Announcing the new SharePointContext helper in Apps for SharePoint 2013

Rate This

☆☆☆☆☆

SharePoint Developer Team

SharePoint Developer Team

Microsoft

19,472 Points 8 3 2

Recent Achievements

Profile Complete Gallery Contributor II New Gallery Contributor

View Profile

7 Nov 2013 7:53 AM

- Comments 1

In Visual Studio 2013, a new SharePointContext helper is available for app for SharePoint projects. This blog post will review the features of this new helper and offer some tips about how to use it.

For those of you developing apps for SharePoint with Visual Studio 2012, you may be familiar with another helper class, TokenHelper. TokenHelper is designed to simplify the authentication and authorization flow between remote web and SharePoint (also called server-to-server flow) in their autohosted or provider-hosted apps. It provides a set of APIs for developers to complete the complex authentication flow in several simple steps. However, when authoring a real-world app for SharePoint, you may quickly find that you have to handle more problems. For example:

- How to manage the lifetime of the parameters passed from SharePoint, such as SPHostUrl or refresh token?
- How to write an app that can run under different authentication modes without a code change?
- How to handle app URL bookmarking?
- When to renew access token?

The new SharePointContext helper is designed to solve these problems. It is built on TokenHelper, attempting to provide a simplified and unified context model for apps to communicate with SharePoint.

The SharePointContext helper consists of two classes:

SharePointContext

SharePointContextProvider

The SharePointContext helper consists of four subclasses (for authentication):

SharePointAcsContext

SharePointAcsContextProvider

SharePointHighTrustContext

SharePointHighTrustContextProvider

The next section will introduce features of the SharePointContext helper.

Unified Context Model

The new context model isolates the authentication layer so that developers can focus on business logic in the code without having to worry about authentication details. The following code will work whether your app uses Access Control Service (low-trust) or certificates (high-trust) for authentication.

```
var spContext =
    SharePointContextProvider.Current.GetSharePointContext(HttpContext);

using (var clientContext = spContext.CreateUserClientContextForSPHost())
{
    if (clientContext != null)
    {
        User spUser = clientContext.Web.CurrentUser;
        clientContext.Load(spUser, user => user.Title);
        clientContext.ExecuteQuery();
    }
}
```

This is because the SharePointContext helper has built-in support for OAuth2 using ACS (SharePointAcsContext) and high-trust authentication using certificates (SharePointHighTrustContext). When an app runs, the SharePointContext helper will automatically detect whether it runs with ACS or with certificates and then initialize either SharePointAcsContextProvider or SharePointHightTrustContextProvider as the default context provider. You can check the static constructor of SharePointContextProvider to see how it works. If you want your app to support other authentication with SharePoint, just register your custom provider for SharePointContext through SharePointContextProvider.Register().

The SharePointContext helper is also integrated with both ASP.NET Web Forms and ASP.NET MVC, so you can use the features of the helper no matter whether you're creating an MVC app or a Web Forms app. However, ASP.NET Web API is not on the supported list because it doesn't share the web application stack with ASP.NET Web Forms and ASP.NET MVC.

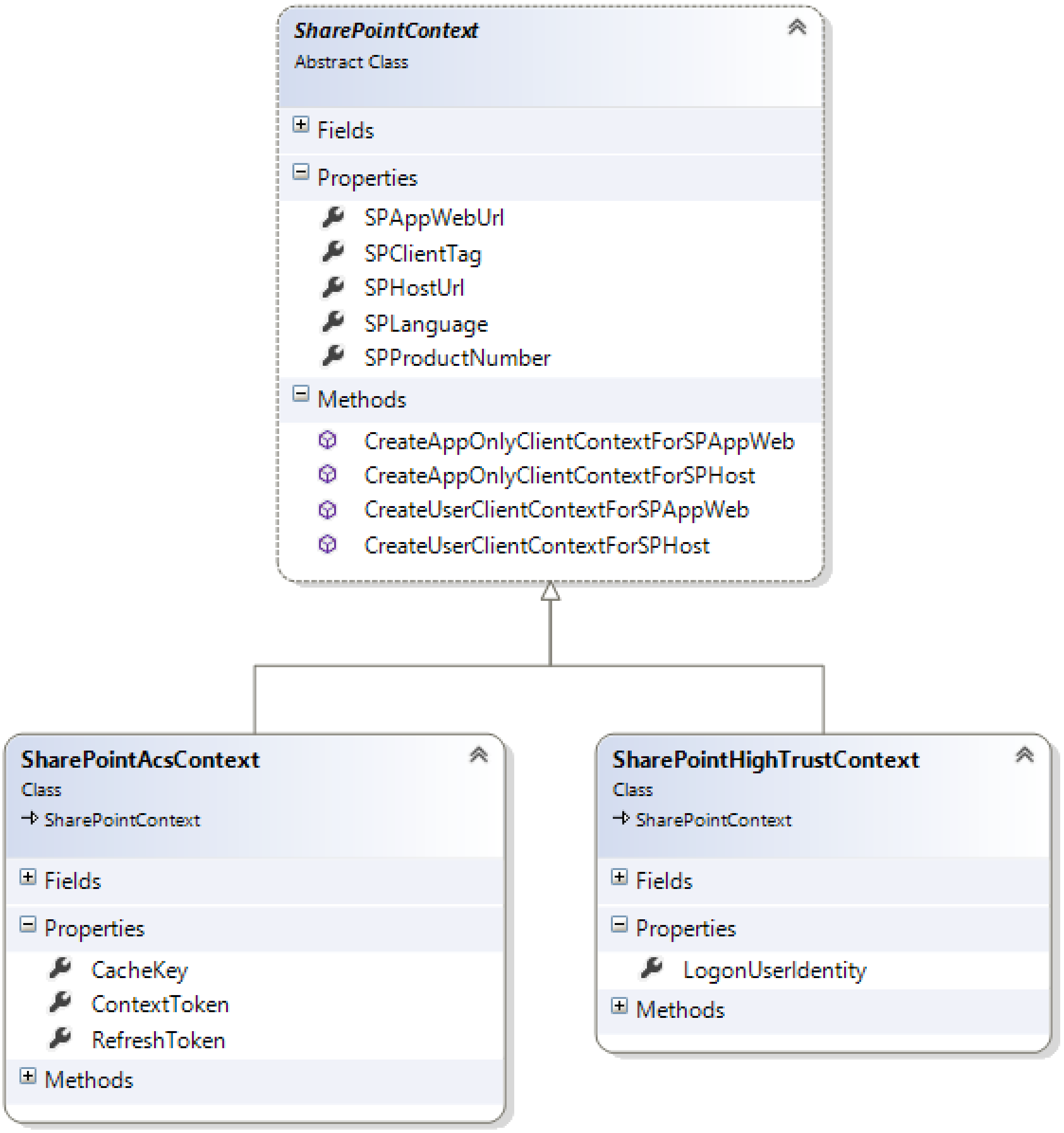

Figure 1. SharePointContext class diagram

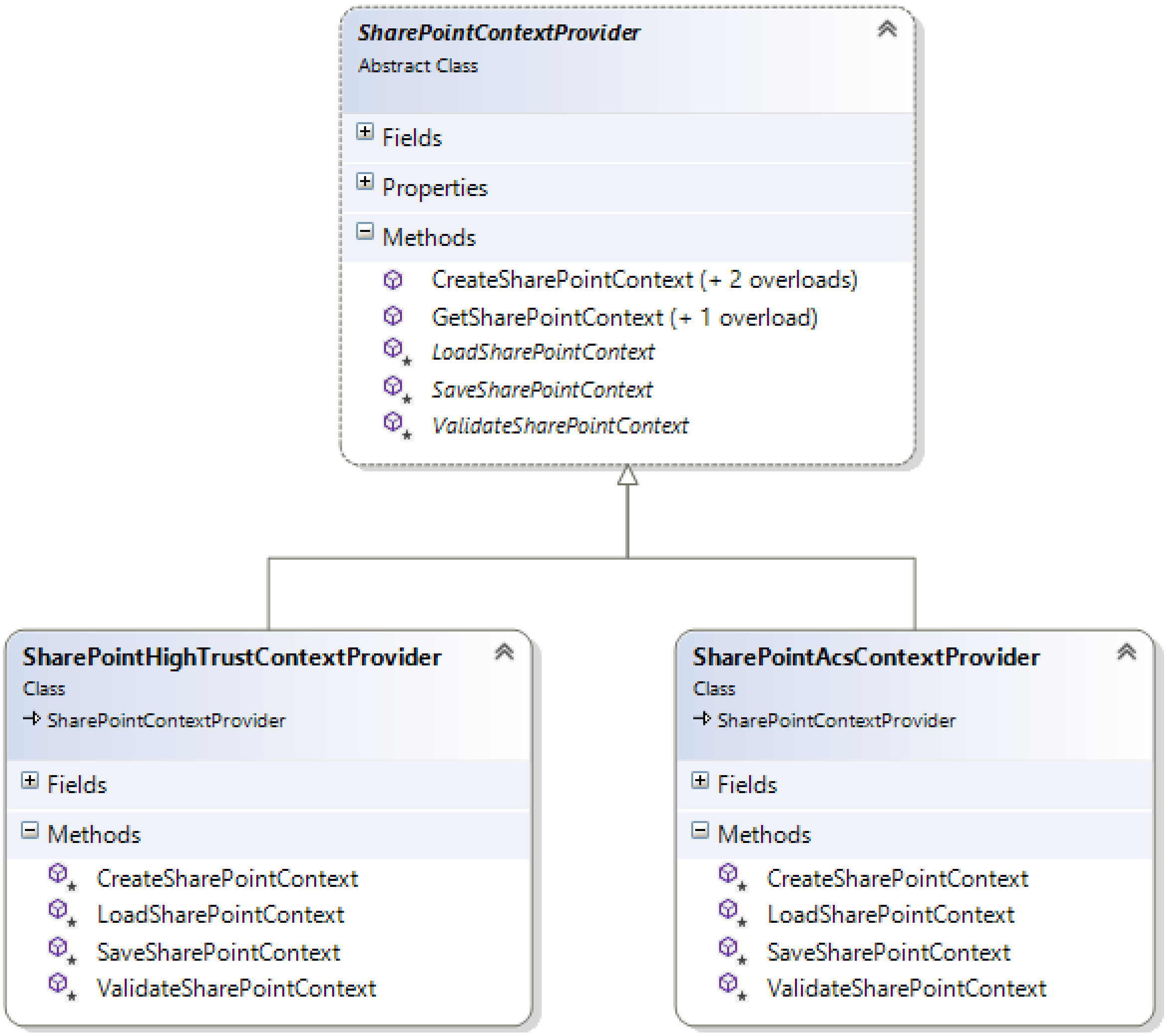
SharePointContextProvider
Abstract Class
Fields
Properties
Methods
CreateSharePointContext (+ 2 overloads)
GetSharePointContext (+ 1 overload)
LoadSharePointContext
SaveSharePointContext
ValidateSharePointContext
SharePointHighTrustContextProvider
Class
SharePointContextProvider
Fields
Methods
CreateSharePointContext
LoadSharePointContext
SaveSharePointContext
ValidateSharePointContext
SharePointAcsContextProvider
Class
SharePointContextProvider
Fields
Methods
CreateSharePointContext
LoadSharePointContext
SaveSharePointContext
ValidateSharePointContext

Figure 2. SharePointContextProvider class diagram

Standard Tokens

A SharePointContext instance is initialized with five read-only properties: SPHostUrl, SPAppWebUrl, SPClientTag, SPLanguage, and SPProductNumber. These properties are passed from a SharePoint host site as part of the query string of app start URL if {StandardTokens} is embedded in the start URL of app manifest. SharePointContext always requires SPHostUrl and some of the other properties to be non-empty, so it's strongly recommended to always append {StandardTokens} as part of the query string to app start URL in app manifest. With these properties correctly set in the context, your app is capable of providing a user experience consistent with the SharePoint host site. For example, by knowing the display language of the SharePoint host site via SPLanguage, your app could display content in the same language as its host site.

App URL Bookmarking

Let's say users love your app, so they bookmark your app by adding the app URL to favorites in their browser. Next time, when they launch the app from the favorites, the first thing your app needs to do is create a new SharePointContext instance to serve this browser session. The new SharePointContext helper has a method called SharePointContextProvider.CheckRedirectionStatus(). It verifies that the current user has authenticated with SharePoint or not. If not, it will construct a redirect URL to let the browser navigate the user to the SharePoint host site to authenticate first.

If you're developing an MVC app, you only need to put the attribute SharePointContextFilterAttribute on the controllers or on the actions that need to communicate with SharePoint. The attribute will verify that a valid SharePointContext has been created before the action is executed by calling SharePointContextProvider.CheckRedirectionStatus().

```
public class SharePointContextFilterAttribute : ActionFilterAttribute
{
    public override void OnActionExecuting(
        ActionExecutingContext filterContext)
    {
        if (filterContext == null)
        {
            throw new ArgumentNullException("filterContext");
        }

        Uri redirectUrl;
        switch (SharePointContextProvider.CheckRedirectionStatus(
            filterContext.HttpContext, out redirectUrl))
        {
            case RedirectionStatus.Ok:
                return;
            case RedirectionStatus.ShouldRedirect:
                filterContext.Result =
                  new RedirectResult(redirectUrl.AbsoluteUri);
                break;
            case RedirectionStatus.CanNotRedirect:
                filterContext.Result = new ViewResult{ ViewName = "Error" };
                break;
        }
    }
}
```

If you're developing a Web Forms app, no problem! SharePointContextFilterAttribute can be mimicked by Page_PreInit() in a Web Forms page like this.

```
protected void Page_PreInit(object sender, EventArgs e)
{
    Uri redirectUrl;
    switch (SharePointContextProvider.CheckRedirectionStatus(
                Context, out redirectUrl))
    {
        case RedirectionStatus.Ok:
            return;
        case RedirectionStatus.ShouldRedirect:
            Response.Redirect(redirectUrl.AbsoluteUri, endResponse: true);
            break;
        case RedirectionStatus.CanNotRedirect:
            Response.Write("Error occurred while processing your request.");
            Response.End();
            break;
    }
}
```

For provider-hosted apps that support multi-tenancy, here is one more tip. For your app to distinguish which request comes from which SharePoint tenant, SPHostUrl should be always appended as part of the query string to any app URLs. The existence of SPHostUrl is important to the SharePointContext helper because it can know which SharePoint host site the user should be redirected to for authentication if the context has not been created.

Renew Access Token

For the apps authenticating with ACS, an access token for SharePoint stays valid for 12 hours once issued, while a refresh token stays valid for six months. For the apps authenticating with certificate (high-trust), technically the lifetime of an access token could be any non-negative number, as it is self-issued by your app. TokenHelper sets the lifetime as 12 hours so it keeps consistent with ACS. When an access token expires, the remote web needs to acquire a new access token before it can continue to access the protected resources on SharePoint. Instead of renewing access token after it expires, the SharePointContext helper adopts a more proactive strategy, which is to renew access token once it finds the token will expire within five minutes. This ensures the access token returned by the SharePointContext helper is usable for at least five minutes, which should be long enough for completing most tasks accessing SharePoint resources.

Extensibility

The SharePointContext helper is designed to be extensible. SharePointContext and SharePointContextProvider are independently customizable. There may be two typical scenarios that you want to extend the existing SharePointContext and SharePointContextProvider or their subclasses:

1. Adopt an advanced caching strategy for SharePointContext.

The default implementation of SharePointAcsContextProvider and SharePointHighTrustContextProvider caches SharePointContext instances in HttpSessionState, which by default stores the data in-memory on a single compute-node.

```
protected override SharePointContext LoadSharePointContext(
    HttpContextBase httpContext)
{
    return httpContext.Session[SPContextKey] as SharePointContext;
}

protected override void SaveSharePointContext(
    SharePointContext spContext, HttpContextBase httpContext)
{
    httpContext.Session[SPContextKey] = spContext as SharePointContext;
}
```

If the remote web application of your app runs in the cloud such as Windows Azure with multiple compute-nodes, you may want to share the SharePoint Context data across all the compute-nodes. To achieve this, you could inherit SharePointContextProvider or one of its subclasses to override LoadSharePointContext() and SaveSharePointContext() and decide what properties in SharePointContext to cache and share.

2. Enable your app to work with SharePoint sites that use other authentications.
If you're developing an app that will work with a SharePoint site using SAML authentication, for example, you will likely need a new SharePointSamlContext inheriting from SharePointContext and a new SharePointSamlContextProvider inheriting from SharePointContextProvider. Although the implementation could vary depending on how the authentication works, the unified context model should have enough flexibility to add a new authentication mechanism to your app without having to modify it.

So how to get these cool features? If you're creating a new provider-hosted or autohosted app for SharePoint in Visual Studio 2013, you will have the new SharePointContext helper, by default, in the web project which is part of the app for SharePoint. In addition, the new SharePointContext helper has also been included in the NuGet package AppForSharePointWebToolkit version 3.0. Hence, if you have an existing app for SharePoint project created in Visual Studio 2012 or Visual Studio 2013 Preview/RC or earlier, you can get the latest version of the SharePointContext helper by updating the NuGet package. Now, go use it in your app and let us know what you think!

David Zhao,
Software Development Engineer, Visual Studio