Understanding MOVING Together Links

# Legacy Links - The Good ‘ol Days

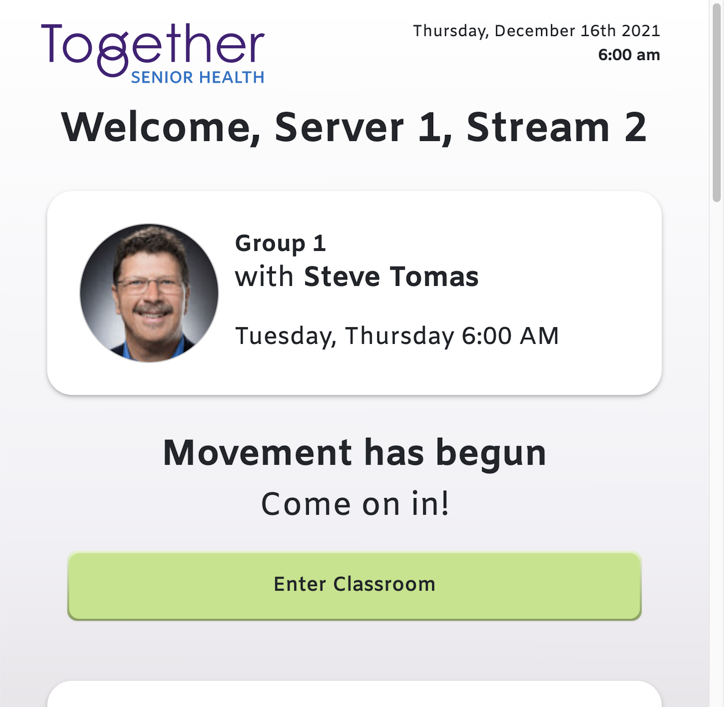
Before the iPad, links to the AV platform had a single format which took the user straight to the landing page and identified the user using their unique ticket:

<https://mt1.test.tsh.care/session/upcoming?ticket=G-P9kYs6er0OW69oNr_NR-j71XfcdN39dObjhQ9042UxmeskxJRQNphVwVU9lxn8>

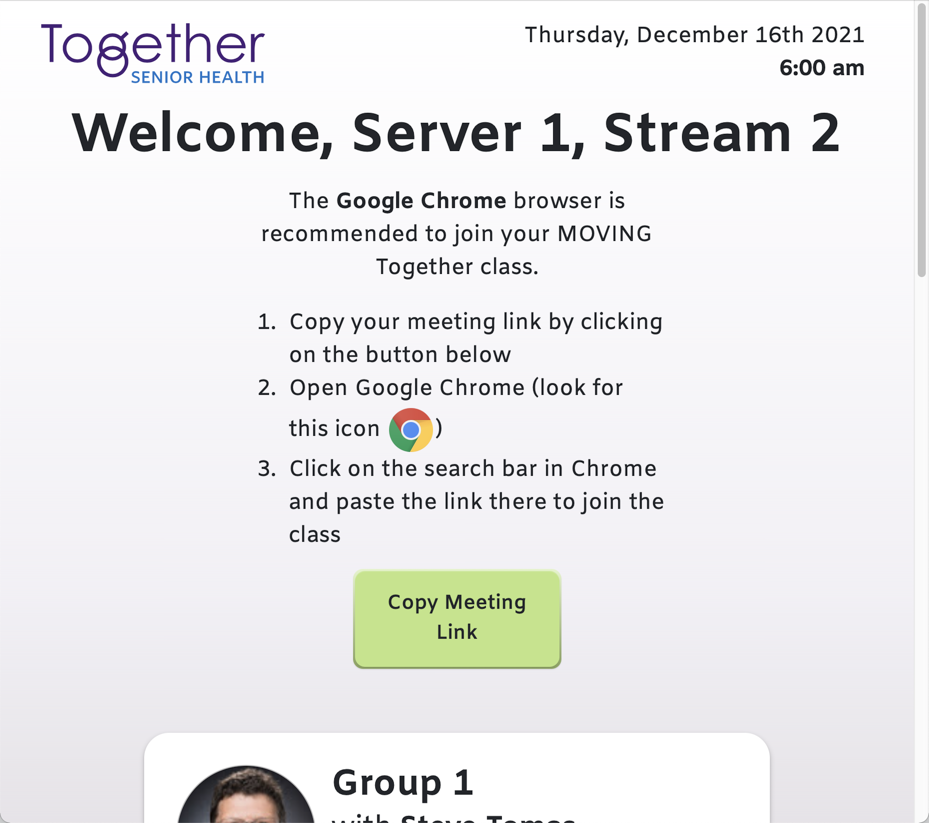
Let’s define some common terminology for the parts of the URL from this sample:

| **Example** | **Part** | **Notes** |
| --- | --- | --- |
| https | Scheme | Lets us know what protocols are used (i.e. how to handle) the link. For the browser, this is always “https” for the AV platform |
| mt1.test.tsh.care | Domain Name | DNS name of the server to use - the test server in this case |
| session/upcoming | Relative URL | Tells the server the specific page to use - the landing page in this case. |
| ?ticket=G-P9…..lxn8 | Query String(s) | Additional parameters specific to the call. This are simply name/value pairs following the internet standard in this form: ?name1=value1&name2=value2&name3=value3. Notice that the first one is preceded by a ‘?’, and the remainder by ‘&’. For our URLs, we usually use a ‘ticket’ value, which identifies the user, and a ‘forceTime’ value, which overrides the current time for testing purposes. |

For desktop platforms, we simply paste the URL into the browser (or follow a link form an email, etc) and the browser takes us directly to the page, using the ticket to identify the user:



Remember that the server will check what browser that the user is using, and will prompt the user to switch to Chrome if it’s not their default browser:



# 

# An App is Born: Scheme-based Links

Once we have created our iPad app, we need some way of telling IOS that we want our app to open instead of the browser. We want this to happen when the user clicks a link in an email, follows a saved bookmark in their browser, or even pastes a link into the browser.

IOS has a special URL format that instructs the browser to open our app, instead of showing the page in the browser. This is done by replacing the *scheme* portion of the URL with a unique identifier for our app (‘tsh’ in our case):

tsh://session/upcoming?forceTime=2021-12-16T14:00:00.000Z&ticket=0RLVTz6dOt-nopLmCbzK7ZkfhfmKLgfTtkaMrXyQZgsCtxNay-LmO7Y\_vMVwcxdw

Paste this link into the browser (or click from an email) and you will be prompted to open the TSH application on IOS:

# 

Once the app opens, the app can access the relative URL and query strings to determine the page to display and identify the user via their ticket.

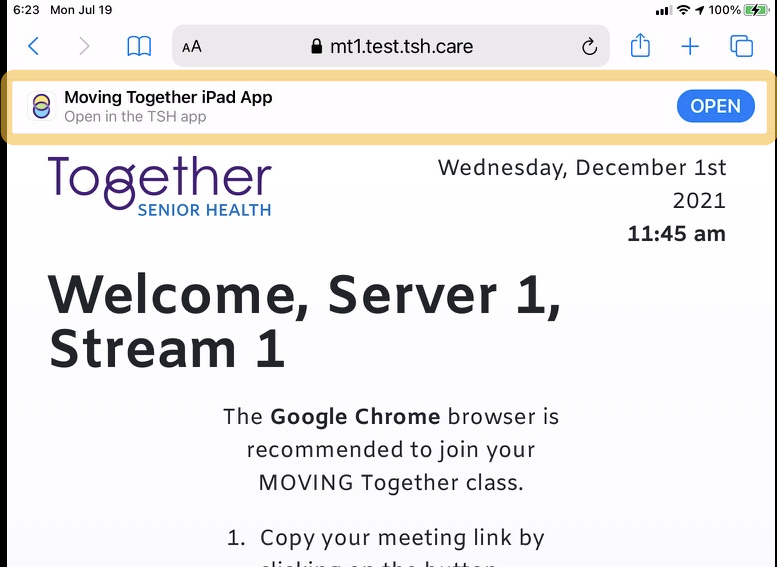
We call these links *scheme-based links*, since they use the scheme portion of a URL to determine how to open the link (‘https’ for browser, ‘tsh’ for our app).

An obvious downside to using the scheme in the URL to tell us how to view the page (browser vs. app) is that we have to give the user a different link, depending on their preference. This gets more complicated for the user, who might even be using both desktop and IOS.

# One Link To Rule Them All

While legacy links work for the browser, and scheme-based links can take the user directly to our app, what we really want is to simplify the experience for the user. Give them one link that takes them to the app, if installed, or opens it in the browser, if the app is not an option.

One option is what Apple calls “Universal Linking”. To implement this, we make a minor modification to the AV site to tell IOS that the page they have visited can also be opened by an app. We’ve implemented this, and it will how a banner at the top of the page to give the user the option of opening our app:

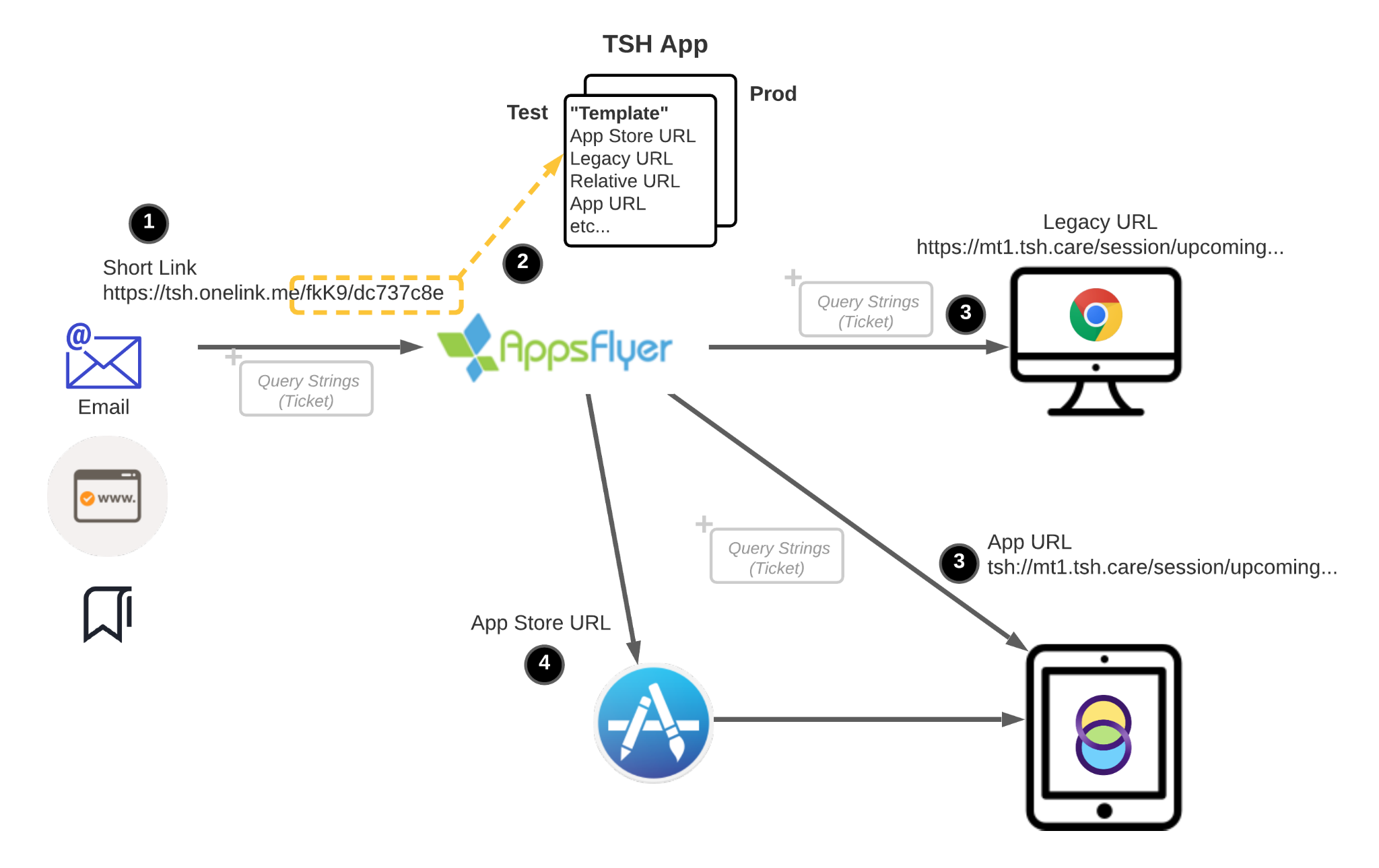


There are some downsides to this approach:

1. It’s too easy to miss what to do next; the “OPEN” button doesn’t pop-up automatically in IOS - the user has to swipe down to see the option.
2. Even if obvious, opening the app is an additional step for the user - it would be better if the app opened directly, instead of going to the browser first.
3. If the app is not installed, the option is not presented and the user hits a dead end. We could prompt them to install the app on the page, but there is more to go wrong. Once they install, then they have to go back to the original link and start over!

Fortunately, we’re not the only app creator to need a solution, and there are 3rd party services that provide a solution; the one we’ve chosen is called [AppsFlyer](https://www.appsflyer.com/). AppsFlyer creates one link that sends the user to the AppsFlyer site. Logic on the site determines the user’s OS, if the app is installed (sending them to the app store if not!) and redirects them either to the correct page in the browser, or to the installed app ready to join their class!

This diagram illustrates how AppsFlyer works:



The process of clicking on a link in an email a bookmark follows these steps:

1. A *short link* sends the user directly to the AppsFlyer server. The short link contains unique short keys which identify the application and *template* associated with the link.  
     
   Additional query string values can be added to the short link. In this case, the link is cleverly referred to as a “long link” (after all, it will be somewhat longer!), and the query strings will follow along in the process. Once the user reaches a landing page in the browser or app, the query string will identify the user who is joining a class.
2. The AppsFlyer server figures out where the user needs to be sent depending on the current operating system (desktop or IOS), and whether the iPad app is installed. The URLs for the destinations in each case are determined by the values in the template.
3. The user is redirected to the browser or app via a URL, with query strings in tow.
4. If the user is on IOS, and the app is not installed, the user is sent to the app’s URL on the Apple app store, when the user can download the app.
5. Upon opening the app after installation, a special integration with AppsFlyer will call back to the AppsFlyer server to determine if the user had previously been following a link, and will redirect the user to the proper page and retrieve the original query strings.

**NOTE**: The AppsFlyer template URLs should *never* contain user ticket or fordeTime information!