, Modus will create canvas embedded in that dom element, where the frames will be displayed.
2. Extract the Modus API
3. Get notified when the stream is ready
4. Subscribe the Modus callback to get notified when the stream is changed
5. Modify the stream is required

The following example showing most of those steps

```
//join a session:
var mxOptions = {
    display:{
        dom:$("#display"),//#display is a DIV where the canvas will be generated inside to display the image
        backgroundColor:0//background color of the displaying canvas
    }
};

MX.joinSession ( inUserData, inSessionData, mxOptions, function(err, MXAPI){
    //get the ModusAPI:
    var modusAPI = MXAPI.ModusAPI;

    //get notified when the stream is ready:
    modusAPI.onStreamReady(function(err){



        //get notified when there is a new stream position
        modusAPI.onNewStreamPosition(function(streamPosData) {
            console.log("The new stream pos: " + streamPosData.currentFrame);
        });
        //and when starting to play
        modusAPI.play(function(){
            conseol.log("Start playing !");
        });
        //and when paused
        modusAPI.pause(function(){
            conseol.log("Stopped playing !");
        });
        //and when muted/unmuted
        modusAPI.mute(function(muted){
            conseol.log("The audio is " + (muted ? "muted" : "unmuted"));
        });

        //what is the current frame ?
        conseol.log("The current frame is: " + modusAPI.currentFrame());

        //modify the transport
        modusAPI.transportLock();//lock the transport before modifiying it
        modusAPI.play();//start playing
        modusAPI.pause();//stop playing
        modusAPI.mute(true);//mute the audio
        modusAPI.setPosition(5);//set position to 5-frames
        modusAPI.transportUnlock();//unlock when done so other can modify the transport as well

    });

});
```

ModusAPI.onStreamReady(onStreamReadyCb)

Get notified when the stream is ready

onStreamReadyCb(err)
Type: function
Callback function to be notified when the stream is ready

err
Type: String
A string representation of the error in case of failure, or null in case of success

ModusAPI.onNewStreamPosition(onStreamPositionCb)

Get notified when the stream is ready

onStreamPositionCb(positionData)
Type: function
Callback function to be notified when the stream position is modified (that is, each time a new frame is displayed)

positionData
Type: Plain object
An object with data regarding the current stream position

currentFrame
Type: Integer
The current frame's position in frames, relative to the source beginning

currentVersion
Type: Integer
The version of this frame (Each new set-position request trigger a new version. The version number is ever incrementing)

ModusAPI.play(onPlayCb)

Start playing / Get notified when playback started

[onPlayCb()]
Type: function
If given, register the onPlayCb function, so it will be invoked when the stream started playing.
If undefined, start playing

ModusAPI.pause(onPauseCb)

Stop playing / Get notified when playback stopped

[onPausedCb()]
Type: function

If given, register the onPauseCb function, so it will be invoked when the stream stopped playing.
If undefined, stop playing

ModusAPI.mute(muted)

Mute / Unmute

muted
Type: boolean
If muted is true or undefined, mute the stream. If false, unmute it

ModusAPI.mute(onMute)

Register a callback to be notified when the mute-state is modified

onMute(muted)
Type: function
Invoke the method when the mute state is modified

ModusAPI.getCurrentFrame()

return the current position of the stream (in frames)

ModusAPI.transportLock(cb)

Lock the stream. While locked, no other user can change the transport

[cb(err)]
Type: function
An optional callback that notified to confirm the lock

err
Type: String
If none-empty, the lock is rejected and the reason is given in this value

ModusAPI.transportUnlock()

Unlock the stream. While locked, no other user can change the transport

ModusAPI.setPosition(frames)

Modify the stream's position

frames
Type: Integer Stream position in frames, relative to the source beginning

ModusAPI.canvas.resize()

When the DOM element where the canvas embedded in is resized, call this method so the displaying canvas will be resized accordingly

User API: Who's there?

Since an MX Session is a collaborative entity, it's commonly important to know at realtime who joined and who left. This is what the user API is good for.

The user API is available via the MXAPI object returned when joining the session. In order to use the User API, one should follow those steps:

1. Join a session
2. Extract the User API
3. Manage the users: Get notified when a new user joins, or another user left

```
1//join the session:
2 MX.joinSession ( inUserData, inSessionData, null, function(err, MXAPI){
3     //get the ModusAPI:
4     var usersAPI = MXAPI.UsersAPI;
5
6     //get notified when a user leave the session
7     usersAPI.onUserQuit(function(userData){
8        console.log("User: " + userData.name + " id: " + userData.id + " just left
the session");
9     });
10
11    //get notified when a user join the session
12    usersAPI.onUserAdded(function(userData){
13        console.log("User: " + userData.name + " id: " + userData.id + " just
joined the session");
14    });
15
16    //how many users are currently inside the session ?
17    console.log("There are " + usersAPI.howMany() + " users inside");
18
19    //get all users (a map from their id to the user data)
20    var users = usersAPI.getUsers();
21    for (var userId in users) {
22        console.log("This is the data of user " + userId + ": " +
JSON.stringify(users[userId]));
23    }
24
25 });
```

UsersAPI.onUserQuit(onUserQuitCb)

Get notified when a user leave the session

onUserQuitCb(userData)
Type: function
Callback to be notified when a user left the session

userData
Type: Plain object
contain details regarding the user that left the session

name
Type: String
The user name

Id
Type: String
A unique user-id for that session

UsersAPI.onUserAdded(onUserAddedCb)

Get notified when a user join the session

onUserAdded(userData)
Type: function
Callback to be notified when a user joined the session

userData
Type: Plain object
contain details regarding the user that joined the session

Name
Type: String
The user name

id
Type: String
A unique user-id for that session

UsersAPI.howMany()

Return the number of users currently login in to the session

UsersAPI.getUsers()

Return map of user id to user data

```
//get all users (a map from their id to the user data)
    var users = usersAPI.getUsers();
    for (var userId in users) {
        console.log(userId + ": " + JSON.stringify(users[userId]));
    }
```

MX MVC API: Concurrent Manipulation

This chapter explores the process of creating an MVC-like API for your scheme and using that API to modify the data and get updated whenever someone else changes the data in the scheme. Of course, all those changes will apply concurrently for multiple users, multiple platforms and devices in real-time.

An API's life cycle starts with compiling the scheme into an API that's associated to it. Then, the API may be used to create sub-APIs that are associated with only a sub-part of your scheme. This allows controlling a specific attribute in your scheme (a JSON, a primitive or a map) via a dedicated API. You can use this API to change and to subscribe-for-changes, for the respective scheme-subpart.

Creating multiple APIs for the same scheme attribute is useful when your client has more than one UI representation for that attribute. Once the API is not required anymore, it should be destroyed.

The usage of multiple APIs for multiple UI representations is much like the usage of a typical MVC. If you're not familiar with the MVC pattern, take a few minutes to read about it, it's an important concept when developing *any* application, and in particular with XTRMX SDK.

The first step is to define your scheme.

The Scheme: Tell us about your world

You have your own application with its own terminology. We encourage you to keep using it. To do so, we introduce our data-model representation we call “The Scheme”. The scheme, represents the collaborative data-model of your app and it may include different kinds of data values and types.

To present the scheme, we’ll use a simple example - a simultaneous task-management application. Tasks can be created, deleted and modified, categorized and archived. Tasks also contain different data types and values.

Scheme Name: The scheme is represented as an object literal with a key-value pair that describes your scheme name and your data-model correspondingly. MX API supports multiple schemes, so it’s important that each scheme will have its own unique name.

In our example the scheme represents a task, therefore we’ll name it “TaskScheme” and we’ll define it as follows:

```
var taskManagementScheme = {
    TaskScheme: {}
};
```

Primitives & Objects: strings, integers, floating point numbers and Booleans are useful in representing an application state. The scheme syntax incorporates these types as key-value pairs contained in an object literal, such that the key represents the semantic meaning of the data and the value determines its type and its default value.

In our example, first we’ll add a “Task” object to the scheme that will encapsulate the overall state of the task.

```
var taskManagementScheme = {
    TaskScheme: {
        Task: {}
    }
};
```

Next, we'll add some attributes to the task's data-model:
Title (String) – the name of the task
Description (String) – a short description of the task
Status (Integer) – an enumerator that describes the state of the task.
We will define it as follows:

```
var taskManagementScheme = {
    TaskScheme: {
        Task: {
            Title: "",
            Description: "",
            Status: 0
        }
    }
};
```

In addition we'll add a group of flags:
Active (Boolean) – to indicate if the task is active
Archived (Boolean) - to indicate if the task is archived
Urgent (Boolean) - to indicate if the task is urgent
We'll add these flags to the scheme bundled as an object literal with the key "Flags" (to keep things semantically clean):

```
var taskManagementScheme = {
    TaskScheme: {
        Task: {
            Title: "",
            Description: "",
            Status: 0,
            Flags: {
                Active:false,
                Archived:false,
                Urgent:false
            }
        }
    }
};
```

Got it? Any primitive – be it a String, an Integer, a Float or a Boolean, packed into whatever hierarchical structure, is welcome.

Maps: Data-models often contain inner data-models that should be dynamically added or removed. To support that, any object literal in the scheme may be wrapped with rectangular brackets. These brackets indicate that their content is actually a set of items that may dynamically grow or shrink in number.

Back to our task list example - naturally, we will have a number of tasks, not just one. Therefore, we'll wrap the Task object literal with a Tasks map, and we'll rename "TaskScheme" to "Task**s**Scheme":

```
var taskManagementScheme = {
    TasksScheme: {
        Tasks: [{
            Task: {
                Title: "",
                Description: "",
                Status: 0,
                Flags: {
                    Active: false,
                    Archived: false,
                    Urgent: false
                }
            }
        }]
    }
};
```

Public methods: Public methods mark attributes in the scheme that are frequently used. A public method allows quick-access wherever the value is in the scheme hierarchy.

In our task list example, let's assume "Status" and "Active" are frequently used:

```
var taskManagementScheme = {
    TasksScheme: {
        Tasks: [{
            Task: {
                Title: "",
                Description: "",
                Status: 0,
                Flags: {
                    Active: false,
                    Archived: false,
                    Urgent: false
                }
            }
        }]
    },
    publicMethods:["Status","Active"]
};
```

Constraints: The scheme syntax has some constraints that should be taken into account when defining a scheme:

1. **Keys-Uniqueness**
   Keys in the scheme do not have to be unique, but in most cases it is convenient to keep them as such. In order to use the API to control an attribute, the SDK "walks" through the scheme and looks for the appropriate key. An API that corresponds to a unique key may be retrieved simply by specifying the key, whereas if the key repeats itself, retrieving its corresponding API will require a more extensive description of the key's relative position within the scheme.
   <CODEC SNIPPEST>

2. **Reserved keys**
   The keys "_id" and "_selected" are reserved and should not be defined in your custom scheme.

3. **Arrays of primitive values**
   Arrays of primitive values are currently not supported as part of the scheme.
   -

4. **Growing the Scheme Hierarchy Dynamically**
   Dynamically growing schemes – such as a folder-structure- are currently not supported.

Next step: Define and build a collaborative API for your scheme
Once you have your scheme ready, MX Compiler may be used to compile it and create a corresponding collaborative API. Call MX to create a new session with your MXAPI.

Create a session your scheme

When creating a new session, or joining an existing one, you can specify as part of the MXOptions object, the scheme to compile.

So how do you get your API? As soon as the session is ready, your callback to the joinSession (or newSession) will be invoked, with the APIs compiled from your schemes as part of the MXAPI argument. You can get your API from there. Each scheme is mapped to its own API, with the scheme name followed by 'API' postfix. For example, if your scheme is called "TasksScheme", then your API will be called "TasksSchemeAPI", and will be available as an attribute of the MXAPI object:

```
var mxOptions = {
    schemes: {TaskScheme:taskManagementScheme}//add your scheme
};
//join the session:
MX.joinSession ( inUserData, inSessionData, mxOptions, function(err, outUserData,
MXAPI){
    //get your API
    var TasksSchemeAPI = MXAPI.TasksSchemeAPI;
});
```

Sub-Scheme APIs: I'm interested only in that

The API provided to the onMXStarted callback of the MX() method controls the whole scheme. Naturally, you'd want to control only part of your scheme at a time.
For example, you might want to change only the title attribute in your scheme, or you'd want to do something specifically when the title attribute is changed.
To do that, a dedicated API (that is responsible just for the part you're interested in) may be generated out of the "container" API that you have in hand.

api.select(selector)

Retrieves an API for an attribute in the scheme that is a descendant of the attribute that corresponds to api (by which the selection is invoked).

Selector
Type: String The sub-scheme API name as it was defined in the scheme

The way to get the API you want, is by using the *select* method. *select* takes a single parameter – the name of the attribute you wish to control, and returns the corresponding API.

```
    var onMXReady = function (apis) {
        TaskSchemeAPI = apis.TaskSchemeAPI;
        TaskSchemeAPI.select("Tasks");
```

Selecting attributes with the same name
If you wish to generate an API for an attribute name that exists more than once in the sub-scheme of the API at hand, you'll have to call *select* multiple times to fetch it.

General select vs. selecting values in lists
Generally, when you select an attribute in your current API's sub-scheme, it doesn't matter how deep the attribute is – the select method will retrieve it. Yet, if you wish to fetch an API for an item that is within a list (a list-item), this will require you to first select the list attribute itself (to get the list's API), and then to select the item's API.

Why? Because a list contains multiple instances of the same object-type, with the same attributes. So you can't simply ask for an API that controls an attribute that's inside that list – our telepathic power is not sharp enough to understand which of the many instances you meant

```
var bigDollAPI = matryoshkaDollAPI.select("Doll");
var mediumDollAPI = bigDollAPI.select("Doll");
var smallDollAPI = mediumDollAPI.select("Doll");
```

Public method: Come here a lot?
An alternative for using *select* is the usage of public methods. Like *select*, public methods will return the required API, the only difference is the syntax. Instead of calling *select*([attribute name]), we'll use [attribute name]()

Note that to use the public method syntax, the attribute has to be marked as public, see scheme-public-methods

```
var myScheme = {
    TaskScheme: {
        Tasks: [{
            Task: {
                Title: "",
                Description: "",
                Status: 0,
                Flags: {
                    Active: false,
                    Archived: false,
                    Urgent: false
                },
                CheckList: [{
                    CheckItem:{
                        Name:"",
                        Completed:false
                    }
```

```
                            }]
                        }
                    }]
                },
                publicMethods: ["Status", "Active", "Urgent",
"Completed"]
            };
….
TaskSchemeAPI.select("Tasks").select(taskId).urgent(urgent);
```

For the API-consistency, public methods are always lower-case letters, regardless of the attribute name in the scheme-definition.

api.destroy()

Destroying an API: Once you're done using an API, put it out of its misery – destroy it! Simply call *destroy*.

api(arg)

Sets a value for the attribute in the data-model that corresponds to api.

arg
Type: Variant The data to set according to the scheme.

To apply a change to your data model, simply apply the API (as a function) while supplying it the new data as an argument. If the data associated with your API is a JSON – you can submit either the whole JSON or just the part you want to change.

```
1     var setUrgent = function(taskId, urgent) {
2
TaskSchemeAPI.select("Tasks").select(taskId).urgent(urgent);
3     };
```

Note that when submitting a sub JSON, it has to be rooted in the scheme that corresponds to the api at hand.

If the data associated with your API is a primitive - just submit the modified primitive.

```
1 TaskSchemeAPI.select("Tasks").urgent(true);
```

It's important to understand that "neighboring" API instances (that reside on the same machine), will get the update immediately (synchronously), while remote clients will get it (a-synchronously) after a few milliseconds

api()

Fetches an Attribute

To get the current data that is associated with your API, just invoke your API without any parameters.

api(onChange)

api(filter, onChange)

Binds a listener for modifications applied to the data-model that corresponds to api (with or without filtering the result)

onChange(data, options)
Type: Function
A callback method to be invoked when the relevant sub-scheme changes.

data
Type: variant
The sub-scheme data – which may be a primitive or JSON, depending on the sub-da

options
Type: plain object
An option object, that describes the notification's details.

filter
Type: String
Filters the notification types reported to the 'onChange' callback.

Once you apply a change, other APIs on the same machine will get updated immediately, so the change is reflected at once. We call it the "short cycle", because it's syncronius – and therefore short in time (and because it's a cycle – repeated each time an API applies a change).

At the same time, the change is sent to XTRMX server (remember? Starting a session and getting connected when you invoked MX(…) ?). The server gets updated with changes applied by all the users, and creates a new version of your data-model once in ~40 milliseconds. The updated version is then sent to the clients connected to that session. As a result, all the clients' APIs are updated with the most recent changes applied to the data-model. We call this cycle the "long" cycle, since it's a-syncronious, and therefore longer in time (and again since it's a cycle).

In addition, as soon as you subscribe to changes through a given API, it will be notified

regarding the current data-model status. We call this notification the “start” cycle.

If you want to get a notification – start, short or long - about the changes related to your API’s sub-scheme, simply apply the API with an callback function as an argument – it will be invoked whenever the relevant data changes.

This callback function has two parameters:

value: The updated value of your sub-scheme.

options: A data object that describes the origin and metadata of the received change.

When your listener callback is invoked, the value (the first argument) it gets is the new value of that sub-scheme. So even if only a portion of the sub-scheme was modified, your callback will get the entire value of the sub-scheme (that is - not *just* the modified part). See the options object for a full description of what was changed – what was added, removed or updated.

```
1              var checkItemAPI =
checkListAPI1.select(checkItemId);
2
checkItemAPI.completed(function(isCompleted,options){
3                    . . .
4              });
```

Filter: Don’t bother me if you don’t have to

When subscribing to a callback, you often won’t want to get all notifications.
In order to filter notifications and get just the ones you want, simply add your subscription call as a comma-delimited string, before the callback.
The following filtering key-words are supported:
“start”: receive only start start notifications.
“short”: receive only short notifications.
“long”: receive only long notifications.
“old”: receive only old notifications.
“me”: receive only notifications triggered by myself.

You can concatenate multiple options using commas, or add logic-not by a “not” prefix. For example: “short,long,notold” will send only short and long notifications, but not start notifications, and only those which are not old.

```
1 TaskSchemeAPI.select("Tasks").add("notold", function(data,
options, id) {
```

onChange callback Options data

The options are submitted as the second argument of the onChange-callback, and give more details regarding the notification.

cycle
Type: String
States the type of the notification – start/short/long cycles

me
Type: Boolean
Denotes if the change was applied by myself

old
Type: Boolean
Denotes if the change is a long notification that reflects data that was already received in a p
short notification

locked
Type: Boolean
Denotes if the sub-scheme is locked

thisClientIsLockingThisScheme
Type: Boolean
Denotes if this sub-scheme locked by my client

otherClientIsLockingThisScheme
Type:
Denotes if this sub-scheme locked by another client

added
Type: Plain object
Specifies the portion of the data that was added to the sub-scheme

removed
Type: Plain object
Specifies the portion of the data that was removed from the sub-scheme

updated
Type: Plain object
Specifies the portion of the data that was updated

rejected

Type: Plain object
Specifies the portion of data that was submitted by this client but was rejected by the server (mainly due to lock from other client imposed on that sub-scheme)

cycles
The cycle value is an indicator for the origin of the incoming change, and also for the update's purpose. The cycle property optional values are:

Start
When you subscribe to your API (in order to get updated when changes occur), the start-notification is immediately fired with the current data. If no value was yet set to the relevant data, the start notification will fire with the default value assigned to the scheme in its definition.
In order to know whether the value received originated from a start-cycle-notification - inspect if the cycle attribute is equal to "start".

Short
The short cycle notification is sent to APIs that are listening to the same sub-scheme as your API, on the same machine. The aim of this "internal" notification is to update all APIs (and therefore UI) so the change may be reflected immediately. This mechanism allows immediate updates for any API that resides on the same machine that initiated a change.
In order to know whether the value received originated from a short-cycle-notification - inspect if the cycle attribute is equal to "short".

An API that triggered a change will not get the short cycle notification. After all – this change.

To reflect the same data more than once, that is, to have multiple views for a given d attribute, simply create an API per view. Each API will receive a short cycle notificatio when another API will apply a change. So all views will reflect the most updated vers the data at any given time.

```
1         wrapAbsoluteAPI1 =
ModusAPI.select("Assets").selected().select("WrapAbsolute").se
lect("Wrap");
2         wrapAbsoluteAPI2
=ModusAPI.select("Assets").selected().select("WrapAbsolute").s
elect("Wrap");
```

Long
The long cycle notification comes from the MX server, containing the collaborated data for all the users currently connected to the session.

In order to know whether the value received originated from a long-cycle-notification - inspect if the cycle attribute is equal to “long”

All updates coming from other clients will be received as long cycle notifications.

Me

The me attribute indicates that the received change was made by the same API that received it, and that no other API has applied a change to the received data.

A short cycle notification will never have the me attribute set to be true, since a short cycle notification is sent only to other (local) API instances

When a change is submitted through the API to a given attribute in the data-model, the data is sent to the MX server. The server then resolves a new version and notifies all clients (including the client that initiated the change). If a single API has modified that part of the data during that long cycle, then this API alone is responsible for the change – in such a case, the “me” attribute will be true.

Old

Whenever an API is listening to changes on a part of the scheme, and another API on the same machine changes the same scheme part – the listening API will get two change notifications. The first will be a short cycle notification (which will be received immediately). The second will be a long cycle notification, originating from the server, in which the changes are resolved with changes coming from other users.
In such cases, the same data will frequently be received by both notifications (since no other user changed that particular scheme part at the same time). In such cases, the later notification will be marked by the “old” flag.

Locking information

MX API supports “partial locks”. That is, your API can lock the sub-scheme it works with, in order to prevent other remote API from changing the same data. While locking a sub-scheme, the rest of the scheme remains unlocked, so it may be freely changed by other users.

locked

A flag that indicates if the sub-scheme that you’re listening to, is locked (by either the listening API itself or by another API).

thisClientIsLockingThisScheme

A flag that indicates that the part of the scheme associated to the listening API is currently locked by a local API instance (an API that resides on the same machine).

When a sub-scheme is locked by a client, the client is allowed to change it. In other v
implement a lock mechanism that works between APIs on the same machine. Why?

system is mainly intended to be a logic layer that serves a UI-oriented layer in which controls are not manipulated in parallel

otherClientIsLockingThisScheme
A flag that indicates that the part of the scheme associated to the listening API is locked by a remote client (so you can't change until it will be unlocked).

If you attempt to change a sub-scheme which is locked by another client, your chang rejected. You can tell that your change was rejected by inspecting the rejected part c parameters set.

Which data was modified?
The following options let you know how the data was modified: What was added to it, removed from it or previously-existing data that was modified:

added
As you may have guessed, the added part contains the data that was added

updated
This part includes *just the change, that is* – the delta between the previous and the current version. It doesn't include parts that were the same before and after the change.

removed
This part includes parts that were removed from the scheme. In particular, it contains list-items that were removed.

rejected
If you try to modify a sub-scheme which is currently locked by another client, your change will be ignored, and echoed via the options.rejected attribute.

Chaining

Whenever using the API to modify or subscribe to a subset of the data-model, the returned value is the API itself. Therefore, more modifications may be applied via chaining.

Additional map API

In the sections above, the modifications were applied to objects or primitives. But what about lists? How can you add or remove a member to a list? This is the scope of the current section.

listAPI.add(item)

Adds an item to a list

[item = null]
Type: plain object
An item of the list (in the format in which it was defined in the scheme). If no item is provided, an item is generated using the default data of the scheme.

Creates a new item in the list. The added item is of the form that was defined in the provided scheme.

Adding a new item:
In case an empty JSON is submitted to the add method (or in case that no argument was supplied), it will add a new element with the default values detailed in the scheme. If a partial JSON was supplied, the missing values will be replaced by the default values provided in the scheme. The return value of the add method is the added item's unique identifier.

```
1    //add a new task:
2    var newTask = function(title, description) {
3        return TaskSchemeAPI.select("Tasks").add({
4            Title:title,
5            Description:description
6        });
7    };
```

The added item identifier:
Each item in a list has a unique id. The id is automatically added to the item's JSON data (although it is not declared in the scheme) as an '_id' attribute. MX API will generate that

id for you when you add a new item.
If you wish to supply the id yourself for your added item, simply add the '_id' key, and its value, to your added item. In such a case, make sure you don't use existing ids, or the "add" operation will just update the already existing item that has the same id.

listAPI.add([filter], onAdded)

Binds a listener for items added to the list

onAdded(addedItem, options, id)
Type: Function
A callback method to be fired when a new item is added to the list

addedItem
Type: plain object
A list item that was added

options
Type: plain object
options

id
Type: String
The id of the added item

filter
Type: String
filter the types of notifications that will not be ignored by the 'onAdded' callback.

Notification upon add
To get notified when items are added, just call *add* with a callback. It will be triggered whenever items are added to the list. The callback receives three arguments: the data of the newly added item, the options parameters-set change notification options, and the newly added item id.
The callback will be triggered once per cycle per each added list-item.

Filtering
As explained in the filter section, you can submit the add method two parameters – the filters-string and the callback - to get notified only on certain occasions.

```
1            ModusAPI.select("Assets").add("notold",
function(asset, options, id) {
2                cb(id, asset.ImageAssetData.ZIndex);
3            })
```

listAPI.del(id)

Removes an item from the list

Id
Type: String The id of the list item to be removed

Removing an item:
Applying the del method with an item id will remove the item from the list.

```
1          TaskSchemeAPI.select("Tasks").del(function(data, options,
id) {
2              console.log("Task '" + id + "' removed, I know it
from a " + options.cycle + "-cycle notification");
3              if (onTaskRemovedCb) onTaskRemovedCb(data, id);
4          });
```

listAPI.del([filter], onRemoved)

Binds a listener for items removed from the list

onRemove(itemRemoved, options, id)
Type: Function
The callback to be executed when an item will be removed from the list

itemsRemoved
Type: Plain object
The items removed from the list

options
Type: Plain object
options

id
Type: String
The removed item id

Filter
Type: String
Filter the types of notifications sent to the 'onRemoved' callback.

Notification upon remove:
To get notified when items are removed, just call *del* with a callback. It will be triggered whenever items are removed from the list. The callback receives three arguments: the data of the removed items, the options parameters-set Change notification options, and the removed item id.

Filter: As explained in the filter section, you can submit the del method two parameters – the filters-string and the callback - to get notified only on certain occasions.

listAPI.update(json)

changes multiple list-items in a single call

json
Type: Plain object An object literal with list-items ids as keys, and their updated data as the corresponding new value.

In some cases, it is useful to change more than one list item in a given manner. The update method allows submitting a JSON value that updates multiple list items in one go.

Using list.update allows manipulating multiple items in different manners per each item via one API call.

```
ModusAPI.select("Assets").update(newOrder);
```

Updating all the items with public methods

Another alternative to change multiple items in a list is with public methods. Instead of applying the public method on every list item, call the public method on the list itself – this will set all the list items at once.

Using public methods via the listAPI allows changing multiple list-items in the same manner. Manipulating different items in different ways is not supported by this alternative.

```
    //set all task as urgent:
    var everythingUrgent = function() {
        TaskSchemeAPI.select("Tasks").urgent(true);
    };
```

Update and filter

Combine the filter method with updating multiple items to update a sub-collection of list items of your choosing.

listAPI.filter(onFilter)

Filters a set of list items

onFilter(id, item)
Type: Function
A function that describes how the list-item should be filtered when receiving notifications regarding that list.

Id
Type: String
The item's id

item
Type: Plain object
The item's data

The onFilter callback should return true for any item that should not be filtered out, and false for any item that should be filtered.

The filter method allows manipulating or being updated upon changes made to some of the list's items. The *filter* method takes a callback as an argument, and the callback takes two parameters – an item's id and the item's data. The onFilter method returns either true or false, such that true means – "Yes, I care about this item", and false means – "I wish to ignore this item".

Once a list is filtered, a manipulation that will be applied to the list items will apply only for the filtered items set. Similarly, listeners intended to be notified upon changes made to the list items, will fire only for the filtered items set.

The *filter* method returns the previous filter method, so you can restore it whenever you like. This way you can temporarily use a filter, and then reset it when you're done using it.

listAPI.getAsArray(cb, [filter,] [comparator])

Retrieves all the list items as an array

cb(itemsAsArray)
Type: function
A callback method that will receive the resulting array as a parameter.

itemsAsArray
Type: Array
The array of the list items

[filter(id, item)= null]
Type: function
A filter method to filter out unneeded list items, see filter

[compeator(item1, item2) = null]
Type: **function**
This method allows you to receive the resulting array of list items in a particular order. The comperator returns a positive number if item1 is bigger than item2, zero if they're equal and a negative number if item2 is bigger than item1.

The getAsArray method retrieves the list items as a JavaScript array. The first argument of the getAsArray method is a callback function that will receive the items list array as a parameter.

```
1
ModusAPI.select("Assets").getAsArray(function(assetsArray){
2
ModusAPI.select("Assets").selected(assetsArray[0]._id);
3               },
4               null,
5               function(asset1, asset2){
6                   return asset1.ImageAssetData.ZIndex <
asset2.ImageAssetData.ZIndex
7               });
```

There are a few bonuses that come along with the getAsArray method:

Filter:
In addition to the "getting a JavaScript array" functionality, a filtering method may also be submitted. The filtering method will filter the data based on the criteria it implements. The filter method behaves as described in filter method, only it's temporary. As soon as you get the array, the previous filter is restored.

Sort

The sort parameter is a callback function that allows you to get the list items output array in a particular order. The sort function serves as a comparator, that is, it takes two list-items, and returns a negative number if the first item is smaller than the second, zero if they're equal, and a positive number otherwise.

```
1               function(asset1, asset2){
2                   return asset1.ImageAssetData.ZIndex <
asset2.ImageAssetData.ZIndex
3               });
```

Selected list item

When using lists, “marking“ or “selecting” a list item is a common need. The selected method aims to enable selecting an asset in a collaborative manner.

listAPI.selected(id)

Sets the selected id in the list.

Id
Type: String The selected item id

To set the selected id, submit the list-item id to the list.selected method

```
1                    ModusAPI.select("Assets").selected(arg);
```

listAPI.selected()

Returns the api of the selected id

Get the selected id

If you want to know which id is currently selected, simply call the list.selected methods without any parameters to get the selected item’s API, and use the id method to get that item’s id.

Modify the selected id

??? this needs fixing

To modify the list item currently marked as the selected item, simply call the list.selected methods without any parameters to get the selected item’s API. Once the selected item is changed, the selected item API will automatically refer the newly-selected item.

Needless to say, you can use select or public-method on the selected API to get the appropriate sub-scheme API on the selected item.

Listen to modifications on the selected id

Once you have the selected item API, apply the API with a function as an argument to bind a supplied callback function to trigger whenever the selected item changes. The callback method will initially trigger with a start notification – indicating the newly selected list item, and from that point on - short and long notifications will be triggered regularly to report changes regarding the selected item.

listAPI.selected(onSelected)

Binds a listener that fires when the selected item in the list is changed.

onSelected(id)
Type: Function
The callback to be executed when a selection is changed.

Id
Type: String
The id of the newly selected item

listAPI.getIds()

Returns the ids of all the items within the list

Which items-ids does your list contain? Just call list.getIds() and you'll know

listAPI.howMany()

Returns how many items are within the list.

```
1     //get the task progress: call cb with two arguments:
how many completed items, and how many items in total
 2     var getProgress = function(checkListAPI, cb) {
 3         //get array of completed-only check-list items:
 4         var total = checkListAPI.howMany();
 5         checkListAPI.getAsArray(function
(completedCheckItems) {
 6                 cb(completedCheckItems.length,
total);//how many compelted vs. how many in total
 7             },
 8             //filter only the completed items
 9             function (id, checkListItem) {
10                 return checkListItem.CheckItem.Completed;
11             });
12     };
```

Locking: This is MINE!

Often, different clients wish to change the same part of the scheme. In these cases, it's

useful to lock the sub-scheme while it's being changed (to avoid contradictions).
The API allows activating a lock on a sub-scheme, such that while a sub-scheme is locked, other clients cannot change it. On the other hand, other client will be free to apply changes to other (unlocked) parts of the data-model.

Other APIs in your client would be able to change a sub-scheme locked by your client, since we do not expect a single user to change multiple attributes (or the same attribute in multiple manners) simultaneously.

While a client is locking a sub-scheme, you'd typically want to ignore notifications that regard it, so your view will be updated exclusively by your modifications.

lock(lockOrUnlock, disableNotifications)

Lock or unlock the API's corresponding sub-scheme

[lockOrUnlock = true]
Type: Boolean
true to lock, false to unlock

[disableNotifications = true]
Type: Boolean
When the disableNotifications flag is on, no notifications will be sent while the API is locked.

locking/unlocking
To lock/unlock a sub-scheme by client, use the lock method with 'true' or 'false' to lock or unlock respectively.

lock in use
If another client is already locking the sub-scheme in discussion, the new locking request will be rejected, and the sub-scheme will remain locked. To know a locking request was rejected use reject-notification]

Disable notification
Often, you don't want to get notifications concerning a sub-scheme while it's locked.

For example, consider a slider that the user slides, in order to change some collaborated data. Naturally, you want to lock that sub-scheme as soon as the user clicks on the slider, and unlock it when the user releases the slider. While the slider is locked, you would want to provide the user a smooth sliding experience. Yet, data received from the long cycle long cycle during sliding (that reports old values - from ~40 msec ago), keeps flowing in. These old values should be ignored, or else the slider will be alternatively set to new values – by the UI, and old values - by the old notifications, causing the slider knob to

jump around.

For this reason, the lock-method also disables the notifications to the locking API. When a sub-scheme is unlocked, the API will resume the receival of notifications. The first notification will include the update state since the locked-state change.

lock(onLocked)

Binds a listener that fires when the API becomes locked or unlocked

onLocked(isLocked)
Type: function
Fired whenever the lock changes

isLocked
Type: Boolean
Is the current item locked on another client

To receive notifications even while an API is locked, simply submit 'false' in the second parameter of the lock method.