# How to: Implement an invitation flow

This article describes how the invitation flow is implemented for the Wingtip Games site.

Contents

[Wingtip scenario 1](#_Toc488096430)

[How to: Send the invitation message 6](#_Toc488096431)

[Create the authentication challenge 6](#_Toc488096432)

[Generate the input claims 7](#_Toc488096433)

[Send the policy link 11](#_Toc488096434)

[Skip the cross-site request forgery (XSRF) 12](#_Toc488096435)

[How to: Configure the relying party policy 13](#_Toc488096436)

[How to: Configure the base policy 14](#_Toc488096437)

[Configure the extension\_VerifiedEmail claim type 14](#_Toc488096438)

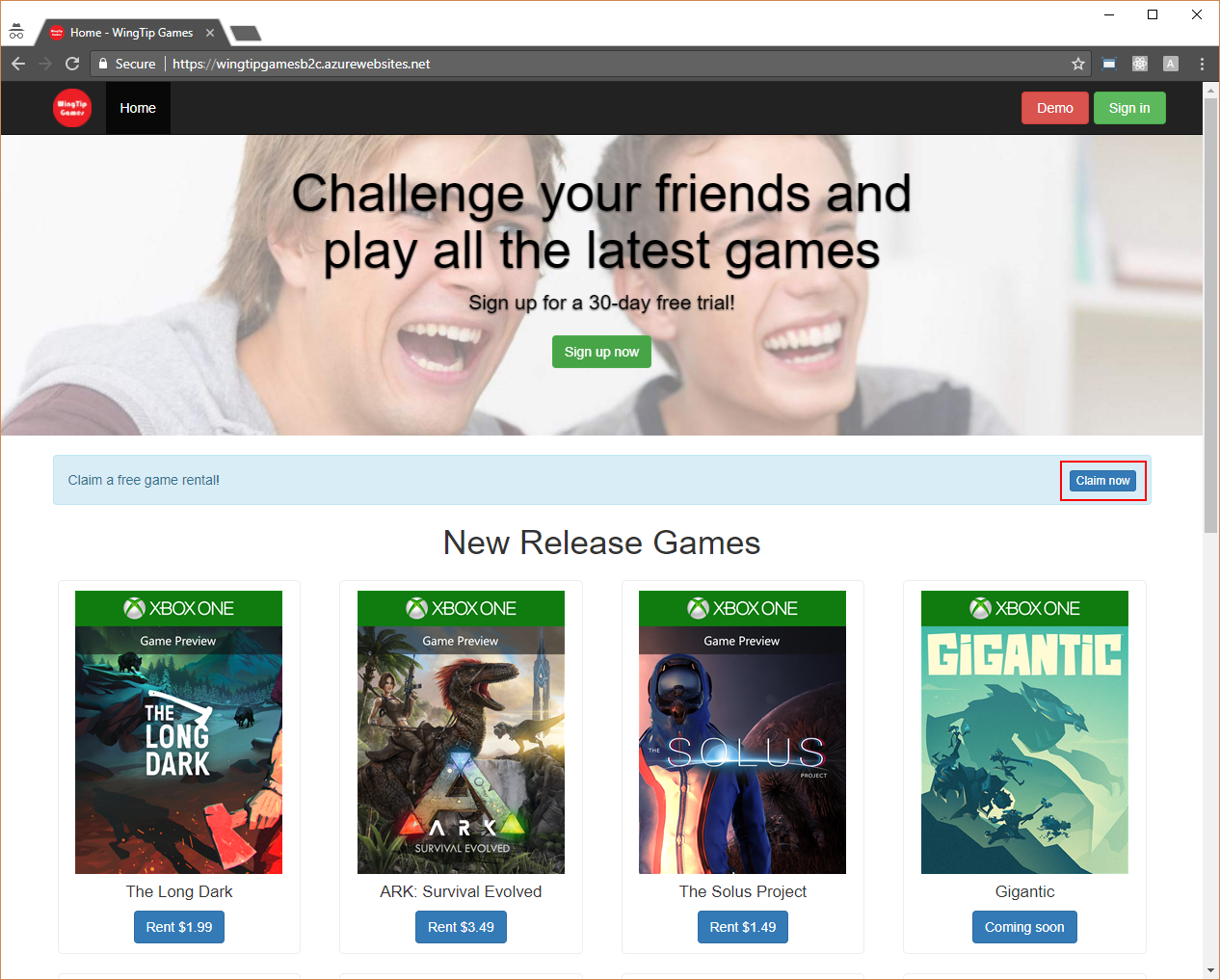
[Configure the CreateEmailFromVerifiedEmail claims transformation 15](#_Toc488096439)

[Configure the LocalAccount-Registration-VerifiedEmail technical profile for the Local Account claims provider 15](#_Toc488096440)

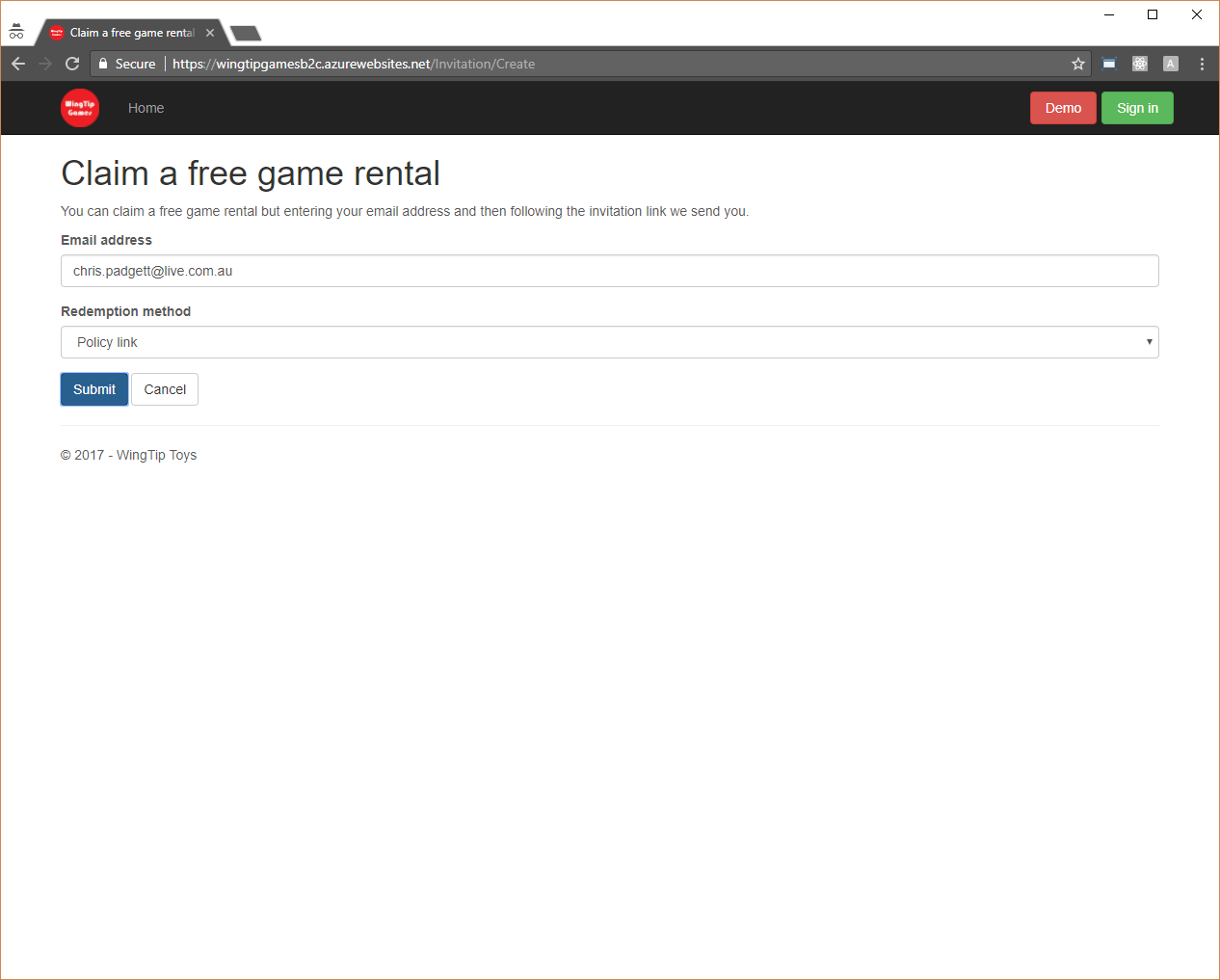
[Configure the Invitation user journey 16](#_Toc488096441)

## Wingtip scenario

Wingtip is offering free game rentals to anonymous users. An end user browses the Wingtip Games site and requests a free game rental by submitting their email address.

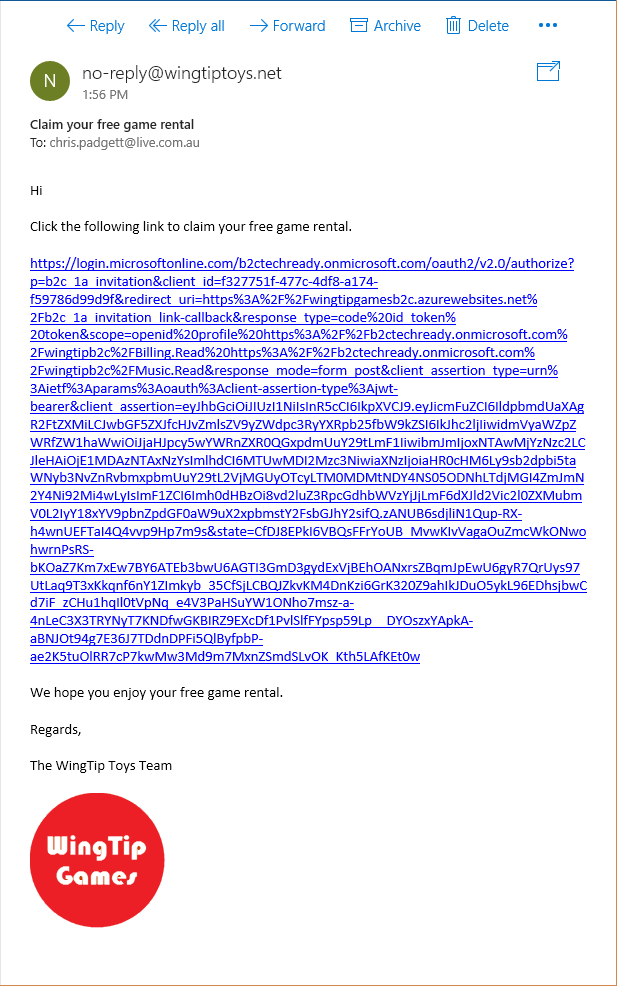


*Figure: An anonymous user browses the Wingtip Games site.*

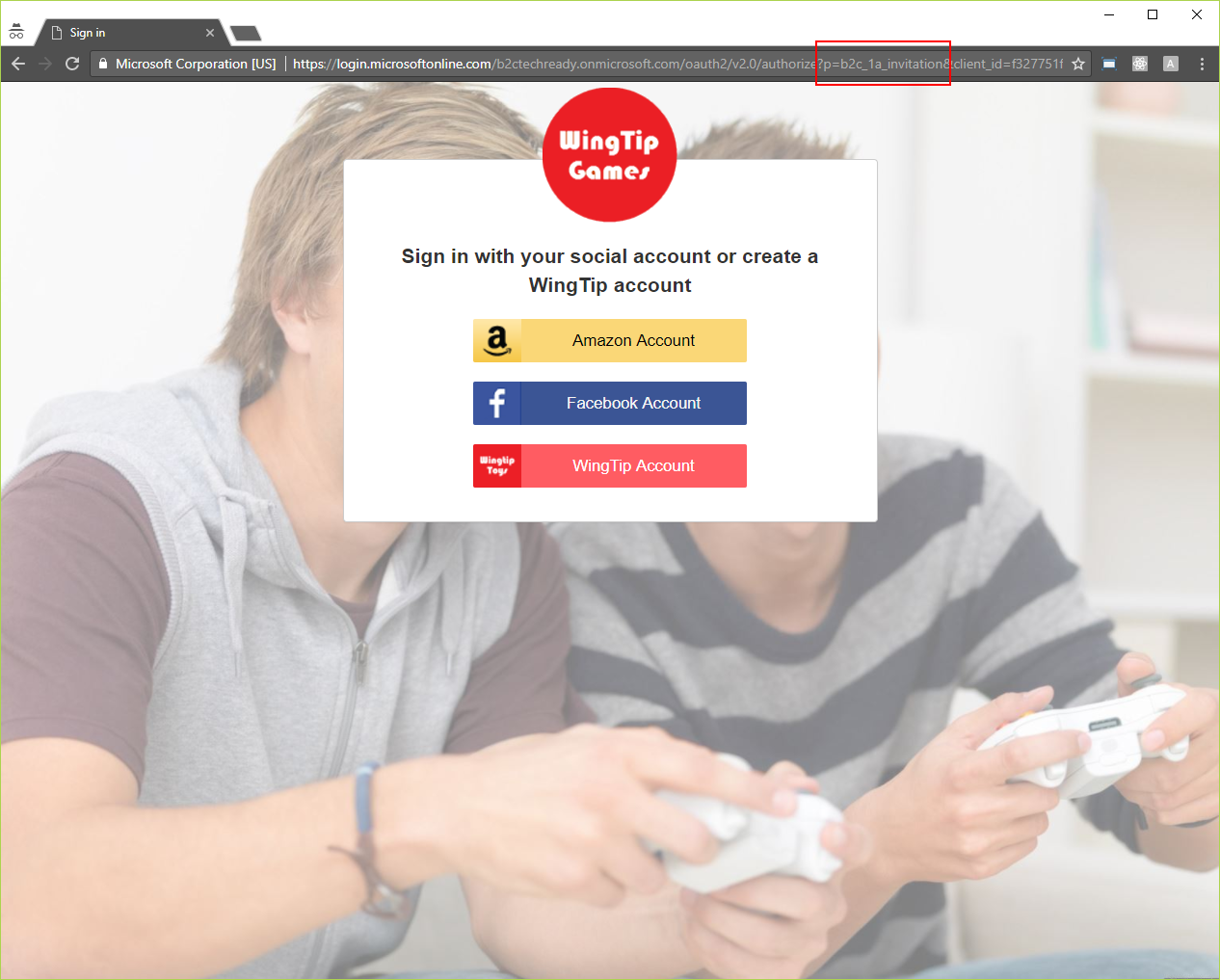


*Figure: The anonymous user requests a free game rental by submitting their email address.*

The Wingtip Games site sends an invitation message, which contains a policy link, to this email address. This policy link, which is generated for the **b2c\_1a\_invitation** policy, is appended with a signed JWT that includes the email address.

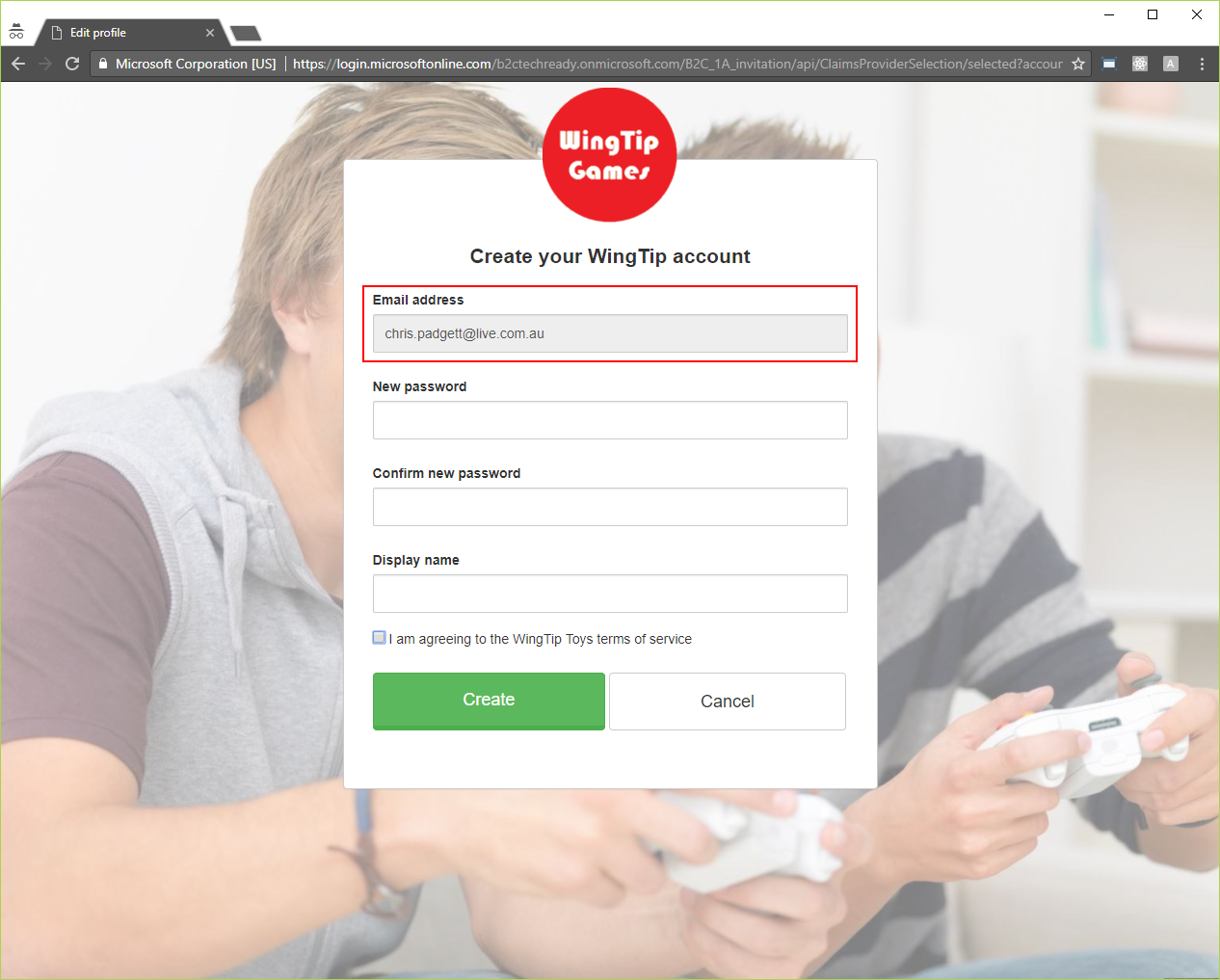


*Figure: The Wingtip Games site sends an invitation message, which contains a policy link, to this email address.*



*Figure: This policy link, which is generated for the* ***b2c\_1a\_invitation*** *policy, is appended with a signed JWT that includes the email address.*

The end user then registers with their email address.



*Figure: The end user then registers with their email address.*

## How to: Send the invitation message

The Wingtip Games site sends the invitation message as follows:

1. Creates the authentication challenge for the **b2c\_1a\_invitation** policy.
2. Generates the input claims for the **b2c\_1a\_invitation** policy.
3. Sends the policy link for the **b2c\_1a\_invitation** policy.
4. Skips the cross-site request forgery (XSRF) logic for the **b2c\_1a\_invitation** policy.

### Create the authentication challenge

The following code describes how the authentication challenge for the **b2c\_1a\_invitation** policy is created by the Wingtip Games site.

namespace WingTipGamesWebApplication.Controllers  
{  
 public class InvitationController : Controller  
 {  
   
 ...  
   
 [HttpPost]  
 public async Task<IActionResult> Create(CreateViewModel viewModel)  
 {  
 if (viewModel == null)  
 {  
 throw new ArgumentNullException(nameof(viewModel));  
 }  
   
 if (ModelState.IsValid)  
 {  
 var authenticationProperties = new AuthenticationProperties();  
 authenticationProperties.Items["player\_profile\_registration\_mode"] = "Basic";  
 // Set the invitation lifetime to 1 day.  
 authenticationProperties.Items["policy\_token\_lifetime"] = InvitationTokenLifetime.ToString();  
 // The end user might open the redemption link in a different browser or session so disable the cross-site request  
 // forgery (XSRF) logic in the OpenID Connect authentication middleware.  
 authenticationProperties.Items["skip\_correlation"] = true.ToString();  
 // Set the email address of the invited user.  
 authenticationProperties.Items["verified\_email"] = viewModel.EmailAddress;  
 authenticationProperties.RedirectUri = Url.Action("Redeemed", "Invitation");  
 await HttpContext.Authentication.ChallengeAsync("b2c\_1a\_invitation", authenticationProperties);  
 return View("Created");  
 }  
   
 return View(viewModel);  
 }  
   
 ...  
   
 }

}

This challenge is created as follows:

* The **player\_profile\_registration\_mode** property is set to “Basic” so the end user is registered with a basic player profile.
* The **policy\_token\_lifetime** property is set to 1 day so the policy link is expired after this period of time.
* Because the end user might open the policy link in a different browser or session, the **skip\_correlation** property is set to **true**, so the Wingtip Games site skips the cross-site request forgery (XSRF) logic for the policy link.
* The **verified\_email** property is set to the email address of the end user so the end user is registered with this email address.
* The **redirect\_uri** property is set to the https://wingtipgamesb2c.azurewebsites.net/Invitation/Redeemed URL so the end user is displayed the free game rental after they have registered.

### Generate the input claims

The following code describes how the input claims for the **b2c\_1a\_invitation** policy are generated by the Wingtip Games site.

namespace WingTipGamesWebApplication  
{  
 public class Startup  
 {  
   
 ...  
   
 private static void ConfigureOpenIdConnectAuthenticationOptions(  
 IApplicationBuilder applicationBuilder,  
 string authenticationScheme,  
 IConfigurationSection openIdConnectAuthenticationConfigurationSection,  
 string policyId,  
 bool requireNonce,  
 ICollection<Claim> staticPolicyClaims)  
 {  
  
 ...  
  
 var openIdConnectAuthenticationOptions = new OpenIdConnectOptions  
 {  
  
 ...  
  
 Events = new OpenIdConnectEvents  
 {  
  
 ...  
  
 OnRedirectToIdentityProvider = async context =>  
 {  
 ...  
  
 // Initialize a instance-level list of claims to be sent.  
 var instancePolicyClaims = new List<Claim>();  
  
 // If an Account controller action has set a player profile registration mode, then add the "player\_profile\_registration\_mode"  
 // claim to this list of claims to be sent.  
 if (context.Properties.Items.ContainsKey(Constants.AuthenticationPropertiesKeys.PlayerProfileRegistrationMode))  
 {  
 var playerProfileRegistrationModeClaim = new Claim("player\_profile\_registration\_mode", context.Properties.Items[Constants.AuthenticationPropertiesKeys.PlayerProfileRegistrationMode]);  
 instancePolicyClaims.Add(playerProfileRegistrationModeClaim);  
 context.Properties.Items.Remove(Constants.AuthenticationPropertiesKeys.PlayerProfileRegistrationMode);  
 }  
  
 // If an Invitation controller action has set a verified email address, then add the "verified\_email" claim  
 // to this list of claims to be sent.  
 if (context.Properties.Items.ContainsKey(Constants.AuthenticationPropertiesKeys.VerifiedEmail))  
 {  
 var verifiedEmailClaim = new Claim("verified\_email", context.Properties.Items[Constants.AuthenticationPropertiesKeys.VerifiedEmail]);  
 instancePolicyClaims.Add(verifiedEmailClaim);  
 context.Properties.Items.Remove(Constants.AuthenticationPropertiesKeys.VerifiedEmail);  
 }  
  
 // Initialize the list of claims to be sent.  
 var policyClaims = new List<Claim>();  
  
 // Add the static-level list of claims to the list of claims to be sent.  
 if (staticPolicyClaims != null && staticPolicyClaims.Any())  
 {  
 policyClaims.AddRange(staticPolicyClaims);  
 }  
  
 // Add the instance-level list of claims to the list of claims to be sent.  
 if (instancePolicyClaims != null && instancePolicyClaims.Any())  
 {  
 policyClaims.AddRange(instancePolicyClaims);  
 }  
  
 // If a list of claims is to be sent to the target policy, then...  
 if (policyClaims != null && policyClaims.Any())  
 {  
 var configuration = await context.Options.ConfigurationManager.GetConfigurationAsync(CancellationToken.None);  
  
 TimeSpan policyTokenLifetime;  
  
 // Get the lifetime of the JSON Web Token (JWT) from the authentication session...  
 if (!context.Properties.Items.ContainsKey(Constants.AuthenticationPropertiesKeys.PolicyTokenLifetime) || !TimeSpan.TryParse(context.Properties.Items[Constants.AuthenticationPropertiesKeys.PolicyTokenLifetime], out policyTokenLifetime))  
 {  
 // ... Or set it to a default time of 5 minutes.  
 policyTokenLifetime = new TimeSpan(0, 0, 5, 0);  
 }  
  
 // Create the JWT containing the list of claims and signed by the client secret.  
 var selfIssuedToken = CreateSelfIssuedToken(  
 configuration.Issuer,  
 context.ProtocolMessage.RedirectUri,  
 policyTokenLifetime,  
 context.Options.ClientSecret,  
 policyClaims);  
  
 // Set the client assertion parameter for the authentication request to this JWT.  
 // The list of claims is received by the target policy as follows:  
 // <InputTokenFormat>JWT</InputTokenFormat>  
 // <CryptographicKeys>  
 // <Key Id="client\_secret" StorageReferenceId="WingTipGamesClientSecret" />  
 // </CryptographicKeys>  
 // <InputClaims>  
 // <InputClaim ClaimTypeReferenceId="extension\_Brand" />  
 // <InputClaim ClaimTypeReferenceId="extension\_PlayerProfileRegistrationMode" />  
 // </InputClaims>  
 context.ProtocolMessage.Parameters.Add("client\_assertion\_type", "urn:ietf:params:oauth:client-assertion-type:jwt-bearer");  
 context.ProtocolMessage.Parameters.Add("client\_assertion", selfIssuedToken);  
 }  
  
 // TODO: Send the policy link.  
  
 context.HandleResponse();  
 },  
  
 ...  
  
 },  
  
 ...  
  
 };  
  
 // The following OpenID Connect authentication middleware allows the cross-site request forgery logic to be disabled for  
 // when a policy link is sent to an end user using an email message.  
 applicationBuilder.UseWingTipOpenIdConnectAuthentication(openIdConnectAuthenticationOptions);  
 }  
   
 ...  
   
 }  
}

These claims are generated as follows:

1. Any “static” claims, i.e. those which are passed to all executions of the **b2c\_1a\_invitation** policy, are passed as arguments to the **ConfigureOpenIdConnectAuthenticationOptions** method.
2. Any “instance” claims, i.e. those which are passed to this execution of the **b2c\_1a\_invitation** policy, are passed as properties, which are created in the previous section, to the **OnRedirectToIdentityProvider** event.
3. All claims are passed in a signed JWT to the **b2c\_1a\_invitation** policy.

This JWT is generated by creating a security token with the following properties:

* + The **Issuer** property is set to the issuer URI of the **b2c\_1a\_invitation** policy.
  + The **Audience** property is set to the redirection endpoint URI of the Wingtip Games site.
  + The **IssuedAt** and **NotBefore** properties are set to the current time.
  + The **Expires** property is set to a future time.
  + The **Subject** property is set to the input claims.
  + The **SigningCredentials** property is set to the client secret of the Wingtip Games site.

This security token is then signed with the client secret of the Wingtip Games site.

The following code describes how the JWT is generated.

namespace WingTipGamesWebApplication  
{  
 public class Startup  
 {  
   
 ...  
   
 internal static string CreateSelfIssuedToken(  
 string issuer,  
 string audience,  
 TimeSpan expiration,  
 string signingSecret,  
 ICollection<Claim> claims)  
 {  
 var tokenHandler = new JwtSecurityTokenHandler();  
 var nowUtc = DateTime.UtcNow;  
 var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(signingSecret));  
 var signingCredentials = new SigningCredentials(key, "http://www.w3.org/2001/04/xmldsig-more#hmac-sha256");  
  
 var tokenDescriptor = new SecurityTokenDescriptor  
 {  
 Audience = audience,  
 Expires = nowUtc.Add(expiration),  
 IssuedAt = nowUtc,  
 Issuer = issuer,  
 NotBefore = nowUtc,  
 SigningCredentials = signingCredentials,  
 Subject = new ClaimsIdentity(claims)  
 };  
  
 var token = tokenHandler.CreateToken(tokenDescriptor);  
 return tokenHandler.WriteToken(token);  
 }  
   
 ...  
   
 }  
}

The following snippet shows the JWT payload that is serialized.

{  
 "brand": "WingTip Games",  
 "player\_profile\_registration\_mode": "Basic",  
 "verified\_email": "chris.padgett@live.com.au",  
 "nbf": 1500263776,  
 "exp": 1500350176,  
 "iat": 1500263776,  
 "iss": "https://login.microsoftonline.com/ec0e2972-3403-4685-983a-7c0b8fbf7f86/v2.0/",  
 "aud": "https://wingtipgamesb2c.azurewebsites.net/b2c\_1a\_invitation-callback"  
}

### Send the policy link

The following code describes how the policy link for the **b2c\_1a\_invitation** policy is sent by the Wingtip Games site.

namespace WingTipGamesWebApplication  
{  
 public class Startup  
 {  
   
 ...  
   
 private static void ConfigureOpenIdConnectAuthenticationOptions(  
 IApplicationBuilder applicationBuilder,  
 string authenticationScheme,  
 IConfigurationSection openIdConnectAuthenticationConfigurationSection,  
 string policyId,  
 bool requireNonce,  
 ICollection<Claim> staticPolicyClaims)  
 {  
  
 ...  
  
 var openIdConnectAuthenticationOptions = new OpenIdConnectOptions  
 {  
  
 ...  
  
 Events = new OpenIdConnectEvents  
 {  
  
 ...  
  
 OnRedirectToIdentityProvider = async context =>  
 {  
 ...  
  
 // DONE: Generate the input claims.  
  
 // Redirect to the identity provider via an email link that contains the authentication request URL.  
 context.Properties.Items.Add(OpenIdConnectDefaults.RedirectUriForCodePropertiesKey, context.ProtocolMessage.RedirectUri);  
 context.ProtocolMessage.State = context.Options.StateDataFormat.Protect(context.Properties);  
 var smtpService = context.HttpContext.RequestServices.GetRequiredService<ISmtpService>();  
 var verifiedEmailClaim = policyClaims.First(c => c.Type == "verified\_email");  
 var authenticationRequestUrl = context.ProtocolMessage.CreateAuthenticationRequestUrl();  
 smtpService.SendInvitationEmail(verifiedEmailClaim.Value, authenticationRequestUrl);  
  
 context.HandleResponse();  
 },  
  
 ...  
  
 },  
  
 ...  
  
 };  
  
 ...  
  
 }  
   
 ...  
   
 }  
}

This link is sent by creating the authentication request and sending it to the email address of the end user.

The following snippet shows the authentication request that is created.

https://login.microsoftonline.com/b2ctechready.onmicrosoft.com/oauth2/v2.0/authorize  
 ?p=b2c\_1a\_invitation  
 &client\_id=f327751f-477c-4df8-a174-f59786d99d9f  
 &redirect\_uri=https%3A%2F%2Fwingtipgamesb2c.azurewebsites.net%2Fb2c\_1a\_invitation-callback  
 &response\_type=code%20id\_token  
 &scope=openid%20profile  
 &response\_mode=form\_post  
 &client\_assertion\_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer  
 &client\_assertion=eyJhbGci...7m9s  
 &state=CfDJ8EPk...Et0w

The **client\_assertion\_type** parameter is set to “urn:ietf:params:oauth:client-assertion-type:jwt-bearer” and the **client\_assertion** parameter is set to the JWT that is generated in the previous section

### Skip the cross-site request forgery (XSRF)

Because the end user might open the policy link in a different browser or session, the following code describes how the Wingtip Games site skips the cross-site request forgery (XSRF) logic for the policy link, which is skipped if the **skip\_correlation** property is set to **true** when the authentication challenge for the **b2c\_1a\_invitation** policy is created by the Wingtip Games site (see the “Create the authentication challenge” section).

namespace WingTipCommon.AspNetCore.Authentication.OpenIdConnect  
{  
 public class WingTipOpenIdConnectHandler : OpenIdConnectHandler  
 {  
 public WingTipOpenIdConnectHandler(HttpClient backchannel, HtmlEncoder htmlEncoder)  
 : base(backchannel, htmlEncoder)  
 {  
 }  
  
 protected override void GenerateCorrelationId(AuthenticationProperties authenticationProperties)  
 {  
 var skipCorrelation = SkipCorrelation(authenticationProperties);  
  
 if (skipCorrelation)  
 {  
 return;  
 }  
  
 base.GenerateCorrelationId(authenticationProperties);  
 }  
  
 protected override bool ValidateCorrelationId(AuthenticationProperties authenticationProperties)  
 {  
 var skipCorrelation = SkipCorrelation(authenticationProperties);  
  
 if (skipCorrelation)  
 {  
 return true;  
 }  
  
 return base.ValidateCorrelationId(authenticationProperties);  
 }  
  
 private static bool SkipCorrelation(AuthenticationProperties authenticationProperties)  
 {  
 if (authenticationProperties == null)  
 {  
 return false;  
 }  
  
 if (!authenticationProperties.Items.ContainsKey("skip\_correlation"))  
 {  
 return false;  
 }  
  
 bool skipCorrelation;  
  
 if (!bool.TryParse(authenticationProperties.Items["skip\_correlation"], out skipCorrelation))  
 {  
 return false;  
 }  
  
 return skipCorrelation;  
 }  
 }  
}

## How to: Configure the relying party policy

The following snippet shows how the **b2c\_1a\_invitation** policy is configured.

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
<TrustFrameworkPolicy  
 PolicySchemaVersion="0.3.0.0"  
 TenantId="b2ctechready.onmicrosoft.com"  
 PolicyId="B2C\_1A\_invitation"  
 PublicPolicyUri=<http://b2ctechready.onmicrosoft.com>  
 ...>  
 <BasePolicy>  
 <TenantId>b2ctechready.onmicrosoft.com</TenantId>  
 <PolicyId>B2C\_1A\_base\_extensions</PolicyId>  
 </BasePolicy>  
 <!-- The following policy is used for registering a user via an invitation link. -->  
 <RelyingParty>  
 <DefaultUserJourney ReferenceId="Invitation" />  
 <UserJourneyBehaviors>  
 <SingleSignOn Scope="Tenant" />  
 <SessionExpiryType>Rolling</SessionExpiryType>  
 <SessionExpiryInSeconds>86400</SessionExpiryInSeconds>  
 </UserJourneyBehaviors>  
 <TechnicalProfile Id="Invitation">  
 <DisplayName>Invitation</DisplayName>  
 <Protocol Name="OpenIdConnect" />  
 **<InputTokenFormat>JWT</InputTokenFormat>  
 <CryptographicKeys>  
 <Key Id="client\_secret" StorageReferenceId="WingTipGamesClientSecret" />  
 </CryptographicKeys>  
 <InputClaims>  
 <InputClaim ClaimTypeReferenceId="extension\_Brand" />  
 <InputClaim ClaimTypeReferenceId="extension\_PlayerProfileRegistrationMode" />  
 <InputClaim ClaimTypeReferenceId="extension\_VerifiedEmail" />  
 </InputClaims>** <OutputClaims>  
 <OutputClaim ClaimTypeReferenceId="displayName" />  
 <OutputClaim ClaimTypeReferenceId="email" />  
 <OutputClaim ClaimTypeReferenceId="identityProvider" />  
 <OutputClaim ClaimTypeReferenceId="newUser" />  
 <OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="sub" />  
 <OutputClaim ClaimTypeReferenceId="extension\_ListenerGenre" />  
 <OutputClaim ClaimTypeReferenceId="extension\_Picture" />  
 <OutputClaim ClaimTypeReferenceId="extension\_PlayerProfilePercentComplete" />  
 <OutputClaim ClaimTypeReferenceId="extension\_PlayerTag" />  
 <OutputClaim ClaimTypeReferenceId="extension\_PlayerZone" />  
 </OutputClaims>  
 <SubjectNamingInfo ClaimType="sub" />  
 </TechnicalProfile>  
 </RelyingParty>  
</TrustFrameworkPolicy>

The **InputTokenFormat** element is set to **JWT** so the relying party policy can receive the JWT that is sent in the previous section.

The **CryptographicKeys/Key** element references the client secret of the Wingtip Games site so the relying party policy can validate the signature of the JWT.

The **InputClaims** element references the input claims so the relying party policy can pass them to the **Invitation** user journey.

## How to: Configure the base policy

The base policy is configured as follows:

1. The **extension\_VerifiedEmail** claim type.
2. The **CreateEmailFromVerifiedEmail** claims transformation.
3. The **LocalAccount-Registration-VerifiedEmail** technical profile for the **Local Account** claims provider.
4. The **Invitation** user journey.

### Configure the extension\_VerifiedEmail claim type

The following snippet shows how the **extension\_VerifiedEmail** claim type is configured.

<ClaimType Id="extension\_VerifiedEmail">  
 <DisplayName>Verified Email</DisplayName>  
 <DataType>string</DataType>  
 <DefaultPartnerClaimTypes>  
 <Protocol Name="OAuth2" PartnerClaimType="verified\_email" />  
 <Protocol Name="OpenIdConnect" PartnerClaimType="verified\_email" />  
 <Protocol Name="SAML2" PartnerClaimType="http://schemas.wingtipb2c.net/identity/claims/verifiedemail" />  
 </DefaultPartnerClaimTypes>  
 <UserInputType>Readonly</UserInputType>  
 </ClaimType>

The **UserInputType** element is set to **Readonly** so the end user can only register with this email address.

### Configure the CreateEmailFromVerifiedEmail claims transformation

The following snippet shows how the **CreateEmailFromVerifiedEmail** claims transformation is configured.

<ClaimsTransformation Id="CreateEmailFromVerifiedEmail" TransformationMethod="FormatStringClaim">  
 <InputClaims>  
 <InputClaim ClaimTypeReferenceId="extension\_VerifiedEmail" TransformationClaimType="inputClaim" />  
 </InputClaims>  
 <InputParameters>  
 <InputParameter Id="stringFormat" DataType="string" Value="{0}" />  
 </InputParameters>  
 <OutputClaims>  
 <OutputClaim ClaimTypeReferenceId="email" TransformationClaimType="outputClaim" />  
 </OutputClaims>  
 </ClaimsTransformation>

This claims transformation copies the **extension\_VerifiedClaim** claim, which is passed in the signed JWT to the **b2c\_1a\_invitation** policy, to the **email** claim, which is stored by the **LocalAccount-Registration-VerifiedEmail** technical profile for the **Local Account** claims provider (see the next section) to the B2C directory.

### Configure the LocalAccount-Registration-VerifiedEmail technical profile for the Local Account claims provider

The following snippet shows how the **LocalAccount-Registration-VerifiedEmail** technical profile for the **Local Account** claims provider is configured.

<TechnicalProfile Id="LocalAccount-Registration-VerifiedEmail">  
 <DisplayName>WingTip Account</DisplayName>  
 <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.SelfAssertedAttributeProvider, Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />  
 <Metadata>  
 <Item Key="ContentDefinitionReferenceId">api.localaccount.registration</Item>  
 <Item Key="IpAddressClaimReferenceId">IpAddress</Item>  
 <Item Key="language.button\_continue">Create</Item>  
 </Metadata>  
 <CryptographicKeys>  
 <Key Id="issuer\_secret" StorageReferenceId="TokenSigningKeyContainer" />  
 </CryptographicKeys>  
 <InputClaimsTransformations>  
 **<InputClaimsTransformation ReferenceId="CreateEmailFromVerifiedEmail" />**  
 </InputClaimsTransformations>  
 <InputClaims>  
 **<InputClaim ClaimTypeReferenceId="extension\_VerifiedEmail" />**  
 </InputClaims>  
 <OutputClaims>  
 **<OutputClaim ClaimTypeReferenceId="extension\_VerifiedEmail" Required="true" />**  
 <OutputClaim ClaimTypeReferenceId="newPassword" Required="true" />  
 <OutputClaim ClaimTypeReferenceId="reenterPassword" Required="true" />  
 <OutputClaim ClaimTypeReferenceId="displayName" Required="true" />  
 <OutputClaim ClaimTypeReferenceId="extension\_TermsOfServiceConsented" Required="true" />  
 <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="localAccountAuthentication" />  
 <OutputClaim ClaimTypeReferenceId="executed-SelfAsserted-Input" DefaultValue="true" />  
 <OutputClaim ClaimTypeReferenceId="newUser" />  
 <OutputClaim ClaimTypeReferenceId="objectId" />  
 <OutputClaim ClaimTypeReferenceId="sub" />  
 <OutputClaim ClaimTypeReferenceId="userPrincipalName" />  
 </OutputClaims>  
 <ValidationTechnicalProfiles>  
 <ValidationTechnicalProfile ReferenceId="AAD-WriteUserByEmail-ThrowIfExists" />  
 </ValidationTechnicalProfiles>  
 <UseTechnicalProfileForSessionManagement ReferenceId="SSOSession-AAD" />  
 </TechnicalProfile>

This technical profile references:

* The **CreateEmailFromVerifiedEmail** claims transformation (see the previous section) as an input claims transformation so the email address of the end user is copied from the **extension\_VerifiedClaim** claim, which is passed in the signed JWT to the **b2c\_1a\_invitation** policy, to the **email** claim, which is written by the **AAD-WriteUserByEmail-ThrowIfExists** technical profile for the **AAD** claims provider to the B2C directory.
* The **extension\_VerifiedEmail** claim as an input claim so the email address of the end user is displayed as read-only to the end user

### Configure the Invitation user journey

The following snippet shows how the **Invitation** user journey is configured.

<UserJourney Id="Invitation">  
 <PreserveOriginalAssertion>false</PreserveOriginalAssertion>  
 <OrchestrationSteps>  
 <OrchestrationStep Order="1" Type="ClaimsProviderSelection" ContentDefinitionReferenceId="api.idpselection.signupsignin">  
 <ClaimsProviderSelections>  
 <ClaimsProviderSelection TargetClaimsExchangeId="AmazonAccountExchange" />  
 <ClaimsProviderSelection TargetClaimsExchangeId="FacebookAccountExchange" />  
 <ClaimsProviderSelection TargetClaimsExchangeId="LocalAccountRegistrationExchange" />  
 </ClaimsProviderSelections>  
 </OrchestrationStep>  
 <OrchestrationStep Order="2" Type="ClaimsExchange">  
 <ClaimsExchanges>  
 <ClaimsExchange Id="AmazonAccountExchange" TechnicalProfileReferenceId="AmazonAccount-OAuth2" />  
 <ClaimsExchange Id="FacebookAccountExchange" TechnicalProfileReferenceId="FacebookAccount-OAuth2" />  
 **<ClaimsExchange Id="LocalAccountRegistrationExchange" TechnicalProfileReferenceId="LocalAccount-Registration-VerifiedEmail" />**  
 </ClaimsExchanges>  
 </OrchestrationStep>  
  
 ...  
  
 <OrchestrationStep Order="15" Type="SendClaims" CpimIssuerTechnicalProfileReferenceId="JwtIssuer" />  
 </OrchestrationSteps>  
 </UserJourney>

This user journey references the **LocalAccount-Registration-VerifiedEmail** technical profile for the **Local Account** claims provider so the end user can register a local account with their email address.