# Create and Update Workspace Items

Deploying a Fabric solution typically involves creating a new workspace and then populating the workspace by creating and configuring workspace items. To be a productive Fabric developer, you must learn the skills to create a set of workspace items that are configured to work together to acquire, transform, analyze and present data.

This chapter teaches the fundamentals of creating and managing workspace items.

That is, all workspace items can be seen as a generic item type with a id and display name. This makes it possible to discover what items exist within a workspace.

The Fabric REST APIs build on top of an essential abstraction of item definition. The workspace item becomes a key ingredient for creating many of the workspace item items

## Workspace Item CRUD APIs

### Create Workspace Item without an Item Definition

### Create Workspace Item using an Item Definition

Working

This chapter covers the essential topic of item definitions. Item

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

### List Workspace Items

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

A screenshot of a computer

Description automatically generated

Call List Items

GET https://api.fabric.microsoft.com/v1/workspaces/{workspaceId}/items

Returns JSON result with value property containing

{

"value": [

{

"id": "11111111-1111-1111-1111-111111111111",

"displayName": "sales",

"description": "A lakehouse description",

"type": "Lakehouse",

"workspaceId": " 99999999-9999-9999-9999-999999999999"

},

{

"id": " 22222222-2222-2222-2222-222222222222",

"displayName": "Create Lakehouse Tables",

"description": "A notebook description",

"type": "Notebook",

"workspaceId": "99999999-9999-9999-9999-999999999999"

}

]

}

Here is C# code using .NET SK

List<Item> items = fabricApiClient.Core.Items.ListItems(WorkspaceId).ToList();

foreach (var item in items) {

Console.WriteLine($"{item.DisplayName} is a {item.Type} with an id of {item.Id}");

}

If the list of items is large enough, the List Items API has the potential to return paginated results. Don’t forget to look for continuation tokens.

### List a Specific Type of Workspace Item

xxx

GET https://api.fabric.microsoft.com/v1/workspaces/{workspaceId}/items**?type=Lakehouse**

Xxxx

List<Item> notebookItems = fabricApiClient.Core.Items.ListItems(WorkspaceId, type:"Lakehouse").ToList();

foreach (var item in notebookItems) {

Console.WriteLine($"{item.DisplayName} is a notebook with an id of {item.Id}");

}

There is another way to accomplish this by going through the item-specific endpoint.

GET https://api.fabric.microsoft.com/v1/workspaces/{workspaceId}/notebooks

X

{

"value": [

{

"id": "11111111-1111-1111-1111-111111111111",

"type": "Lakehouse",

"displayName": "sales",

"description": " A lakehouse description",

"workspaceId": "99999999-9999-9999-9999-999999999999",

"properties": {

"oneLakeTablesPath": "https://onelake.dfs.fabric.microsoft.com/{WORKSPACE\_ID}/{LAKEHOUSE\_ID}/Tables",

"oneLakeFilesPath": "https://onelake.dfs.fabric.microsoft.com/{WORKSPACE\_ID}/{LAKEHOUSE\_ID}/Files",

"sqlEndpointProperties": {

"connectionString": "abcdef12345678.datawarehouse.fabric.microsoft.com",

"id": "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa",

"provisioningStatus": "Success"

}

}

}

]

}

x

### Query Item-specific Properties

## Create Workspace Items

The primary way to create workspace items is using the Create Item API. You will learn about several different ways to accomplish this in this chapter.

Item definition represent an essential aspect of the Fabric REST API programming model.

Here’s what a bit confusing. Some types of workspace items are created by creating an item definition and passing it in a call to the Create Item API. Other workspace items are created by calling the Create Item API without passing an item definition. For example, you can create a lakehouse without using an item definition. However, you need to use a

### Call the Create Item API without an Item Definition

### 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

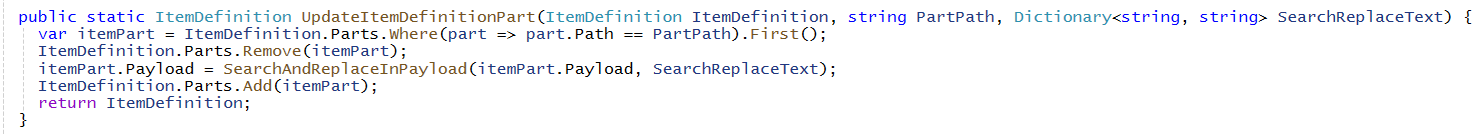
* Create and bind connections
* Refresh import-mode semantic models
* Create dictionary with semantic model Ids to redirect reports from source workspace to clone workspace

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