

# Programming the Power BI Service API



# Agenda

- Power BI Service API Overview
- Creating App Workspaces and Workspace Associations
- Retrieving Data about Datasets, Reports and Dashboards
- Publishing PBIX Project Files
- Patching Datasource Credentials & Refreshing Datasets
- Cloning Power BI Content across Workspaces



# Access Token Acquisition (Native Client)

- With interactive login

```
static string aadAuthorizationEndpoint = "https://login.windows.net/common/oauth2/authorize";
static string resourceUriPowerBi = "https://analysis.windows.net/powerbi/api";
static string urlPowerBiRestApiRoot = "https://api.powerbi.com/";

public const string clientId = "315e87eb-a6a0-4886-9b20-9f7ecdaca888";
public const string redirectUrl = "https://localhost/app1234";

static string GetAccessToken() {
    // create new authentication context
    var authenticationContext = new AuthenticationContext(aadAuthorizationEndpoint);

    // use authentication context to trigger user sign-in and return access token
    var userAuthnResult = authenticationContext.AcquireTokenAsync(resourceUriPowerBi,
                                                                clientId,
                                                                new Uri(redirectUrl),
                                                                new PlatformParameters(PromptBehavior.Auto)).Result;

    // return access token to caller
    return userAuthnResult.AccessToken;
}
```

```
string userName = "tedp@sharepointconfessions.onmicrosoft.com";
string userPassword = "Dublin@1234";

UserPasswordCredential creds = new UserPasswordCredential(userName, userPassword);
var userAuthnResult = authenticationContext.AcquireTokenAsync(PowerBiServiceResourceUri,
                                                                clientId,
                                                                creds).Result;

// cache access token in AccessToken field
AccessToken = userAuthnResult.AccessToken;
```



# Access Token Acquisition (web app)

```
private static string aadInstance = "https://login.microsoftonline.com/";
private static string resourceUrlPowerBi = "https://analysis.windows.net/powerbi/api";
private static string urlPowerBiRestApiRoot = "https://api.powerbi.com/";

private static string clientId = ConfigurationManager.AppSettings["client-id"];
private static string clientSecret = ConfigurationManager.AppSettings["client-secret"];
private static string redirectUrl = ConfigurationManager.AppSettings["reply-url"];

private static async Task<string> GetAccessTokenAsync() {
    // determine authorization URL for current tenant
    string tenantID = ClaimsPrincipal.Current.FindFirst("http://schemas.microsoft.com/identity/claims/tenantid").Value;
    string tenantAuthority = aadInstance + tenantID;

    // create ADAL cache object
    ApplicationDbContext db = new ApplicationDbContext();
    string signedInUserID = ClaimsPrincipal.Current.FindFirst(ClaimTypes.NameIdentifier).Value;
    ADALTokenCache userTokenCache = new ADALTokenCache(signedInUserID);

    // create authentication context
    AuthenticationContext authenticationContext = new AuthenticationContext(tenantAuthority, userTokenCache);

    // create client credential object using client ID and client Secret";
    ClientCredential clientCredential = new ClientCredential(clientId, clientSecret);

    // create user identifier object for logged on user
    string objectIdentifierId = "http://schemas.microsoft.com/identity/claims/objectidentifier";
    string userObjectID = ClaimsPrincipal.Current.FindFirst(objectIdentifierId).Value;
    UserIdentifier userIdentifier = new UserIdentifier(userObjectID, UserIdentifierType.UniqueId);

    // get access token for Power BI Service API from AAD
    AuthenticationResult authenticationResult =
        await authenticationContext.AcquireTokenSilentAsync(
            resourceUrlPowerBi,
            clientCredential,
            userIdentifier);

    // return access token back to user
    return authenticationResult.AccessToken;
}
```



# Initializing an Instance of PowerBIClient

- PowerBIClient object serves as top-level object
  - Used to execute calls against Power BI Service
  - Initialized with function to retrieve AAD access token

```
static string GetAccessToken() ...

static PowerBIClient GetPowerBiClient() {
    var tokenCredentials = new TokenCredentials(GetAccessToken(), "Bearer");
    return new PowerBIClient(new Uri(urlPowerBiRestApiRoot), tokenCredentials);
}

static void Main() {
    PowerBIClient pbiClient = GetPowerBiClient();
    var reports = pbiClient.Reports.GetReports().Value;
    foreach (var report in reports) {
        Console.WriteLine(report.Name);
    }
}
```





# The Power BI Service API

- Microsoft.PowerBI.Api.V2
  - AvailableFeatures
  - AvailableFeaturesExtensions
  - Capacities
  - CapacitiesExtensions
  - Dashboards
  - DashboardsExtensions
  - Datasets
  - DatasetsExtensions
  - Gateways
  - GatewaysExtensions
  - Groups
  - GroupsExtensions
    - IAvailableFeatures
    - ICapacities
    - IDashboards
    - IDatasets
    - IGateways
    - IGroups
    - IImports
  - Imports
  - Imports.BlockList
  - ImportsExtensions
    - IPowerBIClient
    - IReports
    - ITiles
  - PowerBIClient
  - Reports
  - ReportsExtensions
  - Tiles
  - TilesExtensions

- Microsoft.PowerBI.Api.V2.Models
  - AddDashboardRequest
  - AdditionalFeatureInfo
  - AssignToCapacityRequest
  - AvailableFeature
  - BasicCredentials
  - BindToGatewayRequest
  - Capacity
  - CapacityUserAccessRightEnum
  - CloneReportRequest
  - CloneTileRequest
  - Column
  - ConnectionDetails
  - ConnectionTypeEnum
  - CredentialDetails
  - CredentialTypeEnum
  - CrossFilteringBehaviorEnum
  - Dashboard
  - Dataset
  - DatasetMode
  - DatasetParameter
  - Datasource
  - DatasourceConnectionDetails
  - EffectivenessIdentity
  - EmbedToken
  - EncryptedConnectionEnum
  - EncryptionAlgorithmEnum
  - FeatureExtendedState
  - FeatureState
  - Gateway
  - GatewayDatasource

- GatewayPublicKey
- GenerateTokenRequest
- Group
- GroupCreationRequest
- GroupRestoreRequest
- GroupUserAccessRight
- GroupUserAccessRightEnum
- Import
- ImportConflictHandlerMode
- ImportInfo
- Measure
- NotifyOption
- ODataResponseListAvailableFeature
- ODataResponseListCapacity
- ODataResponseListDashboard
- ODataResponseListDataset
- ODataResponseListDatasetParameter
- ODataResponseListDatasource
- ODataResponseListGateway
- ODataResponseListGatewayDatasource
- ODataResponseListGroup
- ODataResponseListGroupUserAccessRight
- ODataResponseListImport
- ODataResponseListRefresh
- ODataResponseListReport
- ODataResponseListTable
- ODataResponseListTile
- ODataResponseListUserAccessRight
- PositionConflictActionEnum
- PrivacyLevelEnum
- PublishDatasourceToGatewayRequest

- RebindReportRequest
- Refresh
- RefreshRequest
- RefreshTypeEnum
- Relationship
- Report
- Row
- SourceReport
- StateEnum
- Table
- TemporaryUploadLocation
- Tile
- TokenAccessLevel
- UpdateDatasetParameterDetails
- UpdateDatasetParametersRequest
- UpdateDatasourceConnectionRequest
- UpdateDatasourceRequest
- UpdateDatasourcesRequest
- UpdateReportContentRequest
- UserAccessRight
- UserAccessRightEnum



# Implementing the GetDatasetsAsync Method

```
public static async Task<DatasetViewModel> GetDatasetAsync(string WorkspaceId, string DatasetId) {  
    PowerBIClient pbiClient = GetPowerBiClient();  
    Dataset dataset = (await pbiClient.Datasets.GetDatasetByIdInGroupAsync(WorkspaceId, DatasetId));  
    IList<Datasource> datasources = (await pbiClient.Datasets.GetDatasourcesInGroupAsync(WorkspaceId, DatasetId)).Value;  
    IList<Refresh> refreshHistory = null;  
  
    if (dataset.IsRefreshable == true) {  
        refreshHistory = (await pbiClient.Datasets.GetRefreshHistoryInGroupAsync(WorkspaceId, DatasetId)).Value;  
    }  
  
    DatasetViewModel viewModel = new DatasetViewModel {  
        WorkspaceId=WorkspaceId,  
        Id = dataset.Id,  
        Name = dataset.Name,  
        Dataset = dataset,  
        Datasources = datasources,  
        RefreshHistroy = refreshHistory  
    };  
  
    return viewModel;  
}
```

```
public static async Task RefreshDatasetAsync(string WorkspaceId, string DatasetId) {  
    PowerBIClient pbiClient = GetPowerBiClient();  
    await pbiClient.Datasets.RefreshDatasetInGroupAsync(WorkspaceId, DatasetId);  
    return;  
}
```



# Creating Workspaces

```
public static async Task<Group> CreateWorkspacesAsync(string WorkspaceName) {  
    PowerBIClient pbiClient = GetPowerBiClient();  
    GroupCreationRequest createRequest = new GroupCreationRequest(WorkspaceName);  
    var workspace = await pbiClient.Groups.CreateGroupAsync(createRequest);  
  
    var secondaryAdmin = "pbiemasteruser@sharepointconfessions.onmicrosoft.com";  
    var userRights = new GroupUserAccessRight("Admin", secondaryAdmin);  
    await pbiClient.Groups.AddGroupUserAsync(workspace.Id, userRights);  
  
    return workspace;  
}
```





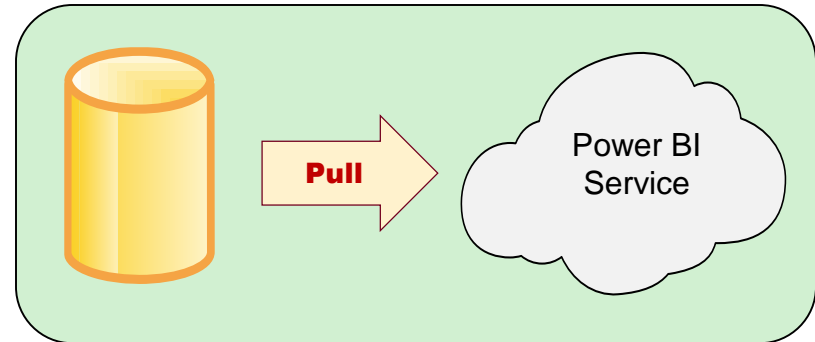
# Importing a PBIX File

```
public static async Task UploadPBIX(string WorkspaceId, string pbixName, string importName, bool updateSqlCredentials = false) {  
    string PbixFilePath = HttpContext.Current.Server.MapPath("/PBIX/" + pbixName);  
    PowerBIClient pbiClient = GetPowerBiClient();  
    FileStream stream = new FileStream(PbixFilePath, FileMode.Open, FileAccess.Read);  
    var import = await pbiClient.Imports.PostImportWithFileAsyncInGroup(WorkspaceId, stream, importName);  
  
    if (updateSqlCredentials) {  
        await PatchSqlDatasourceCredentials(WorkspaceId, importName);  
    }  
  
    return;  
}
```

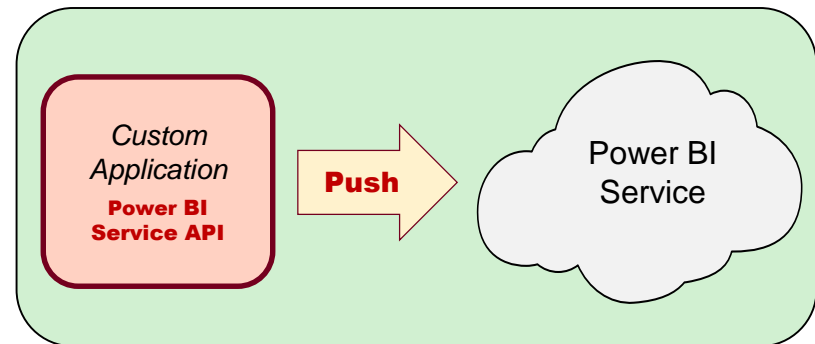


# Pull Datasets versus Real-time Datasets

- Pull Datasets
  - Imported Datasets
  - DirectQuery Datasets
  - Live Connect Datasets

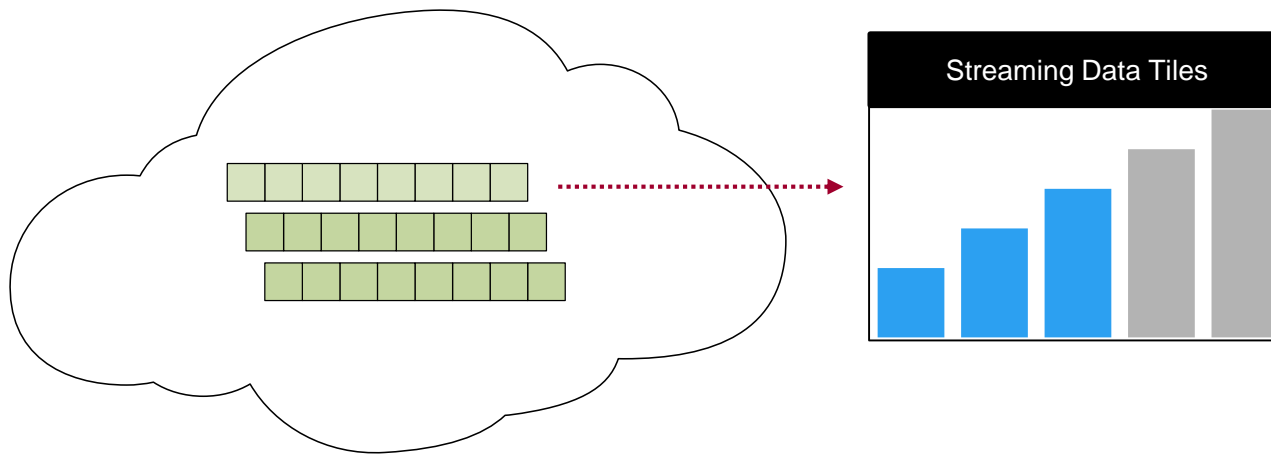


- Real-time Datasets
  - Streaming Datasets
  - Push Datasets
  - Hybrid Datasets



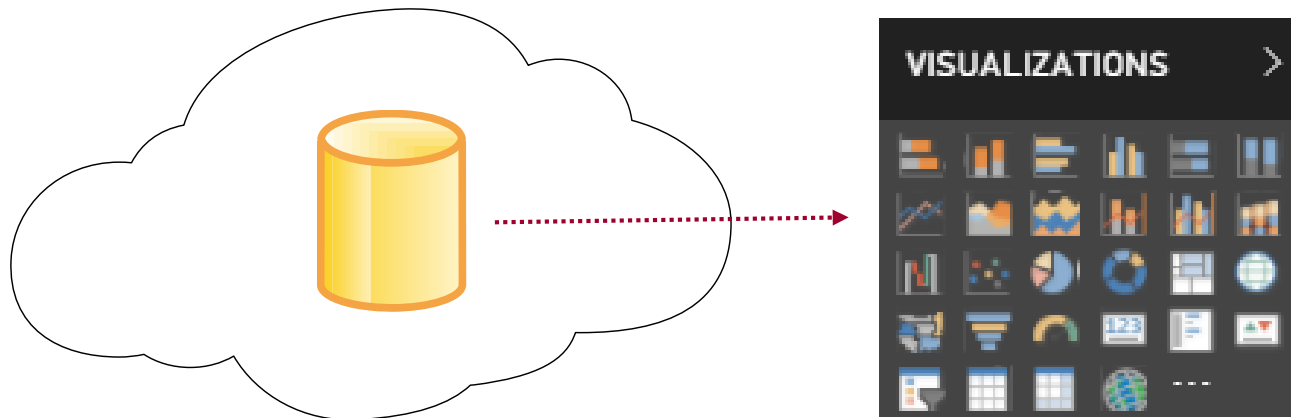
# Streaming Datasets

- Data stored in cloud-based cache – not persisted in DB
- Restricted to single table - no rich data modeling
- Not supported by standard Power BI report designer
- Dashboard created using specialized streaming data tiles
- No support for DAX, aggregation or filtering



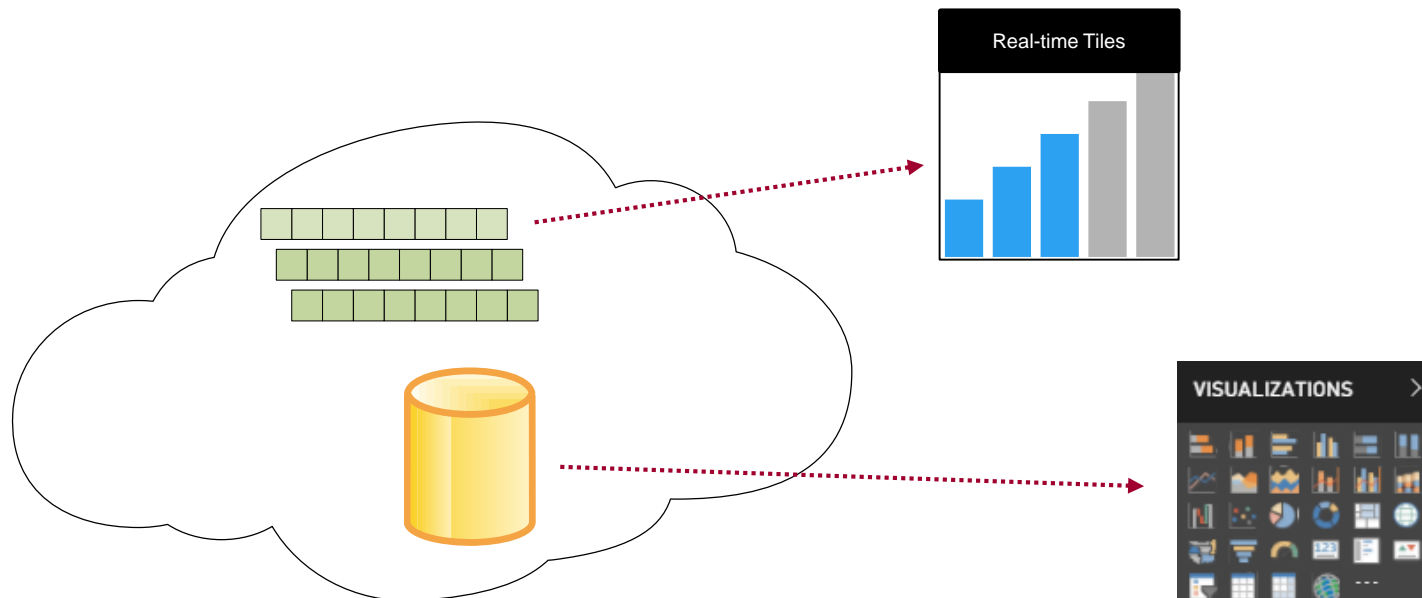
# Push Datasets

- Data stored in Azure SQL DB – not in cache
- Supports multiple tables and table relationships
- Supported by standard Power BI report designer
- Supports DAX, measures, aggregation & filtering



# Hybrid Datasets

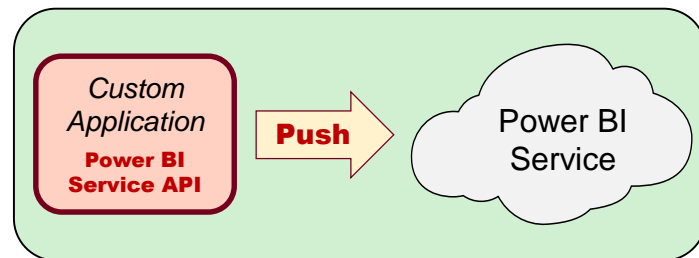
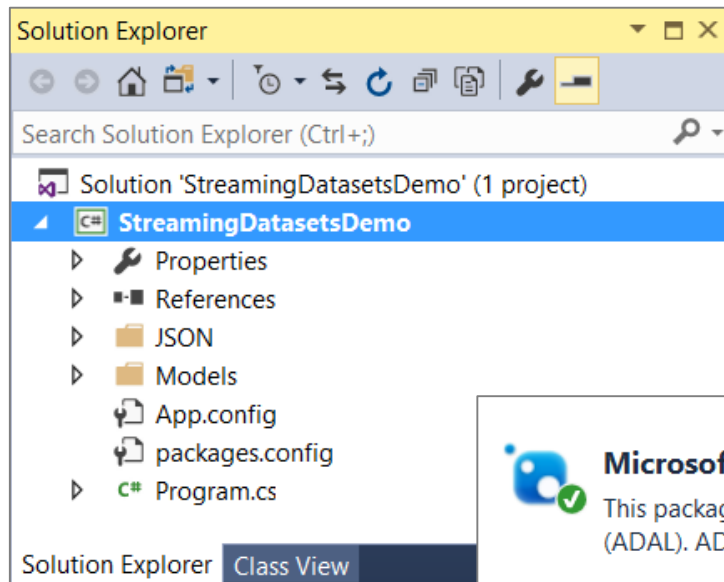
- Data stored in cloud-based cache AND in Azure SQL DB
- Restricted to single table and no rich data modeling
- Supported by streaming data tiles
- Supported by Power BI report designer





# The StreamingDatasetsDemo Project

- Console application project in Visual Studio 2017
  - Installed package for Azure AD Authentication library
  - Installed package to serialize .NET objects to JSON



**Microsoft.IdentityModel.Clients.ActiveDirectory** by Microsoft Corporation v3.17.0

This package contains the binaries of the Active Directory Authentication Library (ADAL). ADAL provides easy to use authentication functionality for your .NET base...



**Newtonsoft.Json** by James Newton-King

v10.0.3

Json.NET is a popular high-performance JSON framework for .NET

# Creating a Streaming Dataset

- Streaming dataset created using JSON schema definition
  - Streaming dataset limited to a single table
  - Columns defined using name and datatype
  - No support for any other column properties (e.g. formatting)

```
DemoStreamingDataset.json  X
{
  "name": "TemperatureReadings",
  "defaultMode": "Streaming",
  "tables": [
    { "name": "TemperatureReadings",
      "columns": [
        { "name": "Run", "dataType": "string" },
        { "name": "Time", "dataType": "Datetime" },
        { "name": "TimeWindow", "dataType": "string" },
        { "name": "TargetTemperature", "dataType": "Double" },
        { "name": "MinTemperature", "dataType": "Double" },
        { "name": "MaxTemperature", "dataType": "Double" },
        { "name": "BatchA", "dataType": "Double" },
        { "name": "BatchB", "dataType": "Double" },
        { "name": "BatchC", "dataType": "Double" }
      ]
    }
  ]
}
```

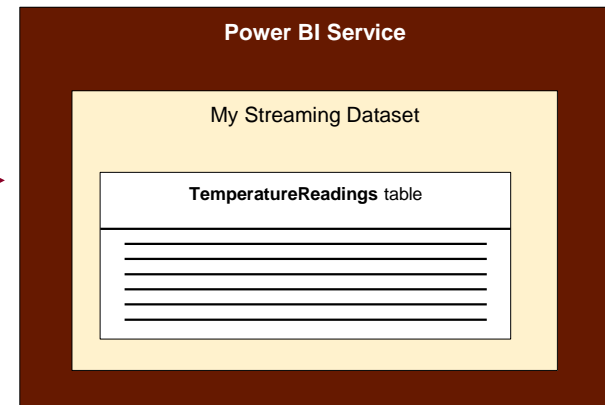
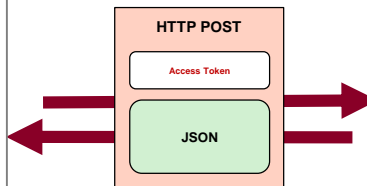


# Creating a Custom Dataset

- Dataset created by executing HTTP POST operation
  - One-time operation done as application begins running

```
// prepare call to create new dataset
1 string restUrlDatasets = ProgramGlobalConstants.PowerBiServiceRootUrl + "datasets";
string jsonNewDataset = Properties.Resources.NewDataset_json;
// execute REST call to create new dataset
2 string json = ExecutePostRequest(restUrlDatasets, jsonNewDataset);
// retrieve Guid to track dataset ID
3 Dataset dataset = JsonConvert.DeserializeObject<Dataset>(json);
CustomDatasetId = dataset.id;
```

```
DemoStreamingDataset.json
{
  "name": "TemperatureReadings",
  "defaultMode": "Streaming",
  "tables": [
    {
      "name": "TemperatureReadings",
      "columns": [
        { "name": "Run", "dataType": "string" },
        { "name": "Time", "dataType": "Datetime" },
        { "name": "TimeWindow", "dataType": "string" },
        { "name": "TargetTemperature", "dataType": "Double" },
        { "name": "MinTemperature", "dataType": "Double" },
        { "name": "MaxTemperature", "dataType": "Double" },
        { "name": "BatchA", "dataType": "Double" },
        { "name": "BatchB", "dataType": "Double" },
        { "name": "BatchC", "dataType": "Double" }
      ]
    }
  ]
}
```



# Adding Rows by Converting C# to JSON

```
TemperatureReadingsRow row = new TemperatureReadingsRow {
    Run = RunName,
    Time = DateTime.Now,
    TimeWindow = currentTimeWindow,
    TargetTemperature = 212,
    MinTemperature = 100,
    MaxTemperature = 250,
    BatchA = temperatureBatchA,
    BatchB = temperatureBatchB,
    BatchC = temperatureBatchC,
};

TemperatureReadingsRow[] rows = { row };
TemperatureReadingsRows temperatureReadingsRows = new TemperatureReadingsRows { rows = rows };
string jsonNewRows = JsonConvert.SerializeObject(temperatureReadingsRows);
string restUrlTargetTableRows = string.Format("{0}/{1}/tables/TemperatureReadings/rows", restUrlDatasets, DatasetId);
string jsonResultAddExpenseRows = ExecutePostRequest(restUrlTargetTableRows, jsonNewRows);
```

```
public class TemperatureReadingsRow {
    public string Run { get; set; }
    public DateTime Time { get; set; }
    public string TimeWindow { get; set; }
    public double TargetTemperature { get; set; }
    public double MinTemperature { get; set; }
    public double MaxTemperature { get; set; }
    public double BatchA { get; set; }
    public double BatchB { get; set; }
    public double BatchC { get; set; }
}

class TemperatureReadingsRows {
    public TemperatureReadingsRow[] rows { get; set; }
}
```



```
{
  "rows": [
    {
      "Run": "Run 06",
      "Time": "2017-10-05T22:43:40.364569-04:00",
      "TimeWindow": "22:43:30",
      "TargetTemperature": 212.0,
      "MinTemperature": 100.0,
      "MaxTemperature": 250.0,
      "BatchA": 152.73999999999995,
      "BatchB": 152.78,
      "BatchC": 152.25
    }
  ]
}
```



# Real-time Dataset Matrix

| Feature                  | Streaming                     | Hybrid | Push                             |
|--------------------------|-------------------------------|--------|----------------------------------|
| Updates in real-time     | Yes                           | Yes    | Yes                              |
| Smooth animations        | Yes                           | Yes    | No                               |
| Backed by Azure SQL DB   | No                            | Yes    | Yes                              |
| Report Designer Support  | No                            | Yes    | Yes                              |
| Allow Rich Data Modeling | No                            | No     | Yes                              |
| Ingestion Rate           | 5 request/sec<br>15KB/request |        | 1 request/second<br>16MB/request |

