Achieving composability: Graph

Session-ID: IN181

Exercises / Solutions

Stephanie Lewellen, Product Manager (Graph), SAP
Chaim Bendelac, Product Manager (API Management), SAP

SAP TECHED

TABLE OF CONTENTS

INTRODUCTION

Graph is a new and innovative capability of API Management, part of the SAP Integration Suite. With Graph, developers access your business data as a single semantically connected **data graph**, spanning the suite of SAP products and beyond. Graph's powerful API reduces the cost and complexity of creating and deploying reusable extensions and other client applications.

Enterprise landscapes continue to expand in scale and complexity. Each additional system, SaaS, or micro-service introduces new protocols, data models, connectivity, and security conventions. Real-world problems often span multiple lines of business, services, and APIs. Consequently, even the most experienced developers struggle to understand all technologies and interfaces involved; developing new business-extending client apps requires an ever-growing range of expertise and skills. The phenomenal adaption of low-code tools by casual developers further increases the challenge.

Enterprises use API Management to partially address this gap: APIs can be renamed, authentication can be streamlined, APIs can be protected against unauthorized access or threats. But this doesn't address the deeper problem: separate, disconnected APIs from multiple, different data sources and systems.

Graph unifies your business APIs in the form of a semantically connected **data graph**, accessed via a single powerful API. Out-of-the-box, it provides developers a single connected and unified view of your SAP-managed business data. Graph consolidates thousands of data entities of SAP systems like SAP S/4HANA, SAP Sales Cloud, and SAP SuccessFactors into one curated, semantically connected, data model. We call this connected data graph a *Business Data Graph*.

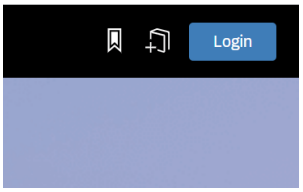
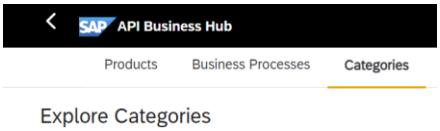
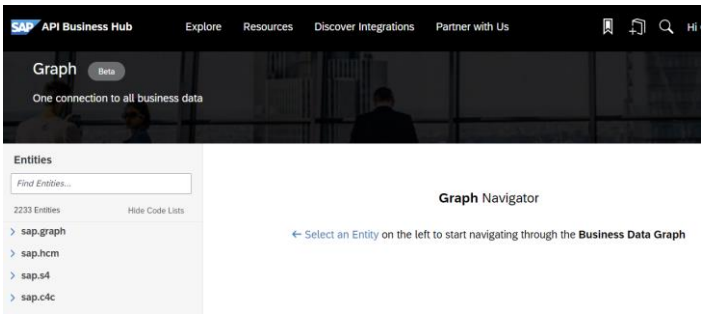
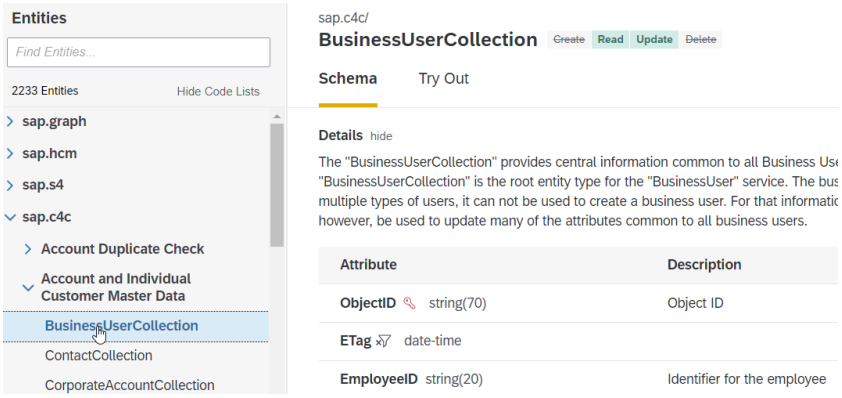
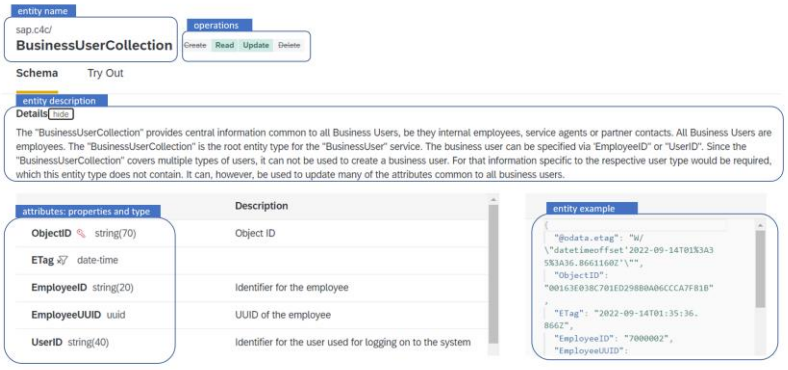
The out-of-the-box data graph of SAP-managed data is the baseline, the starting point to your own data graph. Expand it by adding your own data sources and your own data models, projections, and compositions, to create a unique data model of your own business. The business data graph is ultimately an abstraction of the data in your landscape. It hides the *intricacies* of that landscape from developers, by exposing the data through a single unified data graph and API.

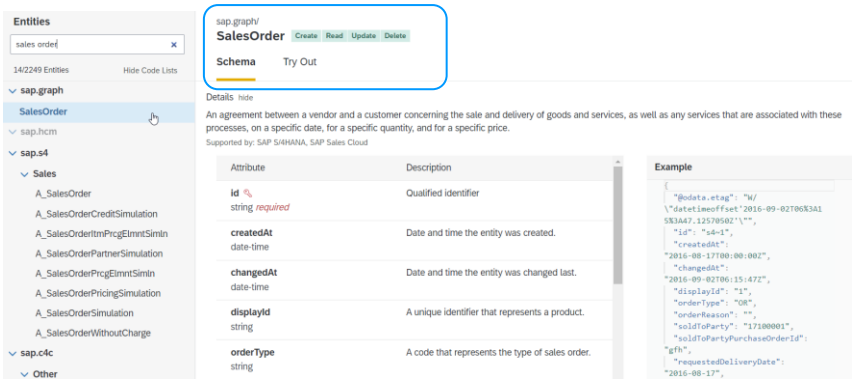
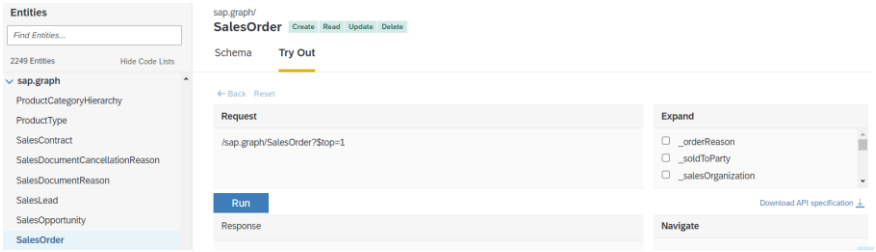
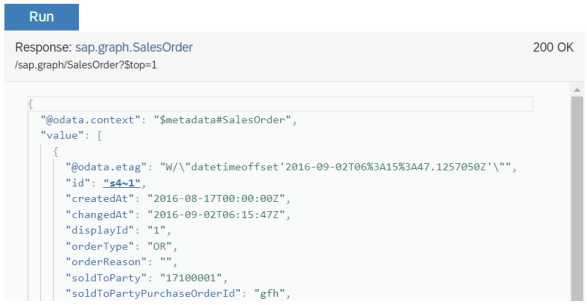
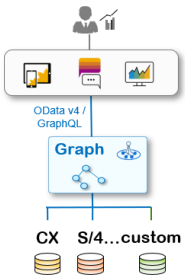
Developers use standard and powerful data graph query languages (OData v4, or GraphQL) to efficiently navigate the data, without being exposed to the complexity of data sources, URLs, connections, replications, VPNs, or underlying security concerns. All the data, through one API. Often a single powerful graph-navigating query can replace complicated programming logic required to issue repeated queries to separate systems or APIs.

Because the decoupling of the system landscape from applications, enterprises can deploy Graph-based applications more easily, across more landscapes, and at a lower cost.

WHAT YOU WILL LEARN

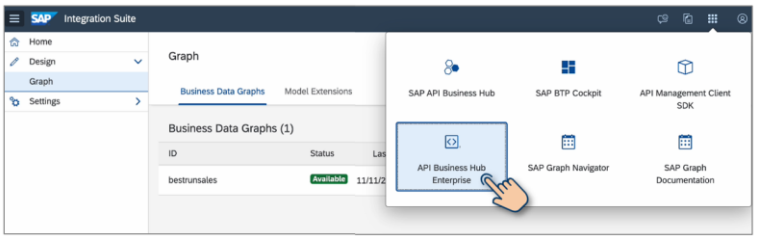
- How to explore the Graph data model and API
- How to use the graph Navigator to explore and navigate your own data, using OData or GraphQL

Explanation	Screenshot
1. Using your browser, go to SAP's API Business Hub at https://api.sap.com/ 2. If not already logged in, you can log in with your SAP id. 3. At the top of the screen, select "categories" or scroll down, until you see the title "Explore Categories"	 
4. Click on the Graph tile	
5. The Graph Navigator opens. The Graph Navigator is a very useful tool to search for, learn about, navigate and explore the rich Graph API. On the left, the **Entities** index, you see a search window, and four namespaces: collections of the available entities.	
6. The bottom three namespaces contain the business objects from SAP SuccessFactors (hcm), SAP S/4 HANA (s4) and SAP sales and Service Cloud (c4c) respectively. Open one of these three namespaces. For instance, pick the **sap.c4c** namespace, and then select **BusinessUserCollection** under Account … Master Data.	
7. You can easily see which *operations* are allowed (read and update, but not create or delete), review the description and the list of attributes, and see an actual data example. Take the time to explore over 2,000 *root entities* (representing 4,500+ entities and sub-entities in SAP systems), in the different namespaces, within the Navigator context.	

Explanation	Screenshot
8. Today, we will explore one of the *unified* entities in the **sap.graph** namespace, the top namespace in the list on the right. Unified entities are *simplified projections* on some of the most complex business objects in SAP systems. They are useful in particular for apps that only require basics, essential information. For instance, you can search for ***sales order*** (or **SalesOrder**) in the search box on the left, and pick the first search result: **sap.graph/SalesOrder.** This too is a synthetic entity, a *projection* on the actual sales information in your systems.	
9. Now switch to the **Try Out** tab at the top. If this is not what you see, you may not be logged in yet. Please log in it now.	
10. You can see an OData request, for one record of sap.graph/SalesOrder ($top=1). Click **Run.** If all is well, you will see your Sales Order appear on the screen.	
11. But wait a minute. Whose data are we looking at? API Business Hub, as an API reference site, maintains a set of "*sandbox servers*" with mock data. In reality, you will want to use Graph to access **your *own* enterprise data**. The rest of this tutorial will use a business data graph that was established as part of a fictional enterprise, called BestRun.	 <= **BestRun** data sources

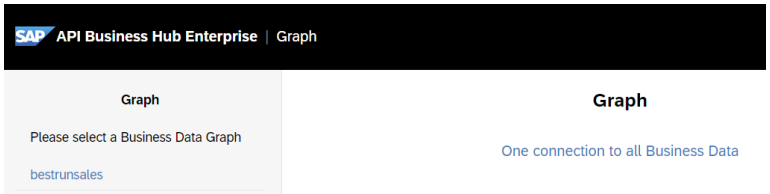
12. Following the installation of the Graph function in Integration Suite, you can navigate from there to Business Hub Enterprise.

    This is your developer portal, where you can find your own APIs.

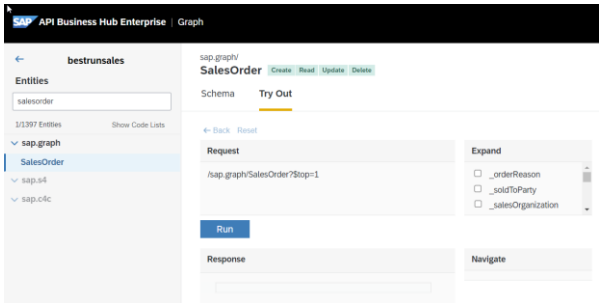


13. In our fictional example, we have defined only one business data graph, called bestrunsales.

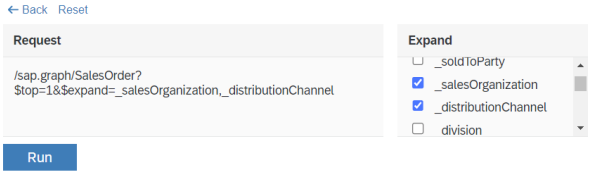
    Select it.



14. Search for "salesorder", select the first search result and switch to the Try Out tab.



15. Now let's have some fun. Check to expand **_salesOrganization** and (after scrolling) **_distributionChannel.** Did you see how this automatically change the OData request?



16. Click **run** to see the result of this query. Scroll down to see the expanded sales organization and distribution channel information.

    This is a (mock) order from one of BestRun's direct sales customers in the US.


```
"soldToParty": "10004",
"soldToPartyPurchaseOrderId": "",
"requestedDeliveryDate": "2015-05-02",
"salesOrganization": "US1100",
"_salesOrganization": {
  "id": "c4c~US1100",
  "displayId": "US1100",
  "name": "Sales Unit US",
  "company": "1000",
  "currency": ""
},
"distributionChannel": "01",
"_distributionChannel": {
  "id": "c4c~01",
  "displayId": "01",
  "name": "Direct sales"
},
```


17. We could try out different variations of "$expand", to get of sense of all the data. Here is another fun query we can try:

    /sap.graph/SalesOrder?$top=1&$expand=items($expand=_product)

    and learn that the customer ordered a TV which is of product type 3.


```
"items": [
  {
    "itemId": "10",
    "parentItemId": "",
    "itemCategory": "ORN",
    "itemText": "Television 3D (55)",
    "soldToPartyPurchaseOrderItemId": "",
    "product": "P100314",
    "_product": {
      "id": "c4c~P100314",
      "createdAt": "2015-03-13T10:07:08Z",
      "changedAt": "2020-04-06T11:09:56Z",
      "displayId": "P100314",
      "name": "Television 3D (55)",
      "productType": "3",
      "status": "3",
      "baseUnit": "EA",
      "division": ""
    },
    "quantity": 1,
```

18. When dealing with a specific single record, we can also *navigate across* the data to access other pertinent information. Let's navigate from our order to its customer (a business partner):

    /sap.graph/SalesOrder/c4c~9000000000/_soldToParty


```
"id": "c4c~10004",
"createdAt": "2012-11-12T18:04:34Z",
"changedAt": "2020-04-08T16:09:55Z",
"displayId": "10004",
"name": "Bluedrive",
"language": "EN",
"type": "_corporateAccount"
```


19. This is a corporation, so we further navigate to:

    /sap.graph/SalesOrder/c4c~9000000000/_soldToParty/_CorporateAccount

    Let's also add some information to the query before we run it:

    ?$expand=addresses,contacts


```
"contacts": [
  {
    "id": "00163E03A0701ED28B9E6D3DC04E7197",
    "contactPerson": "1000087",
    "isMainContact": true,
    "department": "0001",
    "function": "0002",
    "phoneNumber": "+1 6503275564",
    "faxNumber": "+1 4082953822",
    "emailAddress": "michaelbluedrive@gmail.com"
  },
```


Etc.

20. Expanding and navigating are two powerful capabilities. But what makes data graphs even more useful are queries that combine these capabilities with other features, such as:

    $select: return only the attributes you need
    $filter: use data selection criteria
    $orderby: return results in sort-order.

    For instance, the following query list ten sorted (ascending order) quotes by amount greater than 2000, with customer names and sales organization:

/sap.graph/SalesQuote?
$top=10&
$expand=_soldToParty($select=name)&
$filter=netAmount ge 2000&
$select=id,netAmount,salesOrganization&
$orderby=netAmount


```
{
  "@odata.context": "$metadata#SalesQuote(id,netAmount,salesOrganization,_soldToParty(name))",
  "value": [
    {
      "@odata.etag": "W/\"datetimeoffset'2020-04-24T05%3A24%3A58.3192950Z'\"",
      "id": "c4c~231",
      "_soldToParty": {
        "id": "c4c~1001",
        "name": "Porter LLC"
      },
      "salesOrganization": "US1100",
      "netAmount": 2000
    },
    {
      "@odata.etag": "W/\"datetimeoffset'2020-04-24T05%3A24%3A58.3192950Z'\"",
      "id": "c4c~644",
      "_soldToParty": {
        "id": "c4c~1010296",
        "name": "Bike World"
      },
      "salesOrganization": "US1100",
      "netAmount": 2107.89
    },
    {
      "@odata.etag": "W/\"datetimeoffset'2020-04-24T05%3A24%3A58.3192950Z'\"",
      "id": "c4c~463",
      "_soldToParty": {
        "id": "c4c~1005201",
        "name": "KM"
      },
      "salesOrganization": "US1100",
      "netAmount": 2325
```

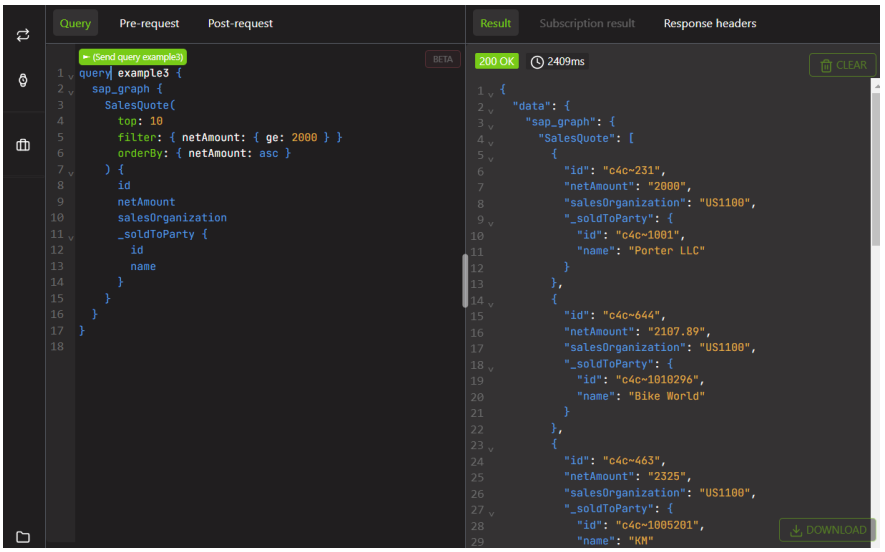

21. Many developers prefer to use GraphQL, as an alternative to OData. GraphQL uses an intuitive JSON specification to describe the requested data, and easily allows a developer to specify a structured data response.

    Let me show you what that looks like, using the same example shown above. Here is what this query looks like in GraphQL:

```
query example3 {
 sap_graph {
    SalesQuote (
        top: 10
        filter: {netAmount: {ge: 2000}}
        orderBy: {netAmount: asc}
    ) {
      id
      netAmount
      salesOrganization
      _soldToParty {
        id
        name
}}}}
```

To execute the query, we switch to an interactive GraphQL client, such as the Altair GraphQL client. On the left you can see the pasted query, on the right, the response pane.

Note how the response nicely matches the requested (nested) data structure.



www.sap.com/contactsap

THE BEST RUN SAP