Web Services Sample

**This sample is compatible with the June 2015 Xbox One XDK.**

# Description

This client-side sample demonstrates how to obtain an Xbox Live Security Token Service (XSTS) authorization token and signature for a signed-in Xbox Live user and how to use the token and signature to interact with either the Xbox Live RESTful services or with your own custom web service.

# Using the sample

To manually configure the console’s NSAL for development of custom services, you will need to use a temporary NSAL.json file. This file is only usable on Xbox One development kits for quick iteration and setting up the services. For retail and later in development you should set up your relying party and title endpoints through the NSAL service. This service delivers the NSAL information to the console’s in retail. This file is not required to communicate with Xbox Live services. A template NSAL.json is provided with this sample as well as a script to copy the needed files to enable it on your console (steps 3 and 4 below).

|  |
| --- |
| ***NOTE:*** *The NSAL.json file should only be used in the context of this sample for quick configuration of an HTTP endpoint. Once HTTPS is enabled on your service, only use NSAL configuration through XDP. The NSAL.json is only functional in development sandboxes and can conflict with service configuration. To remove the NSAL.json use the second script provided with this sample.* |

To update the template:

1. Set the **Host** variable to your development server endpoint (for example, **GameTestSvr1.yourdomain.com**).
2. Set the **RelyingParty** variable to the name of your relying party as set up through XDP, or the configuration spreadsheet (for example, **http://\*.yourdomain.com**), and then save the file.
3. Once you have saved the file, deploy it to the console’s S:\ with the command **xbcp NSAL.json XS:\NSAL.json**.
4. To enable the system to rely on your NSAL.json file, also copy the enforceNSAL file to the S:\

If you are using a wildcard in your Host devinition of an endpoint, use “wildcard” as the value of the **HostType** variable. Otherwise use “fqdn” for a Fully Qualified Domain Name as in the example.

When calling a custom service URL, if you see **AUTH\_ERROR** in the UI, then your NSAL.json file might not be set up properly or your server’s relying party certificate might not be ingested into the Xbox Live service.

By default, to demonstrate calling a RESTful service with the XSTS token, the sample is already set up and enabled to call the Xbox People service. To enable the sample to use your own custom service calls, update the URLs to the host name or IP of your development or test server.

When the response from a service call has been received, the sample will output the response and the HTTP status code to the screen.

|  |  |
| --- | --- |
| Action | Gamepad Control |
| Call the Xbox People service and request a token for the service (this action will work with any signed-in account) | Y button |
| Call a custom service URL and request a token for the service (use this action with the provided sample server) | A button |
| Call the custom getclaims service and request a token for the service (use this action with the provided sample server) | X button |
| Open up the Account Picker for sign-in / sign-out | Left bumper |
| Display Help | BACK button |
| Exit the sample | Left trigger + right trigger + right bumper |

# Implementation notes

This sample requires that a user be signed-in in order to make the service calls. To sign in a user, press the Left Bumper on the controller to open up the account picker and select or recover a user account. For more information on setting up a user account, refer to the Xbox One XDK documentation topic **Xbox Live Authentication**.

In this version of the XDK, the sample also requires your Xbox One console to use the XDKS.1 **SandboxID**. To switch to this **SandboxID**, in the Xbox One XDK command prompt, use:

xbconfig sandboxid=XDKS.1

The sample uses an **XboxSampleFramework::HTTPRequest** object to build the HTTP request, to obtain the necessary token and signature, and then to send the request. To see code flow of how to implement this object, refer to the **HTTPRequest::Open** API in HTTPRequest.cpp.

Retrieve the XSTS token and signature by calling the following API:

[IAsyncOperation](https://developer.xboxlive.com/en-us/platform/development/documentation/software/Pages/IAsyncOperation%601_WF_T_mar2013.aspx)<[GetTokenAndSignatureResult](https://developer.xboxlive.com/en-us/platform/development/documentation/software/Pages/GetTokenAndSignatureResult_MXS_T_mar2013.aspx)>^ GetTokenAndSignatureAsync(

[String](http://msdn.microsoft.com/en-us/library/hh755812.aspx) ^*url*,

[String](http://msdn.microsoft.com/en-us/library/hh755812.aspx) ^*headers*,

[Array<unsigned char*>*](http://msdn.microsoft.com/en-us/library/hh755812.aspx) *body*)

**Note**   The **body** parameter must be at least a single-byte array set to zero—even if there is no body to the request call. Also, the **headers** parameter must be a single string of all the headers that will be used in calculating the returned signature. The format of the string must be: Header1: Value\r\nHeader2: Value\r\n.

Because this API runs asynchronously, the sample demonstrates how you can set the code to wait for the results and then how to copy the results once they are returned. The token needed for authentication is then retrieved from **GetResults()->Token** and the signature from **GetResults()->Signature.**

The XSTS token is added to the HTTP request with the **Authorization** header:

Authorization: <XSTS Token>

**Note**   In previous versions of the XDK, you were required to insert **XBL3.0 x=** before the token for the authorization header. This declaration is no longer required, because the token that you retrieve from **GetTokenAndSignatureAsync** will include that declaration.

The signature is also added to the HTTP request as a header:

Signature: <Signature>

**Note**   In the April XDK, the signature returned will be a blank string. In a future XDK, the string returned will be an actual signature. For now, Xbox Live Service calls do not require a signature. The actual header name is also subject to change in a future XDK when signatures are accepted.

Alternatively, the Xbox One System will retrieve and add the token and signature headers for you if you add the following HTTP header to a web service request:

xbl-authz-actor-10: [XboxUserHash]\r\n

The XboxUserHash is the user the call is on behalf of and can be retrieved with the following:

User::Users->GetAt( m\_userIndex )->XboxUserHash->Data()

The Web Services Sample uses **IXMLHTTPRequest2 COM** objects to complete the call to the specified service.

# Known issues

If the sample has been inactive for a long time on the console, the sample might not send out additional service requests or it might appear that a service request is still pending a return. Simply close the sample and restart it.

This sample currently does not support Xbox One sensor input.