

Appendix A

PLATFORM INFRASTRUCTURE

In this appendix the technical details of the many-screen platform infrastructure are outlined. The infrastructure offers client and server code that can be extended to implement the functionality required for a specific application. The modular code design and use of the model view controller pattern allow for this extensibility and easy code maintenance.

COMPONENTS

The platform consists of two main components the server and base code for application clients. Clients, which import the base code responsible for implementing the functionality listed below, can operate in one of two modes, either application or television. The mode is selected by the inclusion of a <DIV> element, in the applications main HTML file. The DIV element is given a class attribute of either 'appWindow' for applications, or 'tvWindow' for television. Where an instance has been designated as television, the application launches a fullscreen video player view and playhead progress is propagated to all other connected clients and to the server to implement video feed resume functionality across the devices. In the current implementation of the platform, only a single client can be registered as a television. Clients running in application mode offer the remainder of

functionality below, with code to implement the playback and scrubbing of video, channel and feed selection, television control, and display of interactive feeds. In addition, the underlying code structure allows for implementing clients to extend this functionality through the creation of new modules, implementing code specifically for that application's requirements..

The server component is responsible for the connection and disconnection of clients, distributing a schema of channels to newly connected clients and logging user interaction across the devices. In addition, the server maintains playheads for each video feed played on a client instance, which has been setup as a television. Therefore, when new application clients are registered they can be informed of where the television feed has played to. This allows clients connecting after there has been some playback on the television, to accurately display statistical and information updates without revealing spoilers and correctly resuming playback.

FUNCTIONALITY

The platform implemented the following functionality, which can be utilised in implementing applications.

Core Functionality is Extendable: The platform implements core functionality common to most multiscreen television applications. However, the platform is implemented using a modular design and the observer pattern (described in the Design Patterns subsection), so that additional functionality, specific to a particular application, is easily incorporated.

Connection and Disconnection: Application instances that implement the platform can connect and disconnect to the server at anytime during execution, therefore clients that connect after viewers have watched video on the television are able to resume playback of the feed at the correct playhead. In addition the server can support connection from any number of clients and can handle unexpected disconnection and reconnection because of crashes or accidental application closure by users.

Download a schema of channels and feeds: Upon connection, the server provides a client with a schema of channels and feeds that are utilised by the application, the schema is encoded in JSON (JavaScript Object Notation). Logically content is divided into channels that contain feeds. Feeds can either be video or interactive content. Video feeds contain a URI to a video object, a name, description and thumbnail, and fields that dictate the times it is available to the user. Interactive feeds, referred to in the schema as TIC feeds, include the same fields as a video feed, with the exception of the video URI that is replaced by a list of URIs to web page content. Additionally, interactive feeds can be paired with video feeds, allowing for synchronised playback with a video, facilitating stats and information synchronised with the programme video. To complete the implementation of that functionality, the feed also contains a list view flag, when enabled users can switch between available pages and each page has an availability time associated with it.

Live Playhead: The code base implements a live playhead for each of the video feeds available in the channel schema. The live playhead mimics the notion of a broadcast by the playhead increasing as time passes, since the feed was made available to the user. Therefore if a user was to select a feed to playback live, 5 minutes after the feed was made available, the feed would start to playback 5 minutes into the video.

Logging: All user interaction is logged to the server using a consistent protocol, (also utilised by the applications message bus and observer module). The application also builds on problems with logging in the Olympics application, caused by browser native HTML5 video controls, through implementation of custom video playback controls.

Television Functionality

The following functionality is specific to application instances operating in the television mode:

Play a video feed: Playback video feeds as initiated by user selection on application clients. Playback on the television application replicates the

experience of watching broadcast television by displaying the video fullscreen and hiding all browser chrome from the user.

On-Screen Notifications: The television application displays simple on screen notifications to users, confirming feed changes, pausing, and fast forwarding and rewinding.

Application Functionality

Finally, the following functionality is implemented by instances of the code base that operate in the application mode. It is expected that applications in this mode will be run on tablet devices.

List channels and feeds: Enumerate the channels and feeds that are described by the schema distributed by the server. Users can select feeds and channels from this list for playback on the tablet and television. A simplified version of this list functionality, along with the television controls described below, could be utilised in the development of a remote control application.

Instigate video playback: Video feeds that are select by users from the feeds list can be played back on both the television and the tablet application. Subsequently, users have the option to select whether to begin this playback, from the beginning, from the live playhead or to resume previous progress from the television or the tablet. If both tablet and television have consumed part of the video, the higher playhead value will be selected to resume from. For example, if the television had played back a video feed from the start to 5 minutes and the tablet from the 5 minutes to 10 minutes, the tablets playhead would be selected to continue playback from 10 minutes.

Playback video: Application instances can playback video in addition to television instances, enabling the parallel viewing. The video player in earlier iterations of the platform utilised default HTML5 video controls, this generated issues with interaction logging, particularly when users scrub through the video, generating numerous seek interactions, which don't accurately reflect a users intention. Therefore video player module

implements custom video controls, which override the browser default controls.

Interactive content: Interactive feeds of content can be instigated and executed from within the clients identified as applications. These feeds consist of a series of a series of static HTML content. These pages can be enumerated in a user selectable list allowing users to select between available statistic updates. Functionality, such as the runner map and playlist in the MarathOn Multiscreen application, which offer more complex interactive experiences, are implemented by an interactive feed consisting of a single file of content. The file contains an HTML DIV element with a class or ID field that is recognised by a custom code module that implements the required functionality. This technique could enable the development of a range of unique applications, such as quiz shows and interactive narratives.

Control the television: User control of the television is conducted through tablet applications. Using these controls viewers can pause and resume the video feed playing on the television, and fast-forward and rewind.

The following subsections describe how this functionality has been implemented in modules, using a message bus protocol and a series of design patterns

DESIGN PATTERNS

To aid code reuse and organisation the following design patterns were utilised in the development of the multiscreen prototyping platform:

JavaScript revealing module pattern: JavaScript doesn't provide mechanisms for code separation or package syntax, the module pattern allows for code to be organised into self-contained functional units that can be added, replaced or removed (Stefanov, 2010). A module is declared as an immediate function that returns an object this enables public and private scope, code declared in the return object has public scope and can be accessed by other modules. In the revealing version of the pattern all methods are declared in private scope but exposed in the return object (Stefanov, 2010). The division of code into

modules supports easier code maintenance and application specific functionality without altering the core code base. The scoping provisions of the module pattern ensure that the core functionality of the platform is accessed safely by an implementing application. For a list of the platform's core modules see the Modules section below.

Observer Pattern: The observer pattern (otherwise known as the publisher/subscriber module) defines a one-to-many dependency between modules. Such that when communication messages are received, a single update is made to the applications state and all other modules are notified of the change automatically (Gamma et al., 1995). For example the communication messages could derive from user interaction, receiving the channel schema, or playhead updates from other clients of the application. The pattern works by observers (in this instance the core and application specific modules) registering a callback function with the subject module. When an update is made to the applications data model the subject iterates through it's list of observer callback functions and publishes a message informing the module of the model change. Therefore the subject is decoupled from the implementation and number of modules, it is up to the observing module to decide if the message is of use or if it should be discarded. The messages used in the platform to inform observer modules of changes to the model are the same. The central application module exposes several functions to allow other modules to safely request update to the data model, which turn cause the subject module to iterate a message through to it's subscribers. The following code snippet shows the implementation of a module and its observer callback function.

```

var demo = (function()
{
    moduleName = "demoModule";

    var observerCallback = function(message){
        if(message == "ready")
        {
            privateFunction();
        }
    }
    app.subscribe(moduleName, observerCallback);

    var privateFunction = function(){
        console.log("this is the private
function");
    }

    var publicFunction = function(){
        Console.log("This is a public function");
    }

    return{
publicFunction: publicFunction
    }
})();

```

The demo module implements three functions the observerCallback function is registered with the observer patterns subject using the call to `app.subscribe(moduleName, observerCallback)`. When the observerCallback function receives a message from the subject it is compared with the message 'ready', if the condition is matched the privateFunction is called and displays a message to the browser console. The final function, publicFunction is made available to other modules in the application by including a reference to it in the return object.

Model-View-Controller: The model-viewer-controller pattern (abbreviated to MVC) is implemented using the Observer pattern and ensures division between the underlying data model used by the application and its presentation to the user, using controller objects or modules as a go between (Gamma et al., 1995). Again the MVC pattern helps making code readable and extendable. The platform's model has two parts an application model, which is received from the server at runtime, and the internal model, derived from the application model but is updated with playhead values and application specific functionality. The majority of modules are controllers communicating with the application and internal models using their observer callback functions, handling user interactions with the application or responding to playhead events. Application views are implemented using a templating language, in this case EJS (EJS - JavaScript Templates, 2014). JavaScript templating allows for executable code to be included in HTML markup, therefore the views are dynamically generated at runtime in easily maintainable code separate from the controller modules. The following diagram shows the relationship between model and controller modules, and templated views in the multiscreen platform.

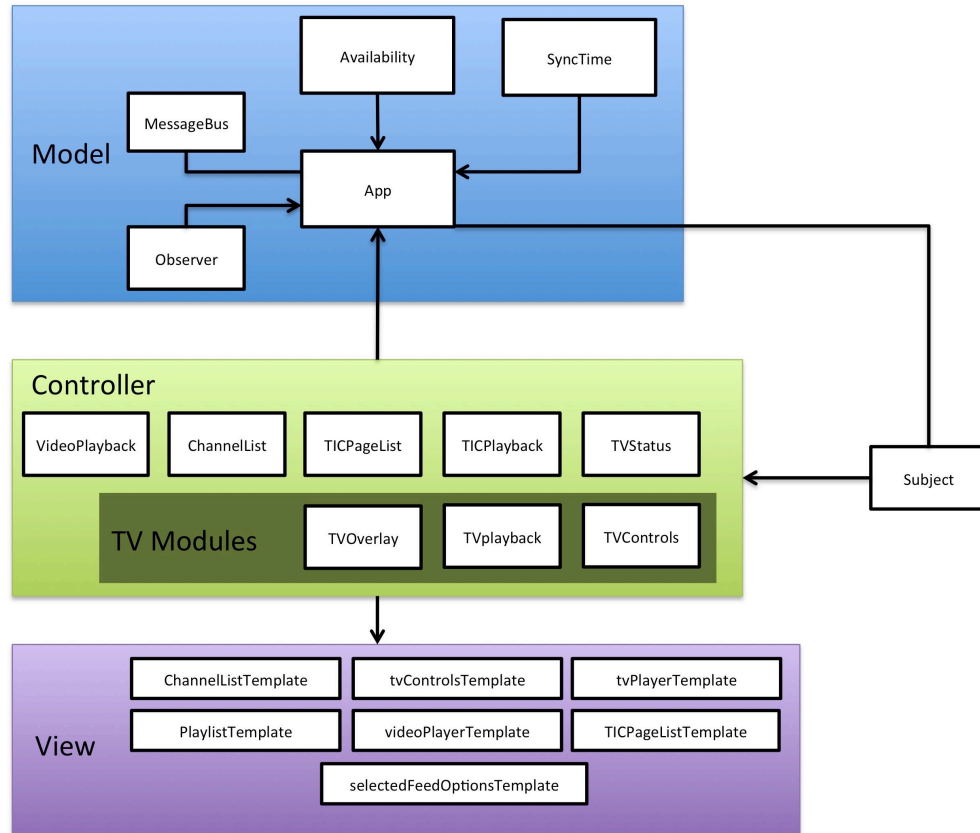


Image A - 1 Multiscreen platform architecture overview

PLATFORM MESSAGING PROTOCOL

Application instances and the server communicate using a message bus, this replaces the long polling techniques used by the Olympics application with a more robust and scalable method of communication. A proprietary message bus, Apache Apollo (Apollo, 2014), is used to broker transmission between components. Platform specific messages are wrapped in the Simple text orientated messaging protocol (abbreviated to STOMP) (STOMP, 2012) that defines client connection and disconnection (to the message bus not the applications server), and message routing. Apollo was chosen as it allows connection natively over web sockets to the web based clients of the application, however Apollo could be replaced with any future product, supporting web sockets and STOMP, and function identically. The following table outlines the message types used by the platform and how they enable the core functionality utilised by the MarathOn Multiscreen application. The communication messages are transmitted across the message bus, in the body section of the STOMP frame, they are encoded in JSON.

Message Type	Description	Example
newConnection	A request from the client to the server to connect. Upon a successful request the server responds with the data message. The connection request also includes the device type either app or television.	<pre>{ "type": "connection", "connectionType": "newConnection", "deviceType": "app" }</pre>
connectionAccept	Response to the server to a connection request, includes a unique identifier for the client and the channel schema for the application to use. Further communication from the client utilises the identifier, so the sever knows how to deal with communication from each client, for example if the client is registered as a television, it updates the servers internal model of television playheads. Additionally all messages include a timestamp, therefore logs have temporal information and nodes can interpret the order of messages correctly.	<pre>{ "type": "connectionAccept", "deviceId": 1416494176508, "data": { "app": [{}], "timestamp": 1416493965553 } }</pre>
Disconnection	A request from the client to the server to disconnect.	<pre>{ "type": "connection", "connectionType": "disconnect", "deviceId": 1416494176184, "deviceType": "tv" }</pre>
Ready	A simple internal message not sent across the message bus informing observing modules that the application and internal models has been setup based on the data received from the server.	<pre>"ready"</pre>
StatusUpdate	User initiated changes to the applications state such as changing the channel or	<pre>{ "statusUpdate": { "type": "feedSelect", "device": "app", "feedId": "raceOverviewVideo", } }</pre>

	<p>selecting a video feed. In the example opposite, the user has selected the professional video feed, which has the ID 'raceOverviewVideo' and the playback location is start. The progress field refers to any progress that the user has already made in the video stream, should the location be resume. In this case it is the first time the video has been played and the progress is set to 0.</p>	<pre>"progress":0, "location":"start"},"timestamp": 1416494240901,"deviceId":1 416494176508}</pre>
PlayheadUpdate	<p>Changes to the playhead state, such as fast forward, rewind and pause. Playhead time updates, raised by the HTML5 video time update method are generated approximately every 200ms, therefore to avoid unnecessary overhead are sent locally, to other observers but not sent across the message bus. This is indicated by the headOnly flag. Other application instances maintain their own playhead clock for the television, responsive to other playheadUpdates and occasional sync updates every 10 seconds. The state field indicates if the video player is paused or playing back and the seeker flag is set to true when the user has fast forwarded or rewound the video feed.</p>	<pre>{"playheadUpdate":{"headOnly":false,"feedId":"raceOvervie wVideo", "progress":5.525578,"state":"p aused", "seeked":false,"eventsViewed": []},"timestamp":14164942471 94,"deviceId":141649417650 8}</pre>
App	<p>The app message type is left for application specific messages. For example, in the MarathOn Multiscreen, the first example opposite adds a new tag to the RunSpotRun dataset, including the tags location along the course (calculated from the latitude and longitude), the</p>	<p>Add new tag</p> <pre>{"app":{"type":"newTag", "tag":{"distance":9412,"id":"14 16494558647454", "latitude":52.9332544804400 56,"longitude":- 1.20449899647435,"raceNo": "1134", "spectatorID":"dcb42e4e-</pre>

	runners bib number, the time in the video and identifiers for the spectator and video.	7196-4d11-837a-9b8aac8a3e8c", "time":1380445760.680195," videoId":"896c3537-c43b-4ed2-a804-d2f1c58fbefc"},"timestamp":1416494347362,"deviceId":1416494176508} Add video to playlist: {"app":{"type":"update", "playlist":[{"feedId":"spectatorp9Video", "startTime":1433,"endTime":1483,"playlistId":"14164946163710", "thumbnail":"http://192.168.0.2/multiscreenWebappMarathon/imgs/thumbs/p9Video.png", "feedName":"spectator 9 videos"}]},"timestamp":1416494405086,"deviceId":1416494176508}
--	--	---

Table A - 1 Multiscreen platform message types

MODULES

The following table outlines the modules implemented by the multiscreen platform and those implemented specifically for the MarathOn Multiscreen application.

Server Core Modules	
Module Name	Module Description
Connection	Handle incoming connections and disconnections from clients, including distributing the channel schema.
Logging	Log user interactions from the message bus into a persistent file.
main	Main module which kicks off the server and handles loading in of persistent channel schema defined by a command line argument.
tvPlaybackState	Maintain the playback status of each channel for the television client.
MarathOn Multiscreen Server Modules	

playlistServer	Maintains the state of the playlist as an ordered array of videos.
Core Modules	
App	<p>The application module stores the application and instance models. It is also responsible for providing other modules safe access to the models and implements the subject for the observer pattern.</p> <p>In addition the app module is the entry point for the application and kick start connection to the message bus.</p>
Availability	The availability module determines which channels and feeds are available to users at any given time during the clients execution. Therefore feeds which have a start time and end time set are only available to users at the appropriate time.
ChannelList	Controller module that handles display and interaction with the channel list and feed selection.
MessageBus	Interface between the Message bus, the client and server. This module handles connection and disconnection to the message bus and server, and passes incoming messages to the application module and outgoing messages to the bus.
Observer	Handles subscription and unsubscription of modules to the subject (application module), publishes messages to subscribing modules.
SyncTime	The SyncTime module provides a synced time with the server, so playhead calculations of other clients are accurate. The syncTime is calculated by requesting a time from the server at the start of the clients execution and adjusting for network latency. All subsequent requests for a time value are calculated as a delta from the requested server time.
TICPageList	Enumerate the HTML pages available in an interactive feed and handle user interaction by issuing a playheadUpdate message to the observer subject to be handled by the TICPlayback module.
TICPlayback	Controller module that displays the selected interactive feed page.
TVControls	Controller module that displays the TV controls on an application client and handles user interaction with them, issuing playhead update messages to be sent across the bus.
TVOverlay	Module to handle the display of information overlays on the television client, such as pause notifications and channel

	changes.
TVPlayback	Responds to user interactions on application clients to playback and control the playback of video feeds on the television.
TVStatus	The TVstatus module is used by clients operating in application mode. The module maintains the playhead status of a TV client.
VideoPlayback	Controller module which handles playback and user interaction of video feeds on application clients.

Table A - 2 Multiscreen platform messages

Example Channel Schema

```
[
  {
    "channelName": "Race Overview",
    "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/marathumb.jpg",
    "startTime": 0,
    "channelId": "raceOverviewChannel",
    "visible": true,
    "endTime": 1911,
    "feeds": [
      {
        "type": "video",
        "feedId": "raceOverviewVideo",
        "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/marathumb.jpg",
        "visible": true,
        "description": "Watch the runners head out
from the start and the elite runners race. Also
watch the runners approach the finish line and
complete the challenge",
        "feedName": "Video: Race Overview",
        "startTime": 0,
        "endTime": 0,
        "videoList": [
          {
            "url":
"http://192.168.0.2/videos/raceOverview.m4v",
            "startTime": "0",
            "endTime": 1911,
            "eventList": []
          }
        ],
        "autoplay": {
```

```

        "tv": "no",
        "app": "no"
    }
},
{
    "type": "TIC",
    "feedId": "raceOverviewTIC",
    "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/info.jpg",
    "visible": true,
    "description": "Check out facts about
marathons and Nottingham's Robin Hood Marathon and
Half Marathon",
    "feedName": "Marathon Facts and Figures",
    "startTime": 0,
    "endTime": 0,
    "linkedToVideo": "raceOverviewVideo",
    "spoilerLock": false,
    "scrubber": false,
    "pageListDisplay": true,
    "pageList": [
        {
            "title": "Race Route",
            "pageId": "ProPage1",
            "time": 30,
            "displayAuto": false,
            "eventListId": 0,
            "custom": {},
            "url":
"http://192.168.0.2/multiscreenWebappMarathon/pages
/route.txt"
        },
        {
            "title": "Wheelchair Race",
            "pageId": "ProPage2",
            "time": 215,
            "displayAuto": false,
            "eventListId": 0,
            "custom": {},
            "url":
"http://192.168.0.2/multiscreenWebappMarathon/pages
/wheelchairStart.txt"
        },
        {
            "title": "Race History",
            "pageId": "ProPage3",
            "time": 350,
            "displayAuto": false,

```

```

        "eventListId": 0,
        "custom": {},
        "url":
"http://192.168.0.2/multiscreenWebappMarathon/pages
/nottRaceOverview.txt"
    },
    {
        "title": "Marathon Facts",
        "pageId": "ProPage5",
        "time": 420,
        "displayAuto": false,
        "eventListId": 0,
        "custom": {},
        "url":
"http://192.168.0.2/multiscreenWebappMarathon/pages
/marathonFacts.txt"
    },
    {
        "title": "Marathon Origins",
        "pageId": "ProPage8",
        "time": 490,
        "displayAuto": false,
        "eventListId": 0,
        "custom": {},
        "url":
"http://192.168.0.2/multiscreenWebappMarathon/pages
/origin.txt"
    },
    {
        "title": "Men's result",
        "pageId": "ProPage6",
        "time": 577,
        "displayAuto": false,
        "eventListId": 0,
        "custom": {},
        "url":
"http://192.168.0.2/multiscreenWebappMarathon/pages
/mensResults.txt"
    },
    {
        "title": "Women's result",
        "pageId": "ProPage7",
        "time": 700,
        "displayAuto": false,
        "eventListId": 0,
        "custom": {},
        "url":
"http://192.168.0.2/multiscreenWebappMarathon/pages
/womensResults.txt"
    }

```



```

    },
    {
      "title": "Robin Hood WR",
      "pageId": "ProPage4",
      "time": 93,
      "displayAuto": false,
      "eventListId": 0,
      "custom": {},
      "url":
"http://192.168.0.2/multiscreenWebappMarathon/pages
/robinhood.txt"
    }
  ],
  "autoplay": {
    "tv": "no",
    "app": "no"
  }
}
],
"autoSelect": false
},
{
  "channelName": "My Videos",
  "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/myVideos.jpg",
  "visible": true,
  "startTime": 0,
  "channelId": "myVideosChannel",
  "endTime": 0,
  "feeds": [
    {
      "type": "video",
      "feedId": "spectatorp2Video",
      "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/p2Video.png",
      "visible": true,
      "description": "Watch all the videos from
spectator2, who watched the race from university
park.",
      "feedName": "spectatorp2 videos",
      "custom": {

    },
    "startTime": 20,
    "endTime": 754,
    "videoList": [
      {

```

```

        "url":
"http://192.168.0.2/videos/spectatorVideos/p2Video.
mp4",
        "startTime": "0",
        "endTime": 754,
        "eventList": []
    }
],
"autoplay": {
    "tv": "no",
    "app": "no"
}
},
{
    "type": "video",
    "feedId": "spectatorp3Video",
    "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/p3Video.png",
    "visible": true,
    "description": "Watch all the videos from
spectator3, who watched the race from University
park.",
    "feedName": "spectatorp3 videos",
    "startTime": 20,
    "endTime": 754,
    "custom": {

    },
    "videoList": [
        {
            "url":
"http://192.168.0.2/videos/spectatorVideos/p3Video.
mp4",
            "startTime": "0",
            "endTime": 754,
            "eventList": []
        }
    ],
    "autoplay": {
        "tv": "no",
        "app": "no"
    }
},
{
    "type": "video",
    "feedId": "spectatorp5Video",

```

```

        "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/p5Video.png",
        "visible": true,
        "description": "Watch all the videos from
spectator5, who watched the race from the corner of
Beacon Road and Queen's Road East in Beeston.",
        "feedName": "spectatorp5 videos",
        "startTime": 20,
        "endTime": 754,
        "custom": {

        },
        "videoList": [
        {
            "url":
"http://192.168.0.2/videos/spectatorVideos/p5Video.
mp4",
            "startTime": "0",
            "endTime": 754,
            "eventList": []
        }
    ],
    "autoplay": {
        "tv": "no",
        "app": "no"
    }
},
{
    "type": "video",
    "feedId": "spectatorp6Video",
    "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/p6Video.png",
    "visible": true,
    "description": "Watch all the videos from
spectator6, who watched the race from the corner of
Beacon Road and Queen's Road East in Beeston.",
    "feedName": "spectatorp6 videos",
    "startTime": 20,
    "custom": {

    },
    "endTime": 754,
    "videoList": [
        {
            "url":
"http://192.168.0.2/videos/spectatorVideos/p6Video.
mp4",

```

```

        "startTime": "0",
        "endTime": 754,
        "eventList": []
    }
],
"autoplay": {
    "tv": "no",
    "app": "no"
}
},
{
    "type": "video",
    "feedId": "spectatorp7Video",
    "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/p7Video.png",
    "visible": true,
    "description": "Watch all the videos from
spectator7, who watched the race from Castle
Boulevard.",
    "feedName": "spectatorp7 Videos",
    "startTime": 20,
    "endTime": 754,
    "custom": {

    },
    "videoList": [
        {
            "url":
"http://192.168.0.2/videos/spectatorVideos/p7Video.
mp4",
            "startTime": "0",
            "endTime": 754,
            "eventList": []
        }
    ],
    "autoplay": {
        "tv": "no",
        "app": "no"
    }
}
},
{
    "type": "video",
    "feedId": "spectatorp8Video",
    "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/p8Video.png",
    "visible": true,

```

```

        "description": "Watch all the videos from
spectator8, who watched the race from University
Park and surrounding area.",
        "feedName": "spectatorp8 videos",
        "startTime": 20,
        "endTime": 754,
        "custom": {

        },
        "videoList": [
            {
                "url":
"http://192.168.0.2/videos/spectatorVideos/p8Video.
mp4",
                "startTime": "0",
                "endTime": 754,
                "eventList": []
            }
        ],
        "autoplay": {
            "tv": "no",
            "app": "no"
        }
    },
    {
        "type": "video",
        "feedId": "spectatorp9Video",
        "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/p9Video.png",
        "visible": true,
        "description": "Watch all the videos from
spectator9, who watched the race from University
Park",
        "feedName": "spectatorp9 videos",
        "startTime": 20,
        "endTime": 754,
        "custom": {

        },
        "videoList": [
            {
                "url":
"http://192.168.0.2/videos/spectatorVideos/p9Video.
mp4",
                "startTime": "0",
                "endTime": 754,
                "eventList": []
            }
        ]
    }

```

```

    ],
    "autoplay": {
        "tv": "no",
        "app": "no"
    }
},
{
    "type": "video",
    "feedId": "spectatorp11Video",
    "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/p11Video.png",
    "visible": true,
    "description": "Watch all the videos from
spectator11 who watched the race from near
Nottingham Station, Castle Boulevard and the finish
line.",
    "feedName": "spectatorp11 videos",
    "startTime": 20,
    "endTime": 754,
    "custom": {

    },
    "videoList": [
        {
            "url":
"http://192.168.0.2/videos/spectatorVideos/p11Video
.mp4",
            "startTime": "0",
            "endTime": 754,
            "eventList": []
        }
    ],
    "autoplay": {
        "tv": "no",
        "app": "no"
    }
},
{
    "type": "video",
    "feedId": "spectatorp12Video",
    "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/p12Video.png",
    "visible": true,
    "description": "Watch all the videos from
spectator12, who watched the race from near
Nottingham Station and the finish line.",
    "feedName": "spectatorp12 videos",

```

```

        "startTime": 20,
        "endTime": 754,
        "custom": {

        },
        "videoList": [
            {
                "url":
"http://192.168.0.2/videos/spectatorVideos/p12Video
.mp4",
                "startTime": "0",
                "endTime": 754,
                "eventList": []
            }
        ],
        "autoplay": {
            "tv": "no",
            "app": "no"
        }
    },
    {
        "type": "video",
        "feedId": "spectatorp13Video",
        "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/p13Video.png",
        "visible": true,
        "description": "Watch all the videos from
spectator13, who watched the race from University
Park",
        "feedName": "spectatorp13 videos",
        "startTime": 20,
        "endTime": 754,
        "custom": {

        },
        "videoList": [
            {
                "url":
"http://192.168.0.2/videos/spectatorVideos/p13Video
.mp4",
                "startTime": "0",
                "endTime": 754,
                "eventList": []
            }
        ],
        "autoplay": {
            "tv": "no",
            "app": "no"
        }
    }

```

```

    }
  },
  {
    "type": "video",
    "feedId": "spectatorp15Video",
    "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/p15Video.png",
    "visible": true,
    "description": "Watch all the videos from
spectator15, who watched the race from Castle
Boulevard and the finish line.",
    "feedName": "spectatorp15 videos",
    "startTime": 20,
    "endTime": 754,
    "custom": {

    },
    "videoList": [
      {
        "url":
"http://192.168.0.2/videos/spectatorVideos/p15Video
.mp4",
        "startTime": "0",
        "endTime": 754,
        "eventList": []
      }
    ],
    "autoplay": {
      "tv": "no",
      "app": "no"
    }
  },
  {
    "type": "video",
    "feedId": "spectatorp16Video",
    "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/p16Video.png",
    "visible": true,
    "description": "Watch all the videos from
spectator16, who watched the race from University
Park.",
    "feedName": "spectatorp16 videos",
    "startTime": 20,
    "endTime": 754,
    "custom": {

    },

```



```

        "videoList": [
            {
                "url":
"http://192.168.0.2/videos/spectatorVideos/p16Video
.mp4",
                "startTime": "0",
                "endTime": 754,
                "eventList": []
            }
        ],
        "autoplay": {
            "tv": "no",
            "app": "no"
        }
    },
    {
        "type": "video",
        "feedId": "spectatorp17Video",
        "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/p17Video.png",
        "visible": true,
        "description": "Watch all the videos from
spectator17, who watched the race from castle
Boulevard.",
        "feedName": "spectatorp17 videos",
        "startTime": 20,
        "endTime": 754,
        "custom": {

        },
        "videoList": [
            {
                "url":
"http://192.168.0.2/videos/spectatorVideos/p17Video
.mp4",
                "startTime": "0",
                "endTime": 754,
                "eventList": []
            }
        ],
        "autoplay": {
            "tv": "no",
            "app": "no"
        }
    },
    {
        "type": "video",
        "feedId": "spectatorp18Video",

```

```

        "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/p18Video.png",
        "visible": true,
        "description": "Watch all the videos from
spectator18, who watched the race from the start
and finish lines.",
        "feedName": "spectatorp18 videos",
        "startTime": 20,
        "endTime": 754,
        "custom": {

    },
    "videoList": [
        {
            "url":
"http://192.168.0.2/videos/spectatorVideos/p18Video
.mp4",
            "startTime": "0",
            "endTime": 754,
            "eventList": []
        }
    ],
    "autoplay": {
        "tv": "no",
        "app": "no"
    }
},
{
    "type": "video",
    "feedId": "spectatorp19Video",
    "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/p19Video.png",
    "visible": true,
    "description": "Watch all the videos from
spectator19, who watched the race from castle
Boulevard and the approach to the finish line.",
    "feedName": "spectatorp19 videos",
    "startTime": 20,
    "endTime": 754,
    "custom": {

    },
    "videoList": [
        {
            "url":
"http://192.168.0.2/videos/spectatorVideos/p19Video
.mp4",

```

```

        "startTime": "0",
        "endTime": 754,
        "eventList": []
    }
],
"autoplay": {
    "tv": "no",
    "app": "no"
}
},
{
    "type": "video",
    "feedId": "spectatorp20Video",
    "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/p20Video.png",
    "visible": true,
    "description": "Watch all the videos from
spectator20, who watched the race from the start
line and the approach to the finish.",
    "feedName": "spectatorp20 videos",
    "startTime": 20,
    "endTime": 754,
    "custom": {

    },
    "videoList": [
        {
            "url":
"http://192.168.0.2/videos/spectatorVideos/p20Video
.mp4",
            "startTime": "0",
            "endTime": 754,
            "eventList": []
        }
    ],
    "autoplay": {
        "tv": "no",
        "app": "no"
    }
}
},
{
    "type": "video",
    "feedId": "spectatorp21Video",
    "thumbNail":
"http://192.168.0.2/multiscreenWebappMarathon/imgs/
thumbs/p21Video.png",
    "visible": true,

```

```
        "description": "Watch all the videos from
spectator21, who watched the race from University
Park",
        "feedName": "spectatorp21 Videos",
        "startTime": 20,
        "endTime": 754,
        "custom": {

        },
        "videoList": [
            {
                "url":
"http://192.168.0.2/videos/spectatorVideos/p21Video
.mp4",
                "startTime": "0",
                "endTime": 754,
                "eventList": []
            }
        ],
        "autoplay": {
            "tv": "no",
            "app": "no"
        }
    },
    "autoSelect": false
}
]
```