RESTful PTK Integration

Some Simple Interactions

Create a new conversation by POSTing an XMLHttpRequest as below

```
function createConversation(segment) {
    xhr = new XMLHttpRequest();
    xhr.open("POST", url, true);
    xhr.setRequestHeader("Content-type","application/json");
    xhr.setRequestHeader("Accept", "application/json");
    xhr.onreadystatechange = function () {
        // onreadystatechange event handler used to react to the web request
        if ((xhr.readyState == 4)) {
            interactionLocation = xhr.getResponseHeader("Location");
            json = eval('(' + xhr.responseText + ')');
            originalWaitTime = json.originalWaitTime;
            originalWaitTimeComponents = originalWaitTime.split(':');
            ewtMinutes = parseInt(originalWaitTimeComponents[0]) + 1;
            updateASAP(ewtMinutes);
        }
    };
    xhr.send("{\"segment\":\""+segment+"\"}");
}
```

The interactionLocation (global) holds the URL needed to interact with the conversation we just created. The mythical updateASAP() function represents a mechanism to express the EWT to the user.

To update the conversation, we again use an XMLHttpRequest. This time we will PUT the request, passing additional data to the conversation.

```
function updateConversationWithUser(user) {
    xhr = new XMLHttpRequest();
    xhr.open("PUT", interactionLocation, true);
    xhr.setRequestHeader("Content-type", "application/json");
    xhr.setRequestHeader("Accept", "application/json");
    xhr.onreadystatechange = function () {
        if (xhr.readyState == 4) {
            jsonPUT = eval('(' + xhr.responseText + ')');
            interactionStatus = jsonPUT.status;
        }
    };
    xhr.send("{\"context\":{\"user\":\"" + user + "\"}}");
}
```

The interactionStatus is holding, not the status of the call to the PTK, but the actual status of the conversation.

To create a callback, we need to PUT again. Instead of updating the context, we will update the contactUri. This attribute can contain both the callback number and an opitonal appointment time (for scheduled callbacks). Here is an example of the simpler ASAP callback. This assumes all data validation has occurred and that phoneNumber is a well formatted number.

```
function updateConversationToAsapCallback(phoneNumber) {
    xhr = new XMLHttpRequest();
    xhr.open("PUT", interactionLocation, true);
    xhr.setRequestHeader("Content-type", "application/json");
    xhr.setRequestHeader("Accept", "application/json");
    xhr.onreadystatechange = function () {
        if (xhr.readyState == 4) {
            jsonPUT = eval('(' + xhr.responseText + ')');
            interactionStatus = jsonPUT.status;
        }
    };
    xhr.send("{\"contactUri\":\"voice://" + phoneNumber + "\"}");
}
```

To better support appointments, we should refactor the code

```
function updateConversationToAsapCallback(phoneNumber) {
    updateConversationToCallback("voice://" + phoneNumber);
}

function updateConversationToCallback(contactUri) {
    xhr = new XMLHttpRequest();
```

```
    xhr.open("PUT", interactionLocation, true);
    xhr.setRequestHeader("Content-type", "application/json");
    xhr.setRequestHeader("Accept", "application/json");
    xhr.onreadystatechange = function () {
        if (xhr.readyState == 4) {
            jsonPUT = eval('(' + xhr.responseText + ')');
            interactionStatus = jsonPUT.status;
        }
    };
    xhr.send("{\"contactUri\":\""+contactUri+"\"}");
}
```

Now, we can just create another one-liner to handle scheduled callbacks

```
function updateConversationToScheduledCallback(phoneNumber,appointmentTime) {
    udpateConversationToCallback("voice://"+phoneNumber+"?appt="+appointmentTime);
}
```

Putting It All Together

Lets create a `<div>` to house our callback widget. We'll class it as a widget, for styling, and get it an id for scripting.

```
<div class="widget" id="cb_widget">

</div>
```

Initially, the wdiget should not be visible, since we haven't yet gotten in touch with Virtual Hold. So, our CSS will look like

```
div.widget {
    display: none;
}
```

The styling information can be embedded inside a `<style>` tag, or linked from our page to an external file as

```
<link rel="stylesheet" type="text/css" href="callback.css"></link>
```

Now, we should have a webpage that displays nothing. This is much the same state we were in before we added any code. But, we've set the stage for great things to come...

Now we can add our createConversation function from above. But, we'll include a couple of new tricks. First, we'll source in the jquery library to simplify control of our web page. Second, we'll have the code actually *do* something (make our widget `<div>` appear) when we get back a reply.

```
<html>
<head>
<title>My Simple Callback</title>
<link rel="stylesheet" type="text/css" href="callback.css"></link>
<script language="javascript" src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
</head>
<body>
<div class="widget" id="cb_widget">
    <button id="cb_button">Call me back now</button>
</div>
<script language="javascript">
var interactionLocation;
$(document).ready(function() {
    createConversation("http://localhost:8000/conversations","PlatformToolKit",onConversation
});

function onConversationCreated(xhr) {
    interactionLocation = xhr.getResponseHeader("Location");
    $('#cb_widget').show();
}

function createConversation(url,segment,callback) {
    xhr = new XMLHttpRequest();
    xhr.open("POST", url, true);
    xhr.setRequestHeader("Content-type","application/json");
    xhr.setRequestHeader("Accept", "application/json");
    xhr.onreadystatechange = function () {
        // onreadystatechange event handler used to react to the web request
        if ((xhr.readyState == 4)) {
```

```
            callback(xhr);
        }
    };
    xhr.send("{\"segment\":\""+segment+"\"}");
}

</script>
</body>
</html>
```

Now we can add a handler for the button click and see if we can get a callback. We'll also make our widget disappear, since our conversation will only be good for this one callback.

```
$('#cb_button').click(function() {
    updateConversationToAsapCallback("7032222");
    $('#cb_widget').hide();
});
```

We can make use of the data returned by the PTK to make our UI more informative

```
function onConversationCreated(xhr) {
    interactionLocation = xhr.getResponseHeader("Location");
    json = eval('(' + xhr.responseText + ')');
    ewt = parseInt(json.originalWaitTime.split(':')[0]);
    $('#cb_button').text("Call me back in less than "+(ewt+1)+" minute(s)");
    $('#cb_widget').show();
}
```