Developing with Power BI Embedded

Power BI has become widely-adopted as a Software-as-a-Service (SaaS) application because it gives licensed users the ability to access and interact with Power BI reports and dashboards through the Power BI Service and through the native apps that Microsoft has published for popular mobile devices such as iPhone, Android and Windows 10. Power BI also provides professional developers with valuable Platform-as-a-Service (PaaS) capabilities which makes it possible to embed Power BI reports and dashboard into custom applications.

This whitepaper focuses on the details of software development which leverages the PaaS capabilities of Power BI to embed reports and dashboards in custom application. It explains the essential terminology, concepts and developer skills required to embed reports and dashboards. As you will learn, there are many difference options and a significant number of details which can make getting started quite difficult. However, once you understand the landscape of this powerful platform, you will be able to effectively develop with Power BI embedding in scenarios for both commercial applications and enterprise development which targets a single organization.

# Power BI Embedding Fundamentals

## Understanding Embeddable Resources

The Power BI platform makes it possible to embed three types of resource which includes reports, dashboards and dashboard tiles. However, the list of supported features for each type of embedded resource is quite different. For example, when you embed a report you will find there are several powerful features that are not available when embedding a dashboard or a dashboard tile.

When you embed a Power BI report in a custom application, it provides interactive behavior to the user. For example, the user can filter data in an embedded report using slicers and highlighting just as if the user were viewing the same report through the Power BI service in the browser. You can also write client-side JavaScript code for an embedded report to implement a custom page navigation scheme or a custom filtering experience.

Embedded dashboards are different than embedded reports because they do not provide the same level of interactive behavior. For example, clicking on a tile inside an embedded dashboard does not redirect the user to a report page as it does when accessing the same dashboard through the Power BI service. However, embedded dashboard do exhibit a modest amount of interactive behavior because hovering over a tile will generate popup tooltips.

There is good news if you are working with real-time dashboards which you have built on top of streaming datasets. You can embed a real-time dashboard in a custom application and it will continue to update itself automatically.

When you embed a dashboard, it is an all-or-nothing proposition when it comes to which tiles are displayed. You also have the option of embedding dashboard tiles individually. This can be handy when you wanted to select which tiles from a dashboard are displayed or you want more flexibility as to where each dashboard tile is displayed. One other thing to keep in mind that you cannot embed a real-time tile from a real-time dashboard. When you are working with real-time dashboards, you must embed the entire dashboard and not the individual dashboard tiles.

## Power BI Premium

## Power BI Embedded Version 1 versus Version 2

## First Party Embedding versus Third Party Embedding

When you begin to design an application that will leverage Power BI embedding features, your first step is to determine whether your scenario calls for first party embedding or third party embedding. Choosing between first party embedding and third party embedding at the start is essential because it impacts the way you design your application and write the code to authenticate and interact with the Power BI service.

First-party embedding is used in a scenario in which each user already has an Azure AD user account and a Power BI user license. For example, a company using Office 365 and Power BI might decide to use first party embedding to develop a custom Intranet-style application which surfaces Power BI reports and dashboards. When developing with first party embedding, your application must be written to authenticate users with Azure AD and to interact with the Power BI service using the identity of the currently authenticated user.

There is one important thing you should always keep in mind when developing with first party embedding. That is any user you can reach with a custom application will also be able to access to the Power BI service using the browser. This means these users can use the Power BI service directly to create, view and edit reports and dashboards without any need for custom application development. Obviously, you shouldn’t develop a custom application to recreate functionality that’s already provided out of the box by the Power BI service. Instead, you should only use first party embedding once you have determined the requirements of your application go beyond what’s available through the browser-based experience of the Power BI service.

Third-party embedding will be more attractive to ISVs because it can be used in scenarios in which users have neither Power BI licenses nor Azure AD user accounts. Therefore, an ISV can use 3rd party embedding to develop commercial SaaS-style applications using a custom identity provider or identity provider service other than Azure Active Directory. There is no need to individually license users with Power BI because the Power BI Premium licensing model allows ISVs to pay a single monthly usage fee to reach any number of users that are unnamed and unlicensed from the perspective of the Power BI service.

# Understanding Row-Level Security

# App Workspaces and Dedicated Capacities

## Power BI SKUs for Dedicated Capacities

# Authentication with Azure Active Directory

There is an important prerequisite skill you need to develop with Power BI embedding. You must learn how to work with Azure AD to register new applications and to configure the permissions your custom applications need to call the Power BI Service API. You must also learn how to write security-related code in your application to authenticate users and to acquire access tokens from Azure AD. You will see the details of how to write this type of code a little later in this blog post.

Let’s start with a quick discussion of how authentication works in Azure AD. When an organization first signs up for a Microsoft cloud-based service such as Office 365 or SharePoint Online, the signup process results in the creation of an Azure AD tenant. An *Azure AD tenant* is a named instance of the Azure AD service which gets created for a specific organization and has an initial domain name such as MyTenantName.onMicrosoft.com. After creating a new Azure AD tenant, an organization can then configure it to support additional domain names (e.g. MyCompany.com) so they do not have to rely on the initial domain name that ends with onMicrosoft.com.

An Azure AD tenant is created with a default directorywhich provides a scope for creating user accounts and groups. Note that an Azure AD tenant and an Azure AD directory are not the same thing because it’s possible to configure a tenant with multiple directories. However, you can say that every user account and group in Azure AD is created within the scope of a specific directory within a specific tenant. When users sign in using the credentials of their Azure AD user account, Azure AD authenticates them to establish their identity and determine their group membership.

There are several different authentication protocols support by Azure AD but the one that is relevant to Power BI embedding is *OAuth2* which is used together with an identity layer built on top known as *OpenID Connect*. To use the OAuth2 protocol, you must first register your application with Azure AD. You accomplish this by creating a new Azure AD application within the scope of a specific tenancy. When you create a new Azure AD application, you can configure it with credentials. This makes it possible for Azure AD to authenticate the application itself and establish an application identity that is distinct from any user identity.

If you have an Azure subscription, you can use the Azure AD portal to create new Azure AD applications by hand. If you do not have an Azure subscription, you can still create an Azure AD application using either a custom PowerShell script or the Azure AD application registration utility for Power BI which is accessible through the URL at <https://app.powerbi.com/apps>.

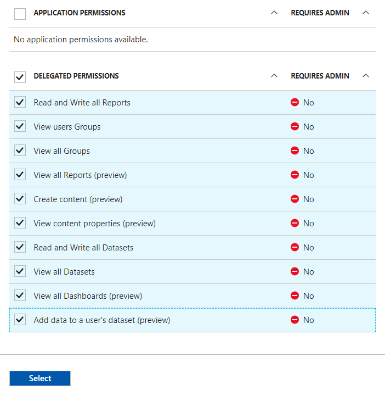
## Delegated Permissions versus Application Permissions

When you create an Azure AD application for a custom application, you must configure the required permissions that allow your application to call other cloud-based services such as the Azure Graph API, the Microsoft Graph API and (most importantly) the Power BI Service API. Each one of these APIs defines its own custom set of permissions. This means that the set of permissions you configure for the Microsoft Graph API will be completely different from the set of permissions you configure for the Power BI Service API.

An Azure application can be configured with two different types of permissions: delegated permissions and application permissions. *Delegated permissions* allow your application to inherit the permissions of the currently logged on user. Delegated permissions are restrictive because the application never has a greater level of permissions than the current user. Application permissions are more powerful than delegated permissions because they are not restricted by the permissions of any user.

Let’s look at an example of delegated permissions and application permissions using the Microsoft Graph API. The API provides a delegated permission named *Read user mailbox settings*. If your application has that permission, you can access the mailbox settings for the current user. The Microsoft Graph API also provides an application permission named *Read and write all user mailbox settings* which is far more powerful. This permission allows your application to access any mailbox in the current Azure AD tenant.

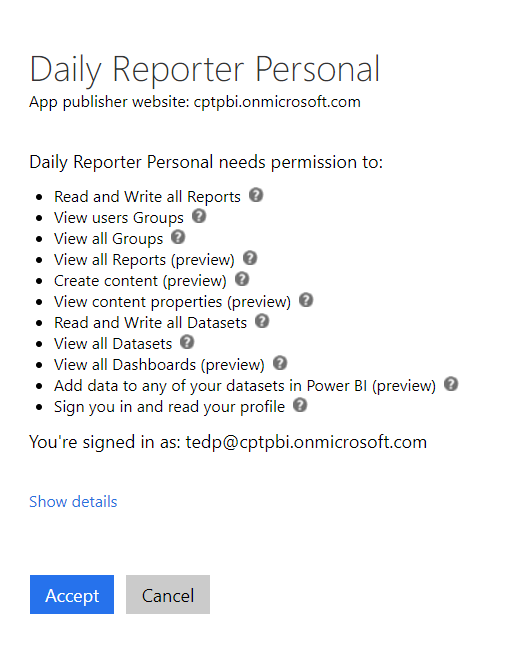
As of July 2017, the Power BI Service API supports delegated permissions but not application permissions. It’s quite possible we will see new application permissions in the future but today all permissions are delegated which means they are constrained by the permissions of the current user. The following screenshot shows what the Power BI Service API permissions look like when configuring them using the Azure portal.



## The Azure AD Common Consent Framework

When you begin working with custom application that use delegated permissions, it’s important to understand that Azure AD requires that each user consents to the set of permissions that the application is using. This is done by infrastructure in Azure AD that is known as the *common consent framework*.

Consider a simple example that illustrates how user consent works. Imagine you are developing a custom application using first party embedding application where users must authenticate using their Azure AD user accounts. As users go through the authentication process for the first time, Azure AD will prompt them with a consent page like the one shown in the following screenshot.



The key concept is that each user must consent before the application can use delegated permissions on behalf of that user. Once a user consents by clicking the Accept button, Azure AD remembers that the user has consents and does not need to interact with the user in future authentication requests.

In many cases the user interaction triggered common consent framework in the user interface experience works smoothly. However, there is a rough edge when working with Power BI 3rd party embedding because you must programmatically log into the master user account in a fashion that cannot deal with any type of user interaction. We will revisit this point a bit later when we register a custom application that uses 3rd party embedding.

## Azure AD’s Support for Web Apps versus Native Apps

When you create a new Azure AD application, you must configure the application type as either Web app or a Native app. When you are developing a web-based application, it usually makes sense to create the Azure AD application as a Web app to reach higher levels of security.

When you create an Azure AD application as a Web app, you can configure it with secret credentials to achieve stronger levels of authentication. In most cases, you will also configure an Azure AD application which as a Web app with one or more Reply URLs. Reply URLs add an extra security dimension because Azure AD can verify that the application is running within a pre-configured DNS domain on the Internet. This can really help to decrease the surface area that is exposed to attackers.

Azure AD also makes it possible to create an Azure AD application as a Native app instead of as a Web app. Native apps are used for specific scenarios such as a .NET application running on the laptop computer or an iOS app running on an iPhone. An important aspect of a Native app is that it is considered to be a *public client*. Unlike a web app which can keep track of server-side secrets, native apps cannot keep secrets such as client credentials. Therefore, Native apps can only authenticate with a user name and password. This means that a native app cannot establish application identity nor can it take advantage of application permissions.

So why am I going into all this detail about native apps? As it turns out, it’s important to 3rd party embedding where you must create the Azure AD application for your custom application as a native app. I will explain why this requirement exists later in this post. For now, I just want you to keep in mind that Native app is more restricted and less secure than a web app in several ways.

## Choosing the Right Authentication Flow

There are two important skills you need when developing a custom application that authenticates with Azure AD. The first thing is to register your application by creating an Azure AD application as we discussed in the previous section. The second thing you need to do is to design and write your application code to implement one of the OAuth2 authentication flows supported by Azure AD.

The primary reason that you implement an authentication flow in your application is to acquire an access token from Azure AD. Once your application has used an authentication flow to acquire an access token, it can then use that access token to make authenticated calls into a cloud-based service such as the Microsoft Graph API and the Power BI Service API.

Azure AD supports the following four authentication flows which are used in different scenarios.

* Client Credentials Grant Flow
* Authorization Code Grant Flow
* Implicit Grant Flow
* User Credentials Flow

The *Client Credentials Grant Flow* is used in a web app to authenticate the application itself and to establish an application identity which has no associated user identity. This authentication flow is used when an application needs to take advantage of application permissions. However, the *Client Credentials Grant Flow* isn’t relevant to Power BI embedding because the Power BI Service API does not currently support any application permissions.

The next two authentication flows are used in a Web app to authenticate the user and to establish user identity. *Authorization Code Grant Flow* is more secure because it requires application to provide a client secret during the authentication process just after requiring the user to provide a secret password. *Implicit Grant Flow* is used by client-side Web applications such as single page applications (SPAs) which run entirely within the browser and cannot keep any hidden secrets. The implicit grant flow authentication is a bit less secure because it does not include a client secret and the access token is passed directly back to the client code running in the browser.

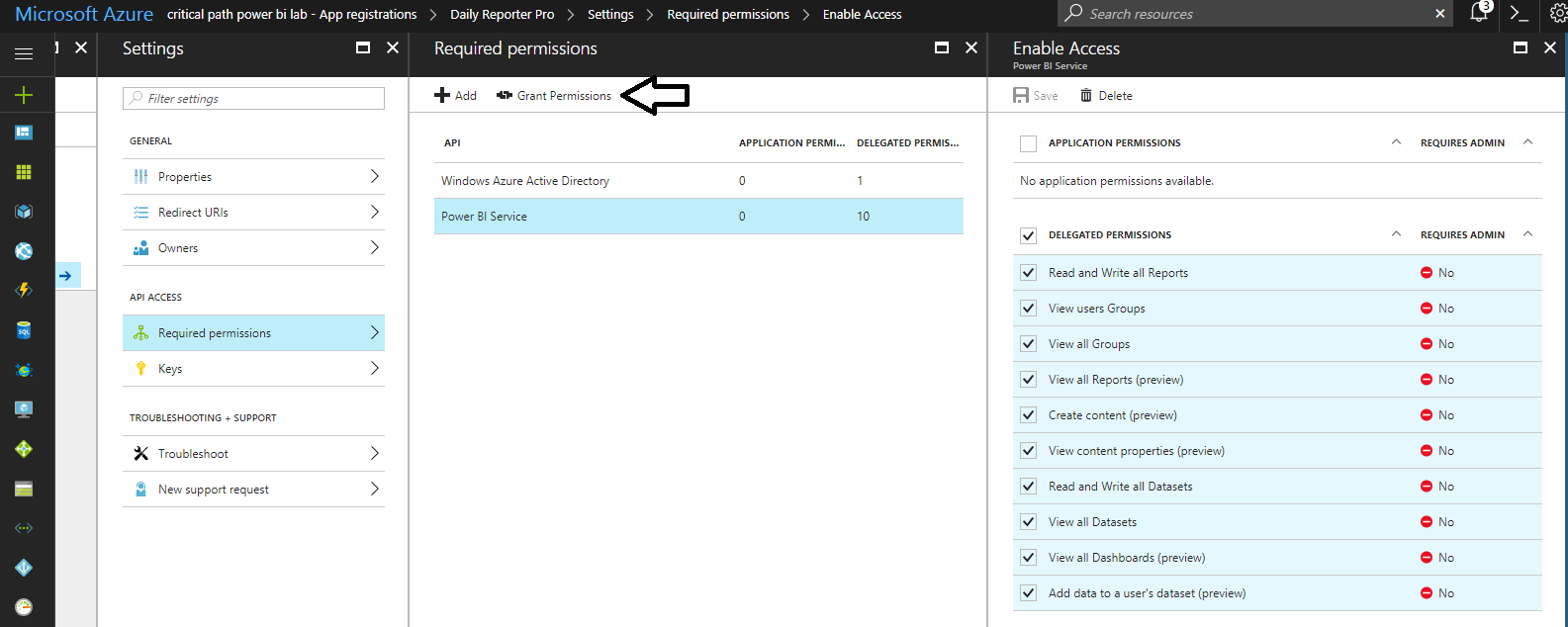
When you are developing a custom application that uses 1st party embedding, you can use either the *Authorization Code Grant Flow* the *Implicit Grant Flow*. For example, you would typically use the *Authorization Code Grant Flow* when developing an ASP.NET MVC which can hide server-side secrets. You would use the Implicit Grant Flow if you are creating an client-side SPA using a JavaScript framework such as Angular or AngularJS. Both authentication flows authenticate the current user and allow your application to acquire an access token that tracks that user’s identity and delegated permissions which can be used.

## Working with User Credentials Flow

The last authentication flow is User Credentials Flow which is particularly relevant to this discussion. When you develop using 3rd party embedding, the code your write does not access the Power BI Service API under the identity of the current user. Instead, you access the Power BI Service API under the identity of a single master user account regardless of who the current user is. Therefore, your code must directly authenticate the master user account against Azure AD using a user name and password.

One tricky aspect of 3rd party embedding is that you must authenticate the master user account in a fashion that does not require any user interaction. Therefore, you must ensure the master user account has consented to the delegated permissions before you attempt to authenticate using the User Credentials flow. Since you are writing code that must run under an unattended execution scenario, it will fail when trying to obtain an access token if consent has not been granted beforehand.

The easiest way to deal with this is to manually configure the consent for delegated permissions in the Azure portal. Once you have configured the required permissions in your Azure AD application, you can manually configure consent for all users by clicking on the *Grant Permissions* button to as shown in the following screenshot.



Now that you have seen how to consent to the required permissions b hand through the Azure portal, you might be thinking how can I write code to automate this process. Unfortunately, you will not find much information by running Internet searches with Google or Bing. From my research this week, I found that Microsoft does not yet support granting consent for permissions through any of their released PowerShell libraries. However, it is possible to write the required code against the Azure Active Directory Graph API client library as shown in a demo application I wrote named [AzureAppRegUtility](https://github.com/CriticalPathTraining/PowerBiEmbedded/blob/master/AzureAppRegUtility/AzureAppRegUtility/Program.cs).

There is a bit of a chicken-and-the-egg issue when using the AzureAppRegUtility. That is you can use this utility to register new Azure AD application for Power BI embedding until your register it with Azure AD and configure the permissions it requires. However, once you have created an Azure AD application for AzureAppRegUtility, you can then use the console application to register as many Azure AD application you need for your work with Power BI embedding.

## Application Permissions and Authentication Flows

## Power BI App Registration Page

## Acquiring and Managing Access Tokens

# Programming with the Power BI Service API

## Writing Code to Upload and Manage PBIX Files

## Access Tokens versus Embed Tokens

An Access Token is required to authenticate with the Power BI service

An Embed Token determines:

* Which resource can be accessed
* Which access level (view, create, edit)
* How long it can be accessed
* The data can be seen by the user

Embed tokens expire when the access token expires

* The app can be developed to silently refresh the embed token when it expires

## Retrieving the Data to Embed an Existing Report

## Retrieving the Data to Embed a New Report

## Retrieving the Data to Embed an Existing Dashboard

## Retrieving the Data to Embed an Existing Dashboard Tile

## Retrieving Embed Tokens for Third Party Embedding

# Programming with the Power BI JavaScript API

## Embedding an Existing Power BI Report

## Interacting with an Embedded Report

## Embedding a New Report

## Embedding an Existing Dashboard

## Embedding an Existing Dashboard Title

# Developing a 3rd Party Embedding Application

Check list of things that have to be done in your Azure AD tenant.

1. Create a master user account
2. Assign the master user account a Power BI Pro license
3. Create an application workspace
4. Configure master user account as admin of the workspace
5. Populate the workspace with datasets, reports and dashboards

### Creating an ASP.NET MVC Project for 3rd Party Power BI Embedding

something one this

Install-Package Microsoft.IdentityModel.Clients.ActiveDirectory

Install-Package Microsoft.PowerBI.Api

Install-Package Microsoft.PowerBI.JavaScript

Microsoft.PowerBi.Api.dll

Dependencies will be imported:

Newtonsoft.Json assembly

Microsoft.Rest.ClientRuntime assembly

### Programming with the Power BI Service API

something on this

### Understanding Power BI Embedded Object

iFrame works in any modern browser control

Relies on web standards: HTML5, JavaScript

Works in web applications, mobile applications, and even thick client applications

### Programming with the Power BI JavaScript API

### The Road Ahead

As of July 2017, the 3rd party embedding features in Power BI do not support a dynamic scheme for Row Level Security (RLS).

Multiple master app accounts can be considered to enforce security filters

* + This is a workaround until the DAX USERNAME function can return an effective username passed by the application—not yet implemented
  + In the meantime, the USERNAME function will return the authenticated Power BI accoun