# Office AMS: Caching Examples

|  |  |
| --- | --- |
| Summary: | Applies to: |
| This sample demonstrates some simple caching approaches using HTML5 local storage as well as HTTP Cookies. | * Office 365 Multi-Tenant (MT) * Office 365 Dedicated (D) * SharePoint 2013 on-premises |
| Solution: | Performance.Caching |
| Author: | Brian Michely (Microsoft) |
| //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // THIS CODE IS PROVIDED \*AS IS\* WITHOUT WARRANTY OF  // ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING ANY  // IMPLIED WARRANTIES OF FITNESS FOR A PARTICULAR  // PURPOSE, MERCHANTABILITY, OR NON-INFRINGEMENT.  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |

## Version log

Change log between different releases.

|  |  |
| --- | --- |
| Version | Notes |
| 1.0 | Initial |

# General comments

This sample shows how one can use a few different mechanisms to cache data and reduce the number of calls to target services. One sample uses HTML5 local storage and the other uses HTTP Cookies. The sample just reads your ‘About Me’ data from your user profile and loads it into a text area where you can make changes and ‘Save’ it for later without writing it back to your profile. This sample does not update your profile. Once you ‘Save’ it for later, the text gets stored into either a local storage key/value pair, or if using the cookies sample, it stores it into a cookie.

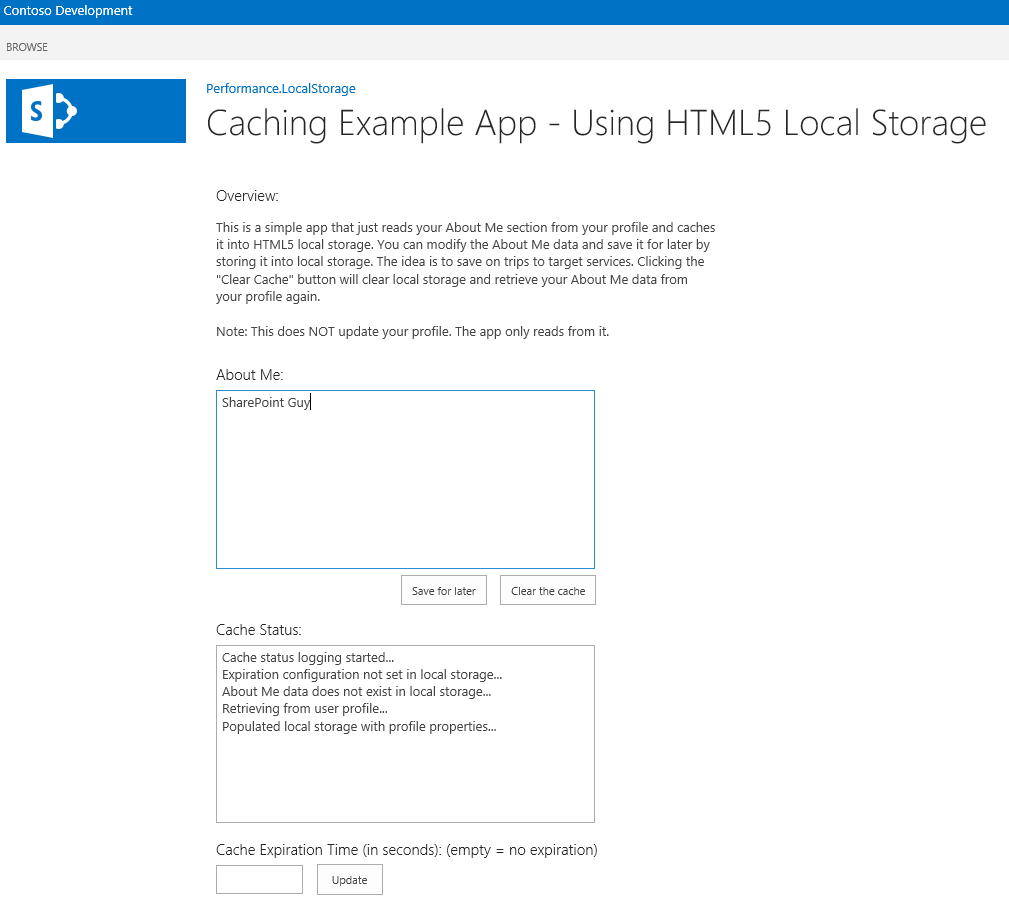
There are a few things to keep in mind. In a way, these two approaches can serve different purposes. We will just call out a few things without writing an entire article on localStorage vs. cookies.

HTML5 Local Storage data stays on the client, while cookies are sent via the header. Cookies give you a limit of 4095 bytes whereas local storage give you up to 5MB. The localStorage implementation of the Storage interface has no expiration, and so you have to clear the storage either by using javascript (which is shown in this sample) or by clearing your browser cache/local data. Also, localStorage on SSL pages will be isolated from http pages.

Be mindful of what you store in these different mechanisms. You don’t want to store things like Access Tokens, passwords or other account information.

# SCENARIo: caching with html5 localstorage

As mentioned, the sample just reads your ‘About Me’ data from your user profile and loads it into a text area where you can make changes and ‘Save’ it for later without writing it back to your profile. This sample does not update your profile. Once you ‘Save’ it for later, the text gets stored into either a local storage key/value pair, or if using the cookies sample, it stores it into a cookie.



The Cache Expiration setting will set the default number of seconds for the localStorage items. When there is a value specified, and the text in the box is saved, two keys are created/updated -> ‘aboutMe’ & ‘aboutMeTimeStamp’. When the app checks localStorage (before a service call is made), it checks to see if the key exists, and checks for and retrieves the timestamp key. It compares the time stamp against expiration settings, and if the key is expired, it makes a call to get the user profile data and refreshes/creates the two keys. If there is no value set in the Cache Expiration, the ‘aboutMe’ key does not expire.

The ‘Clear Cache’ button will bust the cache by removing the key, which will trigger a call to get your user profile data and then it will create a new key and store the ‘About Me’ data in it.

// Bust the cache

function clearCache(key) {

// Remove key/value

localStorage.removeItem(key);

// Remove key/value expiration

localStorage.removeItem(key + expiryKeySuffix);

cachingStatus += "\n" + "...";

cachingStatus += "\n" + "Cleared the cache...";

cachingStatus += "\n" + "Retrieving profile data from user profile...";

$('#status').val(cachingStatus);

// Retrieve AboutMe data from profile and repopulate text area

getUserProperties();

}

There is some functionality to manage local storage key expiration as this is not built into HTML local storage. This function will check to see if the key is expired for example.

// Check to see if the key has expired

function isKeyExpired(TimeStampKey) {

// Retrieve the example setting for expiration in seconds

var expiryStamp = localStorage.getItem(TimeStampKey);

if (expiryStamp != null && cacheTimeout != null) {

// Retrieve the timestamp and compare against specified cache timeout settings to see if it is expired

var currentTime = Math.floor((new Date().getTime()) / 1000);

if (currentTime - parseInt(expiryStamp) > parseInt(cacheTimeout)) {

return true; //Expired

}

else {

return false;

}

}

else {

//default

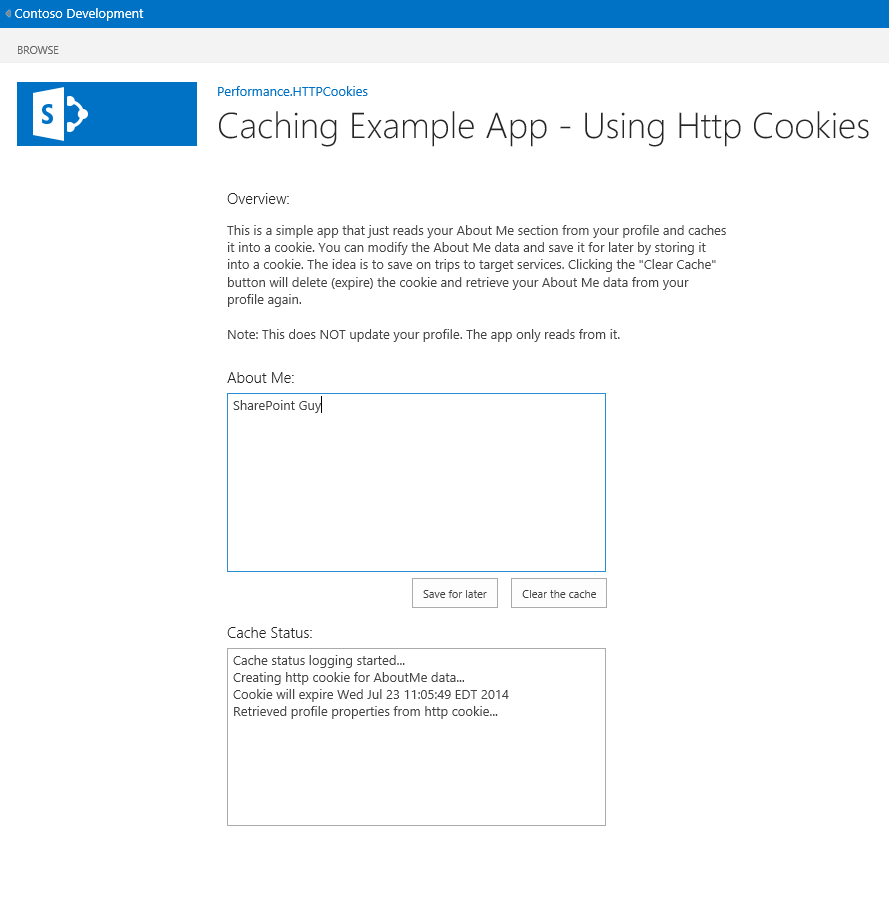
return true;

}

}

# SCENARIo: caching with http cookies

As mentioned above, the sample just reads your ‘About Me’ data from your user profile and loads it into a text area where you can make changes and ‘Save’ it for later without writing it back to your profile. This sample does not update your profile. Once you ‘Save’ it for later, the text gets stored into either a local storage key/value pair, or if using the cookies sample, it stores it into a cookie. The screenshot below shows the http cookies sample UI.



This sample will first check to see if cookies are enabled, then it will check to see if the cookie exists. If it does, it will retrieve the value from it and render in the text area. If not, it will pull from the profile, populate the cookie, set expiration and then render in UI.

# Dependencies

* Microsoft.SharePoint.Client.dll
* Microsoft.SharePoint.Client.Runtime.dll