AT&T MOBILE VIDEO MODULE DESIGN

The **Mobile Video Module (MVM)** is a software code module that enables playing a video invoked from a link on an AT&T website page in a browser on a mobile device. The MVM is intended to be used with other AT&T video playback modules and functions.

For a video to play correctly to completion using the MVM, the video must: be correctly encoded; be stored in its appropriate location on the server; and, have a correct path to that location. The correct path must be submitted to the MVM by the caller of the MVM.

If the above requirements are not fulfilled, one or more errors will be triggered in the MVM and the MVM will not play a video correctly to completion. The MVM will attempt to gracefully recover from such errors, but cannot play the video.

The MVM must also provide a certain amount of statistical reporting information. This information is generated by receiving and processing certain events during the playback process.

MOBILE DEVICES AND THE MVM

The types of mobile devices for which the MVM is intended are phones and tablets. A tablet is an electronic device that is smaller than traditional desktop and laptop computers and larger than phones. A tablet has a screen incorporated into its body. A phone is an electronic device with a physical form that lends itself to voice and other communication via wired or wireless networks. A phone is smaller than computers and tablets. A phone has a screen incorporated into its body. The screen area may be between two and ten square inches.

Mobile devices are controlled by a variety of operating systems. There are dozens, perhaps hundreds of such operating systems; however, at any given time, only a few are widely used. Which operating systems, and which versions of those operating systems are most widely used, varies from month to month. Hence, it is imperative that the MVM be general enough to respond to this shifting group of devices without constant code modification.

MOBILE DEVICE OPERATING SYSTEMS AND THE MVM

The MVM is intended to work with certain mobile device operating systems. The MVM may need to be modified as more is learned about particular operating systems and devices. The operating systems, listed below by usage, account for about 90 percent or more of the mobile devices in use as of Q2 2012:

Google Android
Apple iOS
Nokia Symbian

Research in Motion BlackBerry
Samsung Bada and Tizen
Microsoft Windows Phone and Windows Mobile

All of the more recent versions of the above operating systems support the HTML5 video element. With the exception of Android, all that is needed in an HTML5 video element are three source elements; one each for MP4, Ogg, and WebM videos. For Android, a source element for MP4 must be provided without a type attribute and a click listener must be explicitly attached to the video element.

All of the more recent versions of the above operating systems support the video properties and events specified for the HTML5 video element.

The properties are:
autoplay, controls, currentSrc, currentTime, defaultPlaybackRate, duration, ended, error, muted, networkState, paused, playbackRate, played, preload, readyState, seekable, src, startTime, volume.

The events are:
abort, canplay, canplaythrough, durationchange, emptied, ended, error, loadeddata, loadedmetadata, loadstart, pause, play, playing, progress, ratechange, seeked, stalled, suspend, timeupdate, volumechange, waiting.

Symbian Browser 7.4.2 supports the HTML5 video element and H.264. It does not support the autoplay, buffered, duration, loop, seekable, seeking, videowidth, and videoheight attributes for video.

As of 3Q 2010 WebKit and the new W3C standard for HTML5 are available on BlackBerry.

Tizen supports the HTML5 video element. Tizen is a Linux-based open-source operating system and software platform backed by Samsung, Intel and other vendors. Samsung plans to phase out its homegrown smartphone platform, Bada, and replace it with Tizen. For the remainder of 2012, Samsung is focused on the development of Android devices, as well as on smartphones running Microsoft's upcoming Windows Phone 8.

Tizen supports the HTML5 video element. Tizen is a Linux-based open-source operating system and software platform backed by Samsung, Intel and other vendors. The software is intended for use in devices such as smartphones, tablets, netbooks, in-vehicle infotainment devices, and smart TVs.

Because the same implementation of Internet Explorer is being used on Microsoft desktop and on mobile devices, the HTML5 implementation is the same on desktop and mobile. For video, this means three things:

1. Same markup: the same <video> HTML markup will run on bothIE9 desktop as well as

on IE9 Mobile, i.e. the embedded video will play without any modification of your code.

2. Same format: IE9 on Mango will play HTML5 video in H.264, the most widely used video format on the mobile web.

3. Same hardware acceleration: On Mango IE9, video playback is accelerated by the GPU on the device, the same way desktop IE9 video performance leverages the GPU.

Windows Phone 8 is based on Windows 8 for the desktop. WP8 provides for and encourages vector graphics. Handset makers will have more freedom in designing displays because it won't matter. Things will work correctly on any size, as on the desktop. Compatibility with Windows goes down to the device driver layer. There is a single hardware driver model across all Windows platforms.

Windows Phone 7 acted as a bridge between earlier Windows Phone versions and Windows Phone 8. Windows Mobile is essentially gone.

Android, iOS, Symbian, RIM, Bada > Tinza, Windows Phone and Windows Phone and Mobile. The operating systems are listed in order of total devices sold as of January of 2012. As with other electronic devices, the amount of usage of an operating system does not always align with this metric. However, in that respect, Android and iOS clearly dominate at this time.

MVM PROGRAMMING REQUIREMENTS

Based on the above information and assumptions, the MVM must:
- play a video in a mobile device's screen with or without a modal dialog.
- work with the communciation interfaces of the above mentioned operating systems.
- work with the user interfaces of the above mentioned operating systems.
- be responsive to gestural controls as well as button and key presses.
- provide statistical reporting information.
- as much as possible, work with the communciation and user interfaces of other operating systems and future operating systems.
- as much as possible, work with user interfaces of downloadable non-native browsers.

To accomplish the above requirements, the MVM must be able to:
- determine if the current device is a mobile device.
- determine if the current device has a screen large enough to display a modal dialog.
- play a video without a modal dialog.
- play a video in a modal dialog.
- provide statistical reporting data related to the played video.

MVM PROGRAM DESIGN ISSUES

The MVM is a JavaScript object containing functions and other code related to playing videos, which are invoked from links on pages of the AT&T website, on mobile devices. Within the MVM object are members for:
- determining if a device is a mobile device or not.
- determining a device has a screen large enough to display a modal dialog.
- playing a video without a modal dialog.
- playing a video within a modal dialog.
- providing statistical reporting data related to the played video.

Below are sections that explain the issues the MVM members must address.

Mobile Device Determination
Determining if a device is a mobile device is based certain information extracted from the user agent string. Using such information as the device's type, brand or model, a device is classified as mobile or not.

For example, if the user agent string contains the word "iPhone" or "iPad" or "iPod", the device is identified as a device running iOS. Devices running iOS are treated as mobile devices.

A mobile device may, or may not, have a screen size large enough to accommodate the modal dialog normally used on the AT&T website for playing videos in standard browsers on desktop and laptop computers. Because a mobile device may be a phone or a tablet, the screen size of a device cannot be reliably inferred from the fact that it is a mobile device. Hence, the screen size must be determined independently from the mobile device determination.

Screen Size Determination
The physical size of a mobile device's screen limits how large the viewport can be for video playback. Below a certain screen size, the viewport dimensions must match the dimensions of the screen or the video image is too small for a user to view it. Hence, for mobile devices of a certain size, e.g., the size of an iPhone, the native video player forces videos to played full screen.

This playback strategy conflicts with the way AT&T videos are played, i.e., in a modal dialog that is substantially larger than a mobile phone's screen size. Therefore, it is necessary to determine the physical size mobile device's screen and, if it is below a certain size, play the video using the native player instead of the modal dialog. Hence, when the player is invoked, the physical size of a mobile device's screen must be determined and compared to a size limit.

A quick way to do such a comparison is to have only one value that can be compared to a standard value. Traditionally, for video screens, that standard value is the length of a diagonal line connecting it diagonally opposite corners measured in inches or millimeters, i.e. the diagonal. Because there are an ever increasing number of devices and screen sizes, it is not practical to depend on a table mapping the devices their diagonals.

To do the necessary comparison, what is needed is a minium diagonal value, i.e., the value below which a video must be played by the native player. The diagonal value of a device is then calculated and compared to the minimum diagonal value.

To reliably calculate the diagonal value for all possible screens, similar data must be available for all screens. To reliably compare the diagonal values, they must be in the same units. Such information is not available in the user agent data.

What is available is information about the browser window's viewport; specifically the viewport's height, width and pixel ratio. These are available respectively as:
window.screen.height
window.screen.width
window.devicePixelRatio

The height and width are given in device dependent pixels. The last property is the ratio between a device's physical pixel and a density-independent pixel, i.e., dip. For device dependent pixels, the physical size of a pixel varies depending on the pixel density of the device's screen, i.e., the screen density, which is related to the resolution of the device.

These three window properties are available on all of the operating systems and browsers of concern. Thus, a reliable comparison may be made between the minimum diagonal value and a device's diagonal, both given in density-independent pixels.

Video Playback Via an HTML5 video Element
Video playback is accomplished by constructing an HTML5 video element and placing the element in the page or in the modal dialog.

It is important to understand that the HTML5 video element is a more powerful and active element than most HTML elements. A lot of automatic behavior is contained in a video element. A typical video element contains one or more source elements, one for each possible video type. When the video element is clicked, the browser steps through each of the source elements. When a source element that is able to be played in the current environment is encountered, a video panel containing the video is presented to the user. The remaining source elements are skipped.

The video element presents the user with a play icon. The user must tap the play icon to initiate video playback. Mobile device operating systems do not permit automatically playing a video without this user action because the video panel often obscures all other controls. If the play icon were not available, it would be difficult for the user to cancel out of an accidently invoked video. The image for the play icon may vary depending on the operating system, but the behavior is the same.

There are three video types that together enable videos to play in almost any environment. They are: MP4, WebM and Ogg. MP4 videos play in most browsers on most devices. MP4 requires licensing for large, for-profit applications. WebM is an open source alternative to

MP4. Ogg is an open source type that plays that plays in Firefox. As MP4 related patents age, its licensing may taper off. Thus, by including appropriately constructed source elements for these three types, a video can be played in any modern browser on any device.

Because the MVM gathers information about a video's environment, only one source element needs to be created for a particular video session. The video element and its child elements are constructed by manipulating the DOM instead of writing HTML code into the page. This technique is used because writing to the DOM happens immediately and predictably. HTML written into a page may or may not be interpreted immediately. By creating the video element and its child elements, references to those elements are immediately available. Also, each time HTML is written into a page, it is likely to invoke page rendering. Multiple rendering can be avoided by creating one document fragment, doing the work in the fragment and inserting it. While typos are always possible, they are easier to avoid when long strings of HTML are not used.

When the video element is constructed in the DOM fragment, certain elements and attributes may be included or excluded depending on factors like the operating system. For example, on Android devices, a click listener must be explicitly attached to the video element.

Reporting Statistical Data
Certain statistical data must be reported when a video completes or is prematurely exited. Exactly what that data comprises depends on the reporting module. It is hoped that the required data can be gathered by responding to the media events associated with a video element. The events are:
abort, canplay, canplaythrough, durationchange, emptied, ended, error, loadeddata, loadedmetadata, loadstart, pause, play, playing, progress, ratechange, seeked, stalled, suspend, timeupdate, volumechange, waiting.

The required reporting data may be gathered directly from the results of one event or may be inferred from the results of multiple events. Each event that may yield useful data requires a listener to be attached to the video element.

MVM PROGRAM ARCHITECTURE

The MVM is a JavaScript object contained in a file named "mobilevideo.js". The namespace for the MVM object is named "MOBILE_VIDEO". Within the MVM object are the following top level member functions and variables. The implemented MVM object also contains supporting member objects, functions and variables not described here.

Public Member Functions
• insertVideoElemInModal() - Inserts a fully constructed video element into a modal dialog. Calls buildVideoElement() to get a document fragment containing the video element.

• insertVideoElemInPage() - Inserts a fully constructed video element into a page. Calls buildVideoElement() to get a document fragment containing the video element.

• gatherReportingData() - Returns data gathered from events triggered during a video session and perhaps other data sources. Relies on the event listeners attached to the video element by attachReportingListeners().

Priviledged Member Functions
• isDeviceScreenSmall() - Determines if a device has a screen that is below a certain size limit. Calls calcScreenDiagonal() to calculate the size of the device's screen diagonal. Compares the device's screen diagonal size to the size limit, e.g., 700 dpi.

Private Member Functions
• calcScreenDiagonal() - Returns the size of the device's screen diagonal. Calculates the size from window.screen.height, window.screen.width and window.devicePixelRatio.

• buildNoVideoImgEl() - Builds and returns an image element that contains text and perhaps other visual elements to indicate to a user that a video was not found.

• buildVideoElement() - Builds and returns a document fragment containing a video element. The document fragment may then be inserted where the video element is needed. One way to do constuct a complete document fragment is to start by creating an empty fragment. Then, create the video element. Create the video element's child elements. Append the child elements to the video element. If needed, e.g., for Android, attach a click listener to the video element. Attach reporting listeners by calling attachReportingListeners(). Append the video element to the document fragment. Return the fragment.

• attachReportingListeners() - Attaches media event listeners to a video element for gathering data for report statistics.

USING THE MVM PROGRAMMING INTERFACE

The MVM's programming interface comprises these functions, described above:
isDeviceScreenSmall()
insertVideoElemInPage()
insertVideoElemInModal()
gatherReportingData()

The caller calls isDeviceScreenSmall(). If the function returns true, the caller calls insertVideoElemInPage(); otherwise it calls insertVideoElemInModal(). When the video completes or the user exits the video, the caller calls gatherReportingData().