CTI
"DIOPHANTUS"
computer
technology
institute & press

MINISTRY OF EDUCATION, LIFELONG LEARNING AND RELIGIOUS AFFAIRS

Patras

Head Offices: N. Kazatzaki str.,
GR-26 504 Patras, Greece, PO BOX 1382
Tel.: +30 2610 960 200
Fax: +30 2610 960 490

Athens

10 Davaki str. Ampelokipi,
GR-115 26 Athens, Greece
Tel.: +30 210 69 30 700
Fax: +30 210 69 30 750

VAT No: 090060693

www.cti.gr

Dear Working Group Members,

I send you this letter on behalf of Computer Technology Institute and Press (CTI) "Diophantus" (http://www.cti.gr/) a member of "The Community Network Game" (CNG) project's consortium (http://www.cng-project.eu/). CNG is a research and development project funded by European Commission under FP7/ICT programme.

During our research and development work in CNG, one of the activities we are responsible for is the identification of possible contribution to standards. Given that our work also included research and development activities on Web Technologies we have the following interesting idea that may contribute to your standardization processes within W3C.

Our idea is the support of Real-Time Messaging Protocol (RTMP) in the future JavaScript standard. RTMP is a protocol developed by Adobe for the support of a higher-level multimedia stream. Its current specification is available at the following link: http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/rtmp/pdf/rtmp_specification_1.0.pdf

Since the earlier versions of JavaScript did not include features for video/audio capturing/playback, flash objects were widely used for the implementation of web-based applications for video-on-demand and video/audio conferencing. These objects covered both media capturing/playback and interaction with the multimedia server. RTMP was developed to support the networking part of these applications and it was widely accepted. It requires the existence of an RTMP server and for this reason many RTMP server's implementations were launched not only from Adobe but also from other developers. An indicative list of the current RTMP server's implementation can be found at the following link: http://www.flashrealtime.com/list-of-available-rtmp-servers/. Among others there are several open-source implementations of RTMP server (e.g., Red5).

The reason we believe that JavaScript should support RTMP protocol is that RTMP has become very popular and many multimedia services have been based on it. Therefore in order to make a smooth migration from flash-based to non-flash-based chat applications we should secure a backward compatibility with the existing services, which are mainly based on RTMP. This kind of backward compatibility towards the legacy services will increase the acceptance of non-flash-based applications both within the users and within the chat service providers. Therefore the support of RTMP will be widely exploited by the future multimedia applications that rely on JavaScript.

More information on the necessary methods/commands as well as the related technical references can be found in the Annex below.

I am available to provide you any further information on this.

Sincerely,

Christos J. Bouras, Professor
Department of Computer Engineering and Informatics, University of Patras
and Computer Technology Institute & Press «Diophantus»

ANNEX: JavaScript for RTMP Support

This document outlines to Real Time Messaging Protocol (RTMP) [1] implementation as a JavaScript API. It should be noted that streams are captured according to Media capture API described in [2].

NetConnection

- **connect**. The client sends the connect command to the server to request connection to a server application instance. Parameter: URL of the server having the following format: *protocol://servername:port/appName/appInstance*
- **call**. The call method of the NetConnection object runs Remote Procedure Calls (RPC) at the receiving end. The called RPC name is passed as a parameter to the call command.
- **createStream**. The client sends this command to the server to create a logical channel for message communication .The publishing of audio, video, and metadata is carried out over stream channel created using the *createStream* command.

NetStream

- **play**. The client sends this command to the server to play a stream. A playlist can also be created using this command multiple times. Parameters:
  - *streamName*, name of the stream to play.
  - *start*, an optional parameter that specifies the start time in seconds.
  - *duration*, an optional parameter that specifies the duration of playback in seconds.
  - *reset*, an optional Boolean value or number that specifies whether to flush any previous playlist.
- **play2**. Unlike the play command, play2 can switch to a different bit rate stream without changing the timeline of the content played. The server maintains multiple files for all supported bitrates that the client can request in play2. Parameters:
  - *startTime*, an encoded object that stores a number value. The value in this field specifies the beginning position of the stream, in seconds. If 0 is passed in the Start Time field, the stream is played from the current timeline.
  - *oldStreamName*, an encoded object that stores a string value. Its value is a string containing the stream name parameter and the old stream name.
  - *streamName*, an encoded object that stores a string value. It stores the name of the stream that is played.
  - *duration*, an encoded object that stores a number value. The value stored in it specifies the total duration of playing the stream.
  - *transition*, an encoded object that stores a string value. Its value defines the playlist transition mode (switch or swap mode) switch: Performs multi-bitrate streaming by switching 1-bit rate version of a stream to another swap: Replaces the value in *oldStreamName* with the value in *streamName*, and stores the remaining playlist queue as is. However, in this case, the server does not make any assumptions about the content of the streams and treats them like different content. Hence, it either switches at the stream boundary or never.
- **deleteStream**. NetStream sends the deleteStream command when the NetStream object is getting destroyed. Parameter:

- *streamID,* ID of the stream that is destroyed on the server.

- **receiveAudio**. NetStream sends the receiveAudio message to inform the server whether to send or not to send the audio to the client. Parameter:
  - *receiveFlag*, true or false to indicate whether to receive audio or not.
- **receiveVideo**. NetStream sends the receiveVideo message to inform the server whether to send the video to the client or not. Parameter:
  - *receiveFlag*, true or false to indicate whether to receive video or not.
- **Publish**. the client sends the publish command to publish a named stream to the server. Using this name, any client can play this stream and receive the published audio, video, and data messages. Parameters:
  - *publishingName*, name with which the stream is published.
  - *publishingType*, type of publishing. Set to “live”, “record”, or “append”.
    - record: the stream is published and the data is recorded to a new file. The file is stored on the server in a subdirectory within the directory that contains the server application. If the file already exists, it is overwritten.
    - append: the stream is published and the data is appended to a file. If no file is found, it is created.
    - live: live data is published without recording it in a file.
- **Seek**. The client sends the seek command to seek the offset (in milliseconds) within a media file or playlist. Parameter:
  - *msecs*, number of milliseconds to seek into the playlist.
- **Pause**. The client sends the pause command to tell the server to pause or start playing. Parameter:
  - *pause/unpause*, true or false, to indicate pausing or resuming play.

References

[1] RTMP specification (online): http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/rtmp/pdf/rtmp_specification_1.0.pdf

[2] Media capture API (online): http://www.w3.org/TR/mediacapture-streams/