# Program Capacities and Workspaces

Fabric REST API provides support for platform types and workspace item types. Platform item type instances created within scope of the current Entra Id tenant. Workspace item type instances created within scope of a Fabric workspace.

A screenshot of a computer

Description automatically generated

## Inspect Capacities

var capacities = fabricApiClient.Core.Capacities.ListCapacities().ToList();

foreach (var capacity in capacities) {

Console.WriteLine($"[{capacity.Sku}] {capacity.DisplayName} (ID={capacity.Id})");

}

## Create Workspaces

Here is code

public static Workspace CreateWorkspace(string WorkspaceName, string Description = null) {

var createRequest = new CreateWorkspaceRequest(WorkspaceName);

createRequest.Description = Description;

// associate workspace with Premium capacity

createRequest.CapacityId = new Guid(AppSettings.PremiumCapacityId);

// call Crete Workspace API

var workspace = fabricApiClient.Core.Workspaces.CreateWorkspace(createRequest).Value;

// determine Id of new workspace

Guid workspaceId = workspace.Id;

// return workspace object to caller

return workspace;

}

## Assign Workspace Roles to User, Groups and Service Principals

Adding users

public static void AddUserAsWorkspaceMemeber(Guid WorkspaceId, Guid UserId, WorkspaceRole RoleAssignment) {

var user = new Principal(UserId, PrincipalType.User);

var roleAssignment = new AddWorkspaceRoleAssignmentRequest(user, RoleAssignment);

fabricApiClient.Core.Workspaces.AddWorkspaceRoleAssignment(WorkspaceId, roleAssignment);

}

Ssss

public static void AddGroupAsWorkspaceMemeber(Guid WorkspaceId, Guid GroupId, WorkspaceRole RoleAssignment) {

var group = new Principal(GroupId, PrincipalType.Group);

var roleAssignment = new AddWorkspaceRoleAssignmentRequest(group, RoleAssignment);

fabricApiClient.Core.Workspaces.AddWorkspaceRoleAssignment(WorkspaceId, roleAssignment);

}

Ssss

public static void AddServicePrincipalAsWorkspaceMemeber(Guid WorkspaceId, Guid ServicePrincipalObjectId, WorkspaceRole RoleAssignment) {

var user = new Principal(ServicePrincipalObjectId, PrincipalType.ServicePrincipal);

var roleAssignment = new AddWorkspaceRoleAssignmentRequest(user, RoleAssignment);

fabricApiClient.Core.Workspaces.AddWorkspaceRoleAssignment(WorkspaceId, roleAssignment);

}

Xxx

var workspace = FabricRestApi.CreateWorkspace(WorkspaceName, "test workspace");

Guid TestUser1Id = new Guid(AppSettings.TestUser1Id);

Guid TestUser2Id = new Guid(AppSettings.TestUser2Id);

Guid TestADGroup1 = new Guid(AppSettings.TestADGroup1);

Guid TestServicePrincipal = new Guid(AppSettings.ServicePrincipalObjectId);

FabricRestApi.AddUserAsWorkspaceMemeber(workspace.Id, TestUser1Id, WorkspaceRole.Admin);

FabricRestApi.AddUserAsWorkspaceMemeber(workspace.Id, TestUser2Id, WorkspaceRole.Viewer);

FabricRestApi.AddGroupAsWorkspaceMemeber(workspace.Id, TestADGroup1, WorkspaceRole.Member);

FabricRestApi.AddServicePrincipalAsWorkspaceMemeber(workspace.Id, TestServicePrincipal, WorkspaceRole.Admin);

## Provision Workspace Identity

xxxxxx

# Program Fabric Connections

Creating connections programmatically is required in scenarios which involve OneLake Shortcuts, Data Pipelines and Semantic Models.

Before we dive into a discussion of connections, let’s take a moment to distinguish between **Inbound Security** versus **Outbound Security**. Inbound security is involved when a custom application executes API calls on Fabric REST API endpoints. A key point is that the custom application runs outside the Fabric environment. Before the application can call to the Fabric REST API, it must first authenticate with the Entra Id Service in order to acquire access tokens. It must then transmit an access token in each and every API call to Fabric REST API endpoints. This is a topic that has already been covered earlier in this guidance document.

Outbound security is different because it involves a scenario where you’ve created some type of workspace item inside the Fabric environment which must connect to an external datasource. For example, you can create a semantic model which connects to an Azure SQL database. In another example, you can create a OneLake shortcut which connects to an ADLS storage container. With outbound security, you can create and bind connections using the Fabric REST API as shown in the following diagram.

A close-up of a screen

Description automatically generated

Here is an important factor to keep in mind. When you create a connection with code, you are not really establishing a connection across the network at that point in time. Instead, you are really just creating a persistent Fabric object with metadata for creating a connection at some point in the future. This metadata includes the datasource type and path as well as security credentials. It is not until the connection is actually used by something in Fabric such as a semantic model or a OneLake shortcut when the Fabric Service reads this metadata and uses it to establish a connection to the datasource across the network.

Connections in Fabric are scoped at level of Entra Id tenant. That means that Fabric connection can be shared across workspaces. Of course, just because you can doesn’t mean that you should.

You can see all the connections you have access to using the Manage Connections and Gateways page in the Fabric Service.

A screenshot of a computer

Description automatically generated

Fabric connections support four different connectivity types.

1. Personal Cloud Connections (PCCs)
2. Sharable Cloud Connections (SCCs)
3. On-prem Gateway Connections
4. Virtual Gateway Connections

Personal Cloud Connections (PCCs) have been used in Power BI for years, However, they are limited because they cannot be shared. Each PCC is owned and exlussively used by a single user or service principal. There is no way to share a PCC.

Sharable Cloud Connections (SCCs) are new and serve as the strategic replacement for PCCs. Once a SCC has been created, it can be shared with other users or service principals. The creator of a Connection is automatically configured with connection Role Assignment of Owner. Other users and service principals can be added to the SCC membership with a Role Assignments of User, UserWithReshare or Owner.

## Create an Anonymous Web Connection

Content to come

## Create a Azure SQL Connectiong using Basic Credentials

Content to come

## Create an Azure Storage Connection using Service Principal Credentials

Content to come

# Create and Update Workspace Items

Fabric solutions built using workspace items. Developers must learn to discover, create and manage workspace items inside scope of a workspace.

A screenshot of a computer screen

Description automatically generated

Discover what items exist in specific workspace by calling List Items API

A screenshot of a computer

Description automatically generated

## Item Definitions in Fabric

Fabric items can be created and updated using item definitions. You can pass item definition when calling Create Item API. You can retrieve item definition for existing Fabric item by calling Get Item Definition. You can modify existing workspace item by calling Update Item Definition passing item definition.

A diagram of a process

Description automatically generated

When calling Create Item, you pass displayName, type and optionally an item definition.

Some type of Fabric Items are created using an item definition while others are not.

So, what exactly is an item definition? An item definition includes array of parts where each part is item-specific file. For example, you can create an item definition for a semantic model using two files which are **definition.pbism** and **model.bim**. Likewise, you can create an item definition for a report using three files which are **definition.pbir**, **report.json** and a report theme file.

A screenshot of a computer

Description automatically generated

File content for parts converted to/from inline base64 format when transmitted across network. Item definition for each item type requires unique set of parts.

## Create a Semantic Model using the CreateItem API

Constructing Item Definitions for the CreateItem API. FabricItemDefinitionFactory class used to build item definitions for calls to CreateItem. Examples provided for semantic models, reports, notebooks and data pipelines. Template files for item definitions maintained in ItemDefinitions folder as embedded resources. The utility method CreateInlineBase64Part converts standard string into Base64 string for inline part.

The method GetImportedSalesModelCreateRequest constructs item definition to create semantic model.

public static CreateItemRequest GetImportedSalesModelCreateRequest(string DisplayName) {

string part1FileContent = FabricIsvPlaybook.Properties.Resources.definition\_pbism;

string part2FileContent = FabricIsvPlaybook.Properties.Resources.sales\_model\_import\_bim;

var createRequest = new CreateItemRequest(DisplayName, ItemType.SemanticModel);

createRequest.Definition =

new ItemDefinition(new List<ItemDefinitionPart>() {

CreateInlineBase64Part("definition.pbism", part1FileContent),

CreateInlineBase64Part("model.bim", part2FileContent)

});

return createRequest;

}

Let’s focus on this code.

createRequest.Definition =

new ItemDefinition(new List<ItemDefinitionPart>() {

CreateInlineBase64Part("definition.pbism", part1FileContent),

CreateInlineBase64Part("model.bim", part2FileContent)

});

N

private static ItemDefinitionPart CreateInlineBase64Part(string Path, string Payload) {

string base64Payload = Convert.ToBase64String(Encoding.UTF8.GetBytes(Payload));

return new ItemDefinitionPart(Path, base64Payload, PayloadType.InlineBase64);

}

This code calls new on **new** on the **ItemDefinition** class of the .NET SDK and initializes with a list of item definition parts. In this case the parts are for a semantic model and they include definition.pbism and model.bim.

var workspace = FabricRestApi.CreateWorkspace(WorkspaceName);

var modelCreateRequest = FabricItemDefinitionFactory.GetImportedSalesModelCreateRequest(ImportedModelName);

var model = FabricRestApi.CreateItem(workspace.Id, modelCreateRequest);

Here is code

public static Item CreateItem(Guid WorkspaceId, CreateItemRequest CreateRequest) {

// call CreateItem API to create new item

var newItem = fabricApiClient.Core.Items.CreateItemAsync(WorkspaceId, CreateRequest).Result.Value;

// return object for new new workspace item to caller

return newItem;

}

This is e URL with the post.

https://api.fabric.microsoft.com/v1/workspaces/{WORKSPACE\_ID}/items

This is what gets sent across the network.

The call

A screenshot of a computer

Description automatically generated

When you call the CreateItem API to create a new semantic model, the call is processed as a long running operation (LRO). The POST request

* Location: https://api.fabric.microsoft.com/v1/operations/{OPERATION\_ID}
* Retry-After : 20

The developer needs to write code that waits 20 seconds and then sends a GET request to the URL in the Location to determine the operation status. The returned result has a status property that must be set to true before the result is available.

A screenshot of a computer

Description automatically generated

Then you can get the result of the LRO which is the metadata for the semantic model that has just been created.

A close-up of a product

Description automatically generated

What nice is that Result.Value, the SDK hides the complexity of having to deal with this call as a LRO.

// call CreateItem API to create new item

var newItem = fabricApiClient.Core.Items.CreateItemAsync(WorkspaceId, CreateRequest).Result.Value;

## Prepare the Semantic Model After Creation

Import-mode semantic model needs to be refreshed after it has been created from item definition. The key takeaway is that call to the CreateItem API creates a semantic model instance but it does not automatically populate semantic model with data.

var workspace = FabricRestApi.CreateWorkspace(WorkspaceName);

// create seantic model

var modelCreateRequest = FabricItemDefinitionFactory.GetImportedSalesModelCreateRequest(ImportedModelName);

var model = FabricRestApi.CreateItem(workspace.Id, modelCreateRequest);

// create new connection for semantic model

var url = PowerBiRestApi.GetWebDatasourceUrl(workspace.Id, model.Id.Value);

var connection = FabricConnectionsApi.CreateAnonymousWebConnection(url);

// bind connection to semantic model using Power BI REST API

PowerBiRestApi.BindSemanticModelToConnection(workspace.Id, model.Id.Value, new Guid(connection.id));

// refresh semantic model using Power BI REST API

PowerBiRestApi.RefreshDataset(workspace.Id, model.Id.Value);

Currently, Power BI REST API required to bind connection and refresh semantic model. Developer binds semantic model to connection using BindToGatewayInGroup API.

public static void BindSemanticModelToConnection(Guid WorkspaceId, Guid SemanticModelId, Guid ConnectionId) {

BindToGatewayRequest bindRequest = new BindToGatewayRequest { DatasourceObjectIds = new List<Guid?>() };

bindRequest.DatasourceObjectIds.Add(ConnectionId);

pbiClient.Datasets.BindToGatewayInGroup(WorkspaceId, SemanticModelId.ToString(), bindRequest);

}

In the fullness of time, this binding and refresh functionality will be added to the Fabric REST API.

## Create Reports using the Create Item API

When creating report from item definition, you must bind report to semantic model. Report definition is maintained in report.json while definition.pbir binds report to specific semantic model.

When you create a new report, you need to bind it to a target semantic model. When you create a new semantic model, you need to capture item Id returned from Create Item. When creating the item definition for the report, you use item Id for semantic model for binding.

Let’s examine a template file for definition.pbir.

{

"version": "1.0",

"datasetReference": {

"byPath": null,

"byConnection": {

"connectionString": null,

"pbiServiceModelId": null,

"pbiModelVirtualServerName": "sobe\_wowvirtualserver",

"pbiModelDatabaseName": "**{SEMANTIC\_MODEL\_ID}**",

"name": "EntityDataSource",

"connectionType": "pbiServiceXmlaStyleLive"

}

}

}

Ssssssss

**// substitute target SemanticModelId into item part named definition.pbir**

string part1FileTemplate = FabricIsvPlaybook.Properties.Resources.definition\_pbir;

string part1FileContent = part1FileTemplate.Replace("**{SEMANTIC\_MODEL\_ID}**", SemanticModelId.ToString());

xxx

public static CreateItemRequest GetSalesReportCreateRequest(Guid SemanticModelId, string DisplayName) {

// substitute target SemanticModelId into item part named definition.pbir

string part1FileTemplate = FabricIsvPlaybook.Properties.Resources.definition\_pbir;

string part1FileContent = part1FileTemplate.Replace("**{SEMANTIC\_MODEL\_ID}**", SemanticModelId.ToString());

string part2FileContent = FabricIsvPlaybook.Properties.Resources.sales\_report\_json;

string part3FileContent = FabricIsvPlaybook.Properties.Resources.CY24SU02\_json;

var createRequest = new CreateItemRequest(DisplayName, ItemType.Report);

createRequest.Definition =

new ItemDefinition(new List<ItemDefinitionPart>() {

CreateInlineBase64Part("definition.pbir", part1FileContent),

CreateInlineBase64Part("report.json", part2FileContent),

CreateInlineBase64Part("StaticResources/SharedResources/BaseThemes/CY24SU02.json", part3FileContent),

});

return createRequest;

}

Dddddd

var workspace = FabricRestApi.CreateWorkspace(WorkspaceName);

// create semantic model

var modelCreateRequest = FabricItemDefinitionFactory.GetImportedSalesModelCreateRequest(ImportedModelName);

var model = FabricRestApi.CreateItem(workspace.Id, modelCreateRequest);

// prepare semantic model

var url = PowerBiRestApi.GetWebDatasourceUrl(workspace.Id, model.Id.Value);

var connection = FabricConnectionsApi.CreateAnonymousWebConnection(url);

PowerBiRestApi.RefreshDataset(workspace.Id, model.Id.Value);

// create CreateItemRequest object with target semantic model Id

var createRequestReport =

FabricItemDefinitionFactory.GetSalesReportCreateRequest(model.Id.Value, ImportedModelName);

// create report which is bound to semantic model created earlier

var report = FabricRestApi.CreateItem(workspace.Id, createRequestReport);

We have just implemented the classic Power BI provisioning flow for an import-mode semantic model and an associated report.

1. Create workspace
2. Create semantic model
3. Create connection
4. Bind connection to semantic model
5. Refresh semantic model
6. Create report bound to semantic model

Report definition has dependencies on names of tables, columns and measures in semantic model. Report has no dependencies on the type of underlying semantic model. A single report definitions can be used across different types of semantic models. Shared report definition can be bound to import-model semantic models and DirectLake semantic models.

## Export Existing Items using Get Item Definition

Get Item Definition API allows you retrieve item definition for existing items. Call Get Items to discover set of item Ids in workspace then call Get Item Definition once for each item Id. Make it possible to create a folder on local file system with set of files for each item definition part

A close-up of a sign

Description automatically generated

POST request to the follow URL.

https://api.fabric.microsoft.com/v1/workspaces/{WORKSPACE\_ID}/items/{ITEM\_ID}/getDefinition

This API is processed as a long running operation (LRO).

https://api.fabric.microsoft.com/v1/workspaces/{WORKSPACE\_ID}/items/{ITEM\_ID}/getDefinition?format=TMSL

This is what you get

A screenshot of a computer

Description automatically generated

Dump to local file system

A close-up of a white background

Description automatically generated

Xxxx

A screenshot of a computer program

Description automatically generated

Xxx

A screenshot of a computer

Description automatically generated

## Update Workspace Items using Update Item

This API is used to update two generic properties that are common across all workspace items. These two properties include display name and description.

public static Item UpdateItem(Guid WorkspaceId, Guid ItemId, string DisplayName, string Description = null) {

var updateRequest = new UpdateItemRequest {

DisplayName = DisplayName,

Description = Description

};

return fabricApiClient.Core.Items.UpdateItem(WorkspaceId, ItemId, updateRequest).Value;

}

### Update Workspace Items using Update Item Definition

You cannot use the Update Item API when you need to update the underlying item definition for a workspace item such as a semantic model or a report. Instead, you must call the Update Item Definition API and pass an item definition that includes the required changes.

A close-up of a sign

Description automatically generated

public static UpdateItemDefinitionRequest GetImportedSalesModelUpdateRequest(string DisplayName) {

string part1FileContent = FabricIsvPlaybook.Properties.Resources.definition\_pbism;

string part2FileContent = FabricIsvPlaybook.Properties.Resources.sales\_model\_import\_v2\_bim;

return new UpdateItemDefinitionRequest(

new ItemDefinition(new List<ItemDefinitionPart>() {

CreateInlineBase64Part("definition.pbism", part1FileContent),

CreateInlineBase64Part("model.bim", part2FileContent)

}));

}

Just like when constructing an item definition for a Create Item, you prepare the exact same type of item definition when you call Update Item Definition.

Note you must include every item definition part when calling the Update Item Definition API. You cannot just pass an item definition with the parts that have changed.

Here is some code.

// get item definition with updated verion 2 file for model.bim

var updateModelRequest = FabricItemDefinitionFactory.GetImportedSalesModelUpdateRequest(ImportedModelName);

// call

fabricApiClient.Core.Items.UpdateItemDefinition(WorkspaceId, ItemId, updateModelRequest);

Send POST request to this URL.

https://api.fabric.microsoft.com/v1/workspaces/{WORKSPACE\_ID}/items/{ITEM\_ID}/updateDefinition

Use Update Item Definition API to update semantic models and reports. You can update semantic models and reports completely independent of one another

A computer screen shot of a computer program

Description automatically generated

Now update the report. Demo with Version 2 of report has updated layout and uses new built-in theme. Updated report content for layout maintained in sales\_report\_v2\_json resource. Report update includes adding new built-in theme named NewExecutive.json.

public static UpdateItemDefinitionRequest GetSalesReportUpdateRequest(Guid SemanticModelId, string DisplayName) {

string part1FileContent = FabricIsvPlaybook.Properties.Resources.definition\_pbir.Replace("{SEMANTIC\_MODEL\_ID}", SemanticModelId.ToString());

string part2FileContent = FabricIsvPlaybook.Properties.Resources.sales\_report\_v2\_json;

string part3FileContent = FabricIsvPlaybook.Properties.Resources.CY24SU02\_json;

string part4FileContent = FabricIsvPlaybook.Properties.Resources.NewExecutive\_json;

return new UpdateItemDefinitionRequest(

new ItemDefinition(new List<ItemDefinitionPart>() {

CreateInlineBase64Part("definition.pbir", part1FileContent),

CreateInlineBase64Part("report.json", part2FileContent),

CreateInlineBase64Part("StaticResources/SharedResources/BaseThemes/CY24SU02.json", part3FileContent),

CreateInlineBase64Part("StaticResources/SharedResources/BuiltInThemes/NewExecutive.json", part4FileContent)

}));

}

Sss

A screenshot of a computer

Description automatically generated

### Clone Workspace Items to a New Workspace

Fabric REST API makes it possible to clone workspaces. Enumerate through all items in source workspace using ListItems. Read item definitions from source workspace using GetItemDefinition. When required, update item definition of source item.

A diagram of a process

Description automatically generated with medium confidence

Steps to cloning semantic models

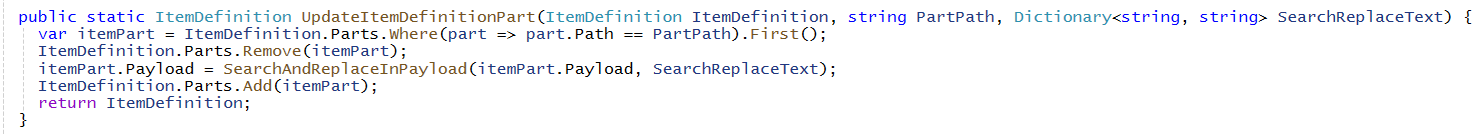
* Create and bind connections
* Refresh import-mode semantic models
* Create dictionary with semantic model Ids to redirect reports from source workspace to clone workspace

Report item definitions require update. You must substitute semantic model Ids to reference semantic models in clone workspace

A screen shot of a computer code

Description automatically generated

X



Now this.

A screen shot of a computer code

Description automatically generated

# Deploy Power BI Solutions with Semantic Models and Reports

Power BI Desktop supports Power BI Desktop Developer Mode. Power BI Desktop has traditionally allowed saving projects in PBIX file format. Power BI Desktop now allows saving projects in Power BI Project (PBIP) format.

* Power BI Desktop saves semantic model definition using model.bim and definition.pbism
* Power BI Desktop saves report definition using report.json and definition.pbir

A screenshot of a computer

Description automatically generated

.pbi folder used to store local resources that should not be saved into source control system.

To use PBIP formats, you must enable **Power BI Project (.pbip) save option** in Power BI Desktop Options dialog.

A screenshot of a computer program

Description automatically generated

Power BI Desktop offers a feature to save the item definitions for semantic models and report in more-modern and advanced formats. Optionally configure semantic models and reports to use newest definition formats.

A white background with black text

Description automatically generated

## TMSL versus TMDL - Semantic Model Formats

Semantic Model Definitions can be saved in TMSL format or TMDL format

* TMSL is simple - a single model.bim file contains entire semantic model definition
* TMDL is more granular – better for source control when multiple developers are working on same model

A screenshot of a computer

Description automatically generated

## PBIR-Legacy versus PBIR - Report Definition Formats

Report Definitions can be saved in PBIR-Legacy format or PBIR format. PBIR-Legacy is original Power BI report definition format with entire definition stored in report.json file. PBIR is modern, more granular format which allows for editing/creating definitions using JSON schema

A screenshot of a computer

Description automatically generated

# Deploy Solutions with Lakehouses and Notebooks

Let’s start with Fabric lakehouse fundamentals. Lakehouse used to store structured data and unstructured data in a single location. Lakehouse data transparently stored in OneLake using Azure Data Lake Storage Gen2 (ADLS Gen2). Lakehouse data accessible across workloads like data engineering, data warehouse, data science & BI.

Lakehouse enables storing and refining data using medallion architecture (bronze>silver>gold)

Lakehouse has a specific structure. The **Tables** section used to store structure data in tables in Delta format. **Files** section used to store unstructured data as files.

A screenshot of a computer

Description automatically generated

Ingesting Data into Lakehouse.

* Running Spark jobs
* Running Data Pipelines
* Creating Shortcuts
* Using ADLS Gen2 APIs
* Running Fabric Gen2 Dataflows

For running Spark jobs, you can run them as notebooks or as Spark Job Definitions

For running Data Pipelines. You can build a data pipeline with activities to copy data from external sources

Creating Shortcuts allows you to create references to make files from an external source appear to be in lakehouse storage

You can use ADLS Gen2 APIs to copy files into the File section of a lakehouse. This is implemented by client application which uploads files into lakehouse storage using push approach

Running Fabric Gen2 Dataflows. Running a dataflow which imports data and saves it delta tables in lakehouse storage

### The Apache Spark Runtime

Fabric Provides Apache Spark Runtime. Spark is a unified engine for large-scale data analytics. Spark code can be executed through Spark Job Definition. Spark code can be executed using interactive Fabric Notebook.

A diagram of a spark session

Description automatically generated

Apache Spark provides APIs to read and write data to/from Lakehouse in OneLake

Steps to provisioning flow which builds out medallion architecture

* **Create bronze layer** by coping CSV files into **Files** section
* **Create silver layer** by loading CSV data into dataframes and saving them as delta tables with OLTP schema
* **Create gold layer** by transforming data from silver tables into delta tables with star schema

A screenshot of a computer

Description automatically generated

You can leverage Fabric support for Spark to create and populate lakehouse tables

* Create lakehouse (or multiple lakehouses for greater isolation)
* Create Spark Environment as workspace item for running Spark jobs (when Environments API becomes available)
* Create Notebook (or Spark Job Definition) containing ETL logic to ingest data files and to create tables
* Run Notebook using Job Scheduler – Run Job On-demand API

### Create a Lakehouse using the Create Item API

Lakehouse created without item definition – you only pass displayName and type.

public static Item CreateLakehouse(Guid WorkspaceId, string LakehouseName) {

// Item create request for lakehouse des not include item definition

var createRequest = new CreateItemRequest(LakehouseName, ItemType.Lakehouse);

// create lakehouse

return CreateItem(WorkspaceId, createRequest);

}

xx

A screenshot of a computer

Description automatically generated

Response from Create Item includes lakehouse id which is required to create notebooks

A screenshot of a computer code

Description automatically generated

### Create a Notebook using the Create Item API

Notebook created calling **CreateItem** with notebook part **CreateLakehouseTables.ipynb**

Notebook code and all other content stored in single JSON file with **ipynb** extension

Fabric also supports calling **CreateItem** with standard Python file with **py** extension

### Create Spark Environments using Create Item

ooeoe

### Configure Workspace Spark Settings

Code used in developer sample application to create lakehouse

### Create a DirectLake Semantic Models on a Lakehouse

# Deploy Solutions using Warehouses and Data Pipelines

# Deploy Solutions using Real-time Intelligence