XirSys WebRTC Client (Alpha) Documentation

Version: 1.2.0 (Alpha)
Author: Lee Sylvester
Date: 3rd June, 2013

Overview

Introduction

The XirSys WebRTC Client Framework provides a flexible middle level JavaScript API for the purpose of constructing RTC HTML5 applications. The framework was designed to ease the development of WebRTC software while remaining flexible enough to facilitate almost any application. XirSys provides the client framework as open source, for use with or without the XirSys suite of services.

Features

- Optional use authentication handling with XirSys servers
- Optional use XirSys realtime messaging protocol
- Full video and audio handling
- Screensharing and Data channel implementation
- Rooms based participant grouping
- User querying and messaging
- Powerful events management
- Audio and video muting / unmuting
- Forceable TURN usage
- Full logging and error handling

The functionality provided by the XirSys platform is increasing and new features will be added when available.

Client Components

The client framework is built up of several independent components. Each component provides a subset of functionality, similar to a class in other languages. The following table briefly describes each component and their roles within the framework, excepting some of the more obvious internal components.

Component	Description
AuthManager	The AuthManager component is the layer between the client functionality and the services and TURN server use provided by XirSys. AuthManager was written with the idea that it should be easily extendable or replaceable should the developer wish to swap out the XirSys services for their own.
Connection	The Connection component provides an interface to WebRTC TURN, STUN and handshaking. It is analogous of the XirSys messaging protocol and should therefore remain unchanged. The

	Connection component provides hooks for instigating the various WebRTC data connections as well as events to enable the developers application to respond to changes in state of users and connections.
DataChannel	The DataChannel component implements functionality specific to data channels within WebRTC. Data channels are not required for video and audio connections, so its functionality has been provided within a separate component, both for flexibility and for complete removal should data channels not be required within the end user application.
EventDispatcher	The EventDispatcher provides common event management within the XirSys client. Developers can also make use of the EventDispatcher component in their own applications for their own custom events.
HandshakeController	The HandshakeController component provides event handlers for facilitating the requests between two participants in order to set up a WebRTC connection. The functionality should not need to be modified by the developer.
Room	The Room component provides functionality to allow users to join or leave a specific room. Users can also query rooms for information, such as its name and who else is participating in the room.
ServerConnector	The ServerConnector component provides low level handling of XirSys specific messaging and should not be interacted with directly by the developer.
Stream	The Stream component allows for direct manipulation of a stream. Currently, this includes muting and unmuting video and audio streams and attaching those streams to HTML elements. Eventually, however, this functionality will be extended to support other, more extensive, functionality.

Figure 1 outlines the API and the relationships between the components.

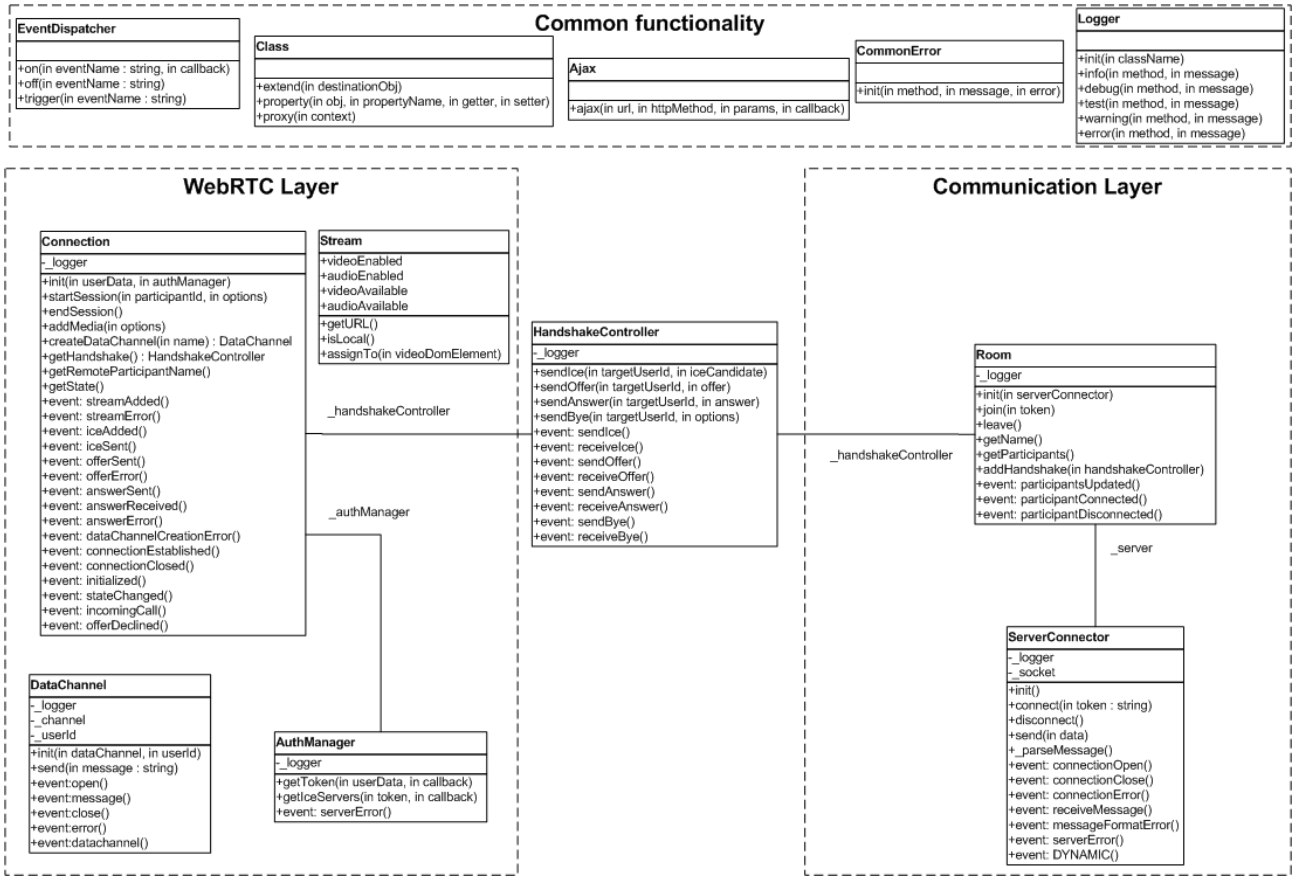

Fig. 1

Getting Started

A Word About Authenticating

Before a WebRTC connection can happen, an applications end user must first authenticate. XirSys doesn't store information about end users; only the owner of the XirSys account. Therefore, to authenticate, users must make their request through the XirSys application owners own services. Once the user has been verified, those services then make a server to server call to a XirSys server to request a token or data before passing that data back to the end user.

XirSys currently provides two interfaces for authenticating; one for websocket authentication and one for TURN authentication.

Authentication with a XirSys Server

XirSys application owners authenticate with the XirSys servers using a secret key. This key is separate from the password used to login to the XirSys portal and was designed to be used from the client in secure environments or when the authentication of end users is not applicable. However, for the most part, the secret key should always be used from a server environment. The secret key can be regenerated if compromised.

Authentication requests require a number of parameters in order to succeed. They include:

- **domain** - the domain of the application. This is specific to a XirSys users account.
- **application** - the name of the application. This is specific to a XirSys users account, though accounts may have more than one application per domain.
- **room** - the name of a room to assign a user. The room must already exist within an application.
- **username** - the name of the user authenticating.
- **ident** - the name of the XirSys application owner.
- **secret** - the secret of the XirSys application owner.

Generating a Websocket Token

Websocket tokens ensure that websocket data usage is restricted to application owners, only. The tokens themselves are encrypted data which is expected by the websocket endpoints and required to setup a connection correctly. Invalid tokens or incomplete data within tokens would result in an unsuccessful websocket connection.

The following is an example PHP script showing how a XirSys users application might request a token from a XirSys server.

```php
<?php

//set POST variables
$url = 'http://endpoint.xirsys.com/getToken';
$fields_string = '';
$fields = array(
        'domain' => $_POST["domain"],
        'application' => $_POST["application"],
        'room' => $_POST["room"],
        'username' => $_POST["username"],
        'ident' => urlencode("xirsys_user"),
        'secret' => urlencode("12345")
);

//url-ify the data for the POST
foreach($fields as $key=>$value) { $fields_string .= $key.'='.$value.'&'; }
rtrim($fields_string, '&');

//open connection
$ch = curl_init();

//set the url, number of POST vars, POST data
curl_setopt($ch,CURLOPT_URL, $url);
curl_setopt($ch,CURLOPT_POST, count($fields));
curl_setopt($ch,CURLOPT_POSTFIELDS, $fields_string);
```

```
//execute post
$result = curl_exec($ch);

//close connection
curl_close($ch);
?>
```

Authenticating with a TURN Server

Authenticating with a TURN server follows a similar path to acquiring a websocket token, with the major difference being that, instead of returning a token, the request returns a WebRTC STUN and TURN connection string.

In WebRTC, STUN and TURN connection strings expose the username and password of the connecting user within the client code. XirSys protects its registrants by generating UUID's for authenticated users for both username and password within the resulting connection string. These credentials then expire within a given time of disconnecting, providing the user does not reconnect within a given period. This ensures that premium usage is only attributed to the correct users account.

The following is an example PHP script showing how a XirSys users application might request the STUN / TURN credentials from a XirSys server. Notice that the room parameter is not necessary for STUN / TURN authentication.

```
<?php

//set POST variables
$url = 'http://endpoint.xirsys.com/getIceServers';
$fields_string = '';
$fields = array(
        'domain' => $_POST["domain"],
        'application' => $_POST["application"],
        'username' => $_POST["username"],
        'ident' => urlencode("xirsys_user"),
        'secret' => urlencode("12345")
);

//url-ify the data for the POST
foreach($fields as $key=>$value) { $fields_string .= $key.'='.$value.'&'; }
rtrim($fields_string, '&');

//open connection
$ch = curl_init();

//set the url, number of POST vars, POST data
curl_setopt($ch,CURLOPT_URL, $url);
curl_setopt($ch,CURLOPT_POST, count($fields));
```

```
curl_setopt($ch,CURLOPT_POSTFIELDS, $fields_string);

//execute post
$result = curl_exec($ch);

//close connection
curl_close($ch);

?>
```

Creating a XirSys Connection

There are a few steps required to set up a user and connect them to the XirSys services. These involve.

- Authenticating the user
- Attaching the user to a WebRTC stream, such as video, audio, data or combinations thereof.
- Joining the user to a room

Authenticating / Initialising

Authentication handling is performed via the AuthManager, but is bridged to WebRTC using the Connection component. Both components are required to implement authentication.

```
userData ={
        domain: "www.example.com",
        application: "myApp",
        room: "myRoom",
        name: "someUser",
        password: "pass"
};
var authManager = new xRtc.AuthManager();
connection = new xRtc.Connection( userData, authManager );
```

The actual calls to the server are transparent, so the above implementation should be all that is required to set up a request for data from the server. However, in order physically create a connection, the user must request the token and join a room.

Configuring Streams

There are two ways to configure streams, depending on whether you wish to attach to an audio and/or video stream or to a data stream.

Configuring audio and video streams makes use of the *Connection.addMedia* function. This function accepts a single object parameter with the boolean values *video* and *audio*. Setting

either of these values to false mutes that particular stream. Omitting the object parameter defaults to both streams being enabled.

```
connection.addMedia( { video: true, audio: true } );
```

Note that both the video and audio streams can be muted and unmuted mid-stream.

Data channels are established using the *Connection.createDataChannel* function. This function also accepts a single parameter, which is a string value representing a unique handler for the stream.

```
connection.createDataChannel( "myChannel" );
```

The function returns a DataChannel object instance.

Joining a Room

Joining a room requires use of the Room and ServerConnector components. The ServerConnector instance ensures the Room object has access to make requests to the XirSys services.

```
serverConnector = new xRtc.ServerConnector();
room = new xRtc.Room( serverConnector );
room.addHandshake( connection.getHandshake() );
```

This sets up the necessary Room object. The *handshake* method helps ensure that the Connection object and Room objects share the same HandshakeController instance. However, to actually join a room, you first need the websocket authentication token.

```
authManager.getToken( userData, function( token ) {
        room.join( token );
});
```

The userData passed in the call to getToken is the same data used to instantiate the Connection object. This parameter may be deprecated in the next version of the API.

Establising a Call

Now your client application is configured, it's time to establish a connection between two users by starting a session, which for the most part is a simple process.

```
options = { connectionType: "server" };
connection.startSession( "someOtherUser", options );
```

The first parameter to *Connection.startSession* is the id or username of the person to establish a connection to. The options parameter provides additional configuration of the session, which currently only supports the connectionType value. The connectionType can be either “default”, “direct” which attempts to force a peer-to-peer connection or “server” which attempts to force a TURN based connection.

Once called, each client - both the calling client and the recipient - will continue to discreetly exchange messages in order to establish the connection. Once all such necessary messages have been exchanged, the Connection objects *streamAdded* event will fire (see “Events” later in this document). It is here that the associated stream object should be attached to the web page video instance.

```
connection.on( xrtc.Connection.events.streamAdded, function( data ) {
        var stream = data.stream.getStream();
        stream.onended = function( evt ) {
                removeVideoById( evt.srcElement.id );
        };
        data.stream.assignTo( myVideoTag );
} );
```

Here, we are doing two things; the first is to ensure that the stream is removed from the video tag once it has ended. The second is to attach the current stream to the video tag. Once this has executed, both audio and video should be displayed to the user.

Events

Events play a core part of the XirSys WebRTC Client Framework. The EventDispatcher component makes both handling events and dispatching events simple. Any component extending the EventDispatcher is able to dispatch events, including custom events.

The EventDispatcher is extended by the following components:

- Connection
- DataChannel
- HandshakeController
- Room
- ServerConnector

See the API section of this document for a detailed list of the events served by these components.

Handling Events

Events are handled by passing the name of the event to handle, as well as a callback handler, to

the component dispatching the event using it's *on* function.

```
someHandler = function() {
        console.log( "event called" );
};
connection.on( "someEvent", someHandler );
```

Event handlers may be passed data in the form of one or more parameters (see "Dispatching Events").

Dispatching Events

Events are dispatched using a components *trigger* function.

```
connection.trigger( "someEvent" );
```

Data can be passed to the invoked handlers by passing one or more additional parameters to the trigger function. Each additional parameter then becomes a parameter of the invoked handler. For example.

```
someHandler = function( arg1, arg2 ) {
        console.log( arg1 + ", " + arg2 );
};
connection.on( "someEvent", someHandler );
connection.trigger( "someEvent", "Hello", "World" );
```

Client API

The following pages outline each of the primary components and details their public functions, properties and events.

AuthManager

The AuthManager component is the layer between the client functionality and the services and TURN server use provided by XirSys. AuthManager was written with the idea that it should be easily extendable or replaceable should the developer wish to swap out the XirSys services for their own.

Properties

Name	Type	Description
settings.tokenHandler	Static	URL for server-side token request handler. This should be updated to that of the developers server-side token request service. See *Getting Started -> Authentication*, above.
settings.iceHandler	Static	URL for server-side STUN and TURN credentials request handler. This should be updated to that of the developers server-side token request service. See *Getting Started -> Authentication*, above.
settings.URL	Static	Base services URL. If using XirSys services and messaging layer, this need not be changed.

Functions

Name	Params	Type	Description
getToken	userData : Object, callback : Function	Instance	Authenticates the user against the application owners custom token handler. Once data is returned, the callback function is invoked with the returned token value.
getIceServers	token : String, callback : Function	Instance	Authenticates the user against the application owners custom ICE handler. Once data is returned, the callback function is invoked with the returned ICE credential data.

Events

Name	Params	Description
serverError	message : String	This event is raised if either the getToken or getIceServers event requests result in an error from the authentication services. The event is passed a string message describing the error.

Connection

The Connection component provides an interface to WebRTC TURN, STUN and handshaking. It is analogous of the XirSys messaging protocol and should therefore remain unchanged. The Connection component provides hooks for instigating the various WebRTC data connections as well as events to enable the developers application to respond to changes in state of users and connections.

Properties

Name	Type	Description
settings.offerOptions	Static	Allows for the inclusion of Media Constraints when creating and offer for connection. (See http://datatracker.ietf.org/doc/draft-burnett-rtcweb-constraints-registry/ for options)
settings.answerOptions	Static	Allows for the inclusion of Media Constraints when creating and answer for a connection offer. (See http://datatracker.ietf.org/doc/draft-burnett-rtcweb-constraints-registry/ for options)

Functions

Name	Params	Type	Description
init	userData : Object, am : AuthManager	Instance	Connection component constructor.
startSession	participant : String, options : Object	Instance	Attempts to create a WebRTC connection to “participant” using the connectionType specified in “options”. If “options” is omitted, then the connectionType value is set to “default”.
endSession		Instance	Ends the currently connected session.
addMedia	options : Object	Instance	Acquires use of local devices, such as video and microphone. Options is an optional object with boolean values for video and audio. {video: true, audio: true}
createDataChannel	name : String	Instance	Opens a data channel on the current session using a unique name handler.
getHandshake		Instance	Returns the HandshakeController instance used by the Connection instance.
getRemoteParticipantName		Instance	Returns the name / id of the remote connected user.
getState		Instance	Returns the state of the current P2P connection.

Events

Name	Params	Description
streamAdded	streamData : Object	Called when a stream has successfully been created between participating connections. “streamData” contains a reference to the Stream instance, as well as the name of the remote participating user. {stream : Stream, participantId : String}
streamError	error : CommonError	Raised when failing to create a stream.
iceAdded	iceData : Object, iceCandidate : Object	Raised when ICE data is received from recipient or ICE server.
iceSent	iceCandidates : String	Raised when ICE data is sent to remote participant.
offerSent	remoteParticipant : String, request : Object	Raised when offer data is sent to remote participant.
offerError	error : CommonError	Raised when failing to create offer for remote participant.
answerSent	offerData : Object, answer : String	Raised when answer data is sent to remote participant.
answerReceived	answerData : Object, sessionDesc : Object	Raised when answer received from remote participant.
answerError	error : CommonError	Raised when failing to create answer for remote participant.
dataChannelCreationError		Reserved for future use
connectionEstablished	remoteParticipant : String	Raised on successful connection to remote participant.
connectionClosed	remoteParticipant : String	Raised on close of connection to remote participant.
initialized		Raised when data established between peers, but prior to connection attempt.
stateChanged	currentState : String	Raised when connection state has changed.
incomingCall	data : Object	Raised when remote participant requests connection to user. Passed data contains functions to invoke in order to accept or decline the call. { participantName : String, accept : Function, decline : Function }
offerDeclined	remoteParticipant : String	Raised when remote participant declines offer for connection.

DataChannel

The DataChannel component implements functionality specific to data channels within WebRTC. Data channels are not required for video and audio connections, so its functionality has been provided within a separate component, both for flexibility and for complete removal should data channels not be required within the end user application.

Functions

Name	Params	Type	Description
send	message : String	Instance	Sends string message to remote participant.

Events

Name	Params	Description
open	data : Object	Raised when DataChannel opens. Passed data is the event object from the WebRTC DataChannel open method.
message	data : Object	Raised when message is received from remote participant. Passed data object is of the form: {event: WebRTCEvent, message: String}
close	data : Object	Raised when DataChannel closes. Passed data is the event object from the WebRTC DataChannel close method.
error	error : CommonError	Raised when DataChannel errors.
dataChannel		Raised when DataChannel initialised.

EventDispatcher

The EventDispatcher provides common event management within the XirSys client. Developers can also make use of the EventDispatcher component in their own applications for their own custom events.

Functions

Name	Params	Type	Description
on	eventName : String, callback : Function	Instance	Assigns the *callback* function as handler to the event *eventName*
off	eventName : String	Instance	Removes all handlers for event *eventName*.
trigger	eventName : String	Instance	Raises event *eventName*. Additional arguments can be passed which will be forward to all invoked handlers.

HandshakeController

The HandshakeController component provides event handlers for facilitating the requests between two participants in order to set up a WebRTC connection. The functionality should not need to be modified by the developer.

Events

Name	Params	Description
sendIce	request : Object	Raised when sending ICE data to remote participant
receiveIce	request : Object	Raised when receiving ICE data from remote participant
sendOffer	request : Object	Raised when sending offer data to remote participant
receiveOffer	request : Object	Raised when receiving offer data from remote participant
sendAnswer	request : Object	Raised when sending answer data to remote participant
receiveAnswer	request : Object	Raised when receiving answer data from remote participant
sendBye	request : Object	Raised when sending a close request to remote participant
receiveBye	request : Object	Raised when receiving a close request from remote participant

Room

The Room component provides functionality to allow users to join or leave a specific room. Users can also query rooms for information, such as its name and who else is participating in the room.

Functions

Name	Params	Type	Description
join	token : String	Instance	Joins the room specified in the passed token request.
leave		Instance	Leaves the current room.
addHandshake	hc : HandshakeController	Instance	Assigns a handshake controller to the room.
getName		Instance	Returns the name of the room.
getParticipants		Instance	Returns a list of the rooms current participants.

Events

Name	Params	Description
join		Raised when user has joined a room.
leave		Raised when user has left a room.
participantsUpdated		Raised when rooms participant list has updated.
participantConnected		Raised when a new participant has entered the room.
participantDisconnected		Raised when a participant has left the room.
tokenExpired		Raised when attempting to join a room with an expired token.

Stream

The Stream component allows for direct manipulation of a stream. Currently, this includes muting and unmuting video and audio streams and attaching those streams to HTML elements. Eventually, however, this functionality will be extended to support other, more extensive, functionality.

Functions

Name	Params	Type	Description
getStream		Instance	Returns the raw stream object.
getURL		Instance	Returns the URL of the stream. Useful for manually assigning as the source of an HTML element.
isLocal		Instance	Returns true if the stream connection is from a local source (not from a remote participant).
assignTo	video : DomElement	Instance	Assigns current stream to a video DOM element.