Thoughts on Web Applications – Well defined URLs

Balaji N.V.
Samsung India Software Operations, Bangalore, India

Abstract

Web apps require more capabilities to realize broader spectrum of applications. Ranking top among those capabilities is well defined URL for installed applications

In this paper, we discuss benefits of such a well-defined URL and outline how this could be defined. Such a capability is already defined partially in [Widget-URI], and the same could be extended to realize what is defined here.

Introduction

In the recent past web technology based platforms - hp's webOS, Google's Chromium OS, PhoneGap and Samsung's bada web app framework – have emerged, and they allow applications to be developed using HTML/CSS/JavaScript. Web apps are central to these platforms.

In the existing systems installed applications differs from web pages when it comes to URI, linking and sharing. Web pages have well defined unique URIs, linkable and sharable. Unfortunately installed web apps do not have any of these important features of web.

In this paper we discuss about unique URL for locally installed web applications. We believe well-defined unique URL is fundamental to enhancing the capability of web applications as it would allow linking and sharing of web applications.

URL of the Installed web application

Installed web apps should use a separate URL scheme which is different from file://. [Widget-URI] defines such a scheme already. However, URI defined in Widget URI specification cannot be linked or shared as it is synthetically generated. It can neither be used as re-direction URL with protocols such as oAuth. The name of the scheme could be widget:// as defined in [Widget-URI] or any other meaningful string. The name “app” is used throughout this paper for the purpose of providing better clarity.

We propose that the sources from where apps are installed (E.g. app stores) should ensure uniqueness of the IDs assigned to the application within its control. They should also ensure the IDs are URIs as defined in [RFC3986]. It is not required for the applications IDs to be unique across different application sources.

The authority part of the URL should be derived from the authority part of the application ID. The URL includes a mandatory path component, and it contains the origin from where the app is obtained. For example, a Yahoo Mail application downloaded from Yahoo store will have following URL:

$$\underbrace{\text{app:}}_{\text{Scheme}}\text{//}\underbrace{\text{mail.yahoo.com/}}_{\text{App ID}}\underbrace{\text{@store.yahoo.com/}}_{\text{App Source}}$$

If no specific app source could be associated with the app ((E.g. app is installed through side loading), then a new globally unique identifier should be generated and set as an app source. The unique identifier could be generated at the app installation time and used till the app gets uninstalled.

Any further path components, query components or fragments specific to an application will follow the mandatory app-source path. URL without App ID part and App source path should be interpreted as a relative URL.

Origin of Installed web application

We propose the origin of the installed application to be defined as follows:

1. Let scheme be app://
2. Let host be the host part of the Identifier of the application
3. Let port be the port part of the Identifier of the application
4. Let the extra data be the app store location from where this app is downloaded from. If no app source could be (in case of side loading) related, then extra data shall be set to new globally unique identifier
    - ID need not be unique across application sources.

Using installed web app's URL as an service End-point

It should be possible to use the above-defined URL as a service end point with various existing mechanisms such as Server-sent-Events, Web Messaging and XMLHttpRequest. The app:// scheme meant for web applications should mimic the HTTP protocol. One possibility is just to dereference the URL, map them to files, read the files and return the contents. But such a mechanism does not give the called application an opportunity to process the query strings and return dynamic contents when invoked with XHR. And, Cross-origin permission granting using [CORS] will not be possible with just file read. Moreover, periodic notifications using EventSource can't be achieved by file read.

Therefore, a slightly different processing model should be defined for XHR and EventSource. When a source application sends an XHR to the target application, that request should be delivered to the target application through an event notification. The target application should be allowed to respond to that request (as done normally by a server based application).

For each XHR request:

- Launch the application, if it is not active
- Notify the target application about arrival of the request by dispatching 'newRequest' event to appMessaging object of the target application
- Request object has a function using which the application may respond to the request. Invocation of 'respond' method should send the response data to the source application.
- The response shall follow all the semantics of an HTTP response. The source should process the received response as if it has received response from an HTTP server.

When a source entity initiates a request to an application having scheme app://, and the target application is installed in the system but not launched, the app should be launched and the message delivered.

When a source entity initiates a request to an application having scheme app://, and the target application is not installed in the system, error response should be returned to the source.

Here we outline the interfaces that the developer should be provided with and the behavior.

```
interface appMessaging : EventTarget {

                attribute Function? onnewRequest;
};

window implements appMessaging;



interface newRequest: MessageEvent {

            //constants for source

            const unsigned short XHR= 0;

             const unsigned short EVENTSOURCE= 1;

            readonly attribute unsigned short source;

            readonly attribute appRequest request;


            void respond(message);
};
```

request object should contain the request header parameters set in the source.

It is to be noted that the above interface and behaviors are not formal specifications but provided only for the purpose of understanding.

In essence, the user agents should handle the URLs with app:// in different ways:

- When used with functions like window.open, the user-agent should dereference the URL and launch the application, if it is not active. The query parameters and fragments should be passed to the start file of the application. While de-referencing the URL, user-agents should consume App source of the path
- When used with functions like XHR and EventSource, it should process as explained above

Conclusion

Lack of well-defined URL is an impediment to the evolution of web apps. Therefore, standards should defined a URL format for installed applications in a way it can be linked, shared, used for inter-app communication and used as re-direct URLs in protocols such as oAuth.

References

[CORS]
Cross-Origin Resource Sharing, Anne van Kesteren

[RFC3986]
Uniform Resource Identifier (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L. Masinter. IETF.

[Widget-URI]
Widget URI scheme, Marcos Cáceres