Oracle® Cloud

Accessing Business Objects Using REST APIs

E92922-03
March 2019

Oracle Cloud Accessing Business Objects Using REST APIs,

E92922-03

Primary Author: Oracle Corporation

Contents

Preface

Accessing Business Objects Using REST APIs describes the supported HTTP methods, HTTP headers, request URL parameters, media types, and other concepts of the REST APIs and the use cases that they support for making REST API calls in web applications created using visual development tools offered by Oracle.

Audience

This document is intended for developers who want to create and publish modern enterprise web applications using visual development tools, including using REST APIs generated by Oracle tooling to access the data of business objects exposed in the web application.

Related Resources

For more information, see these Oracle resources:

- Oracle Public Cloud

  http://cloud.oracle.com

- About Oracle Cloud in *Getting Started with Oracle Cloud*

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements (for example, menus and menu items, buttons, tabs, dialog controls), including options that you select.

Convention	Meaning
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates language and syntax elements, directory and file names, URLs, text that appears on the screen, or text that you enter.

Part I
Getting Started

To interact with business objects using REST APIs you should be familiar with how to access the resources backing the business objects.

Topics

1

Introduction to Accessing Business Objects

You can use REST APIs that rely on HTTP requests and responses as the interface to access the business objects of your domain.

Topics

About Accessing Business Objects

You can access business objects by making REST API calls enabled by the REST API framework.

In web applications, REST resources acted on by REST APIs are backed by business objects exposed in the visual development tool. For example, web application developers working with Oracle Visual Builder can decide on the set of attributes to expose on the business objects and the actions to make available (both standard CRUD operations and custom methods).

The design-time choices that you make when creating business objects allow the tooling to generate metadata that it uses to define REST resources and REST APIs. The data used by the web application is shaped by the resource's backing business object, with the parent-child relationships intact. Using these REST APIs exposed by the tooling, you may interact with business objects to access its data. As a result, your application may invoke CRUD operations to interact with the REST resources and business objects.

Use Cases and Examples

As a web application developer, working in an Oracle visual development tool like Oracle Visual Builder, you can generate REST API endpoints to manage and interact with the business objects of the application.

Here are the types of things that you can do using REST APIs to access the data of business objects:

- Get a description of the REST resource, including the resource collection attributes and available actions.
- Interact with the REST resource using standard HTTP request methods, including GET, POST, PATCH, and DELETE.
- Allow the server to decide whether to create or update depending on whether the record exists or not.

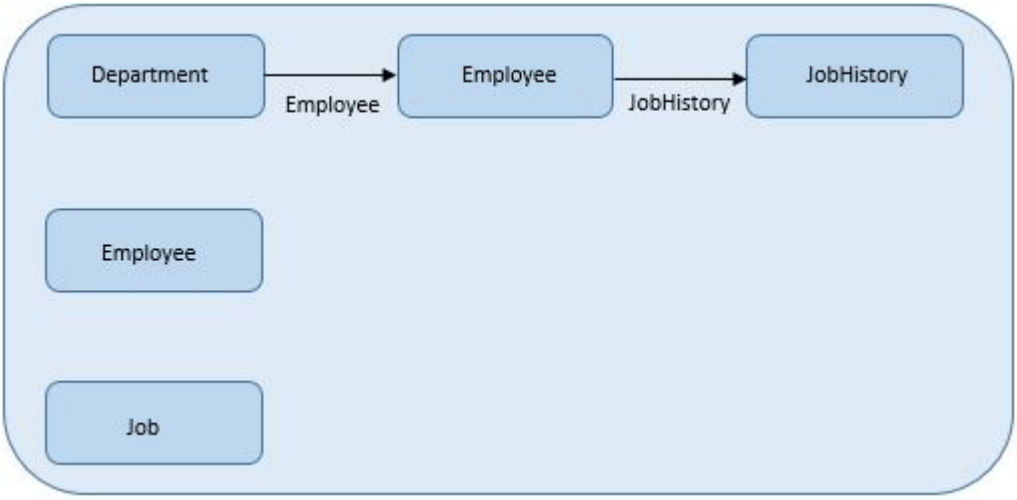
- Specify a framework version in a request to interact with the old format when the REST API Framework has made backward incompatible changes.
- Perform advanced queries and sorting on a resource collection and shape the returned payload using URL parameters.
- Use ETag for change detection and optimistic concurrency control.

To understand how certain design time features used to create web applications support the availability of these REST API capabilities, consult the Oracle documentation for your visual development tool.

About the Resource Samples in This Guide

This guide describes typical use cases for interacting with and manipulating business objects. All samples are based on DEPARTMENTS, EMPLOYEES, JOBHISTORY, and JOBS tables in the Oracle HR schema. This figure depicts the business objects generated from these tables in the web application.

Figure 1-1 Business Objects Used with REST APIs in this Guide



Understanding the REST API Framework

REST APIs are supported by the REST API framework to access business objects. The REST API framework supports the exchange of information between the web application and server at runtime.

The REST API framework is an Oracle framework that allows web application developers to expose a REST API based on the REST architectural style. The framework itself does not constitute a Web API, but supports creating and interacting with the business objects. The REST API framework determines the functionality to interact with business objects created by web application developers.

In the web application, REST resources acted on by REST APIs are backed by business objects exposed in the web application. When you work in an Oracle visual development tool, like Oracle Visual Builder, you can decide on the set of attributes to expose from business objects and the actions to make available (both standard CRUD operations and custom object functions defined by the web application).

The design-time choices that you make are captured as metadata by the tooling, which the tooling uses to generate REST resource definitions. You may interact with these resource definitions through the REST API, supported by the REST API framework. For example, you may invoke CRUD operations to interact with the REST resources and business objects, where the data is shaped by the resource's backing business object, with the parent-child relationships intact.

Oracle releases may introduce new REST API framework functionality to support additional business object interaction use cases. To allow you to manage the level of functionality exposed to customers in your production applications, Oracle defines numeric versions of the framework that refer to a specific level of the REST API framework functionality. As Oracle introduces new framework versions, you may opt into the new version to gain the new functionality, or you may decide not to opt in and instead preserve the level of functionality supported by the current REST API framework version.

Understanding Business Objects as REST API JSON Objects

REST APIs represent the business object as a JSON-encoded resource collection.

REST resource collections are synonymous with business objects created by web application developers. For example:

- A `Department` resource collection is based on a `Department` business object.
- An `Employee` resource collection is based on an `Employee` business object.

The payload returned by REST APIs contains one or more resource collections, comprised of the resource items queried by the REST API and the individual items of the business object. The resource collections preserve the relationship of master-detail coordinating business objects.

As the table below shows, the *resource collection* is the REST API payload representation of all items of a particular business object. The *resource items* are the rows and attributes of the payload item object, which represent the items of the business object.

> **Note:**
>
> The format of resource collections and contained items are defined by specific REST API media types as JSON-encoded entities. For more details, see Media Types.

Table 1-1 JSON Objects and Business Object Representation

JSON Object	Business Object
`resource collection`	A business object comprised of one or more items. `Department`, `Employee` are examples of resource collections that represent `Department` and `Employee` business objects.

Table 1-1 (Cont.) JSON Objects and Business Object Representation

JSON Object	Business Object
`resource item`	An item of a business object. The specific department `10` or employee `1012` are examples of resource items that represent items of the `Department` and `Employee` business object.

Understanding Framework Support for Query Syntax

REST API calls can make use of a query expression syntax to query business objects.

Beginning with REST API framework version 2, REST API calls can make use of an expanded query expression syntax to query business objects. Note that version 2 and later will interpret the `q` query parameter value differently than framework version 1, and therefore opting into framework version 2 or later introduces a backward incompatible change to web applications that rely on framework version 1.

When you decide to opt into framework version 2 (or later), REST APIs calls will process fetch requests for the `q` query parameter using the expanded expression syntax, whereas requests using the version 1 “query-by-example” syntax will become invalid and will return an error. If you do not opt into framework version 2 or later, the default for your release version will be framework version 1. Alternatively, you may preserve the base functionality by creating a new release version identifier that you associate with framework version 2, while leaving the existing release identifier defined in the web application as framework version 1.

In version 1, filtering business object collections using query parameters is limited to a query-by-example syntax, which separates expressions using a semi-colon, as follows:

```
GET <base_url>/Department?q=Dname SA*;Loc BOSTON
```

Whereas, starting in version 2, a new advanced query syntax supports filtering business object collections using rowmatch expressions, as follows:

```
GET <base_url>/Department?q=Dname like 'SA*' or Loc = 'BOSTON'
```

Such expressions include the case-insensitive name of a resource item, followed by an operator and one or more operand values (depending on the operator used). The filter can be as simple as a single expression, or it can combine expressions using the `and` and `or` conjunctions with matching sets of parentheses for grouping.

Benefits of the Advanced Query Syntax Offered in Framework Version 2 and Later

The advantages of rowmatch expression include the following.

- They may use supported operators:

  DepartmentNumber = 20

  DepartmentNumber <> 20

  DepartmentNumber <= 20

  DepartmentNumber < 20

DepartmentNumber >= 20

DepartmentNumber >20

DepartmentNumber between 20 and 40

DepartmentNumber not between 20 and 40

DepartmentNumber in (20, 30, 40)

DepartmentName like '%S%'

DepartmentName like 'RE%'

DepartmentName not like 'RE%'

Location is null

Location is not null

- They may involve multiple attributes:

  DepartmentNumber = 10 or DepartmentName like 'RESEARCH'

  DepartmentNumber > 10 and DepartmentNumber < 40

  DepartmentNumber < 20 or DepartmentNumber > 30

  (DepartmentNumber = 10 or DepartmentNumber = 30) and (DepartmentName like 'SALES')

  DepartmentNumber BETWEEN 20 and 40) and (Location like 'DAL%')

  (DepartmentNumber > 0 and DepartmentNumber < 100) and (DepartmentName <> 'SALES') and (Location not like 'NEW%')

  (DepartmentNumber = 10 or DepartmentNumber = 30) and (DepartmentName = 'ACCOUNTING' or DepartmentName = 'SALES')

  (DepartmentNumber = 10 and DepartmentName like 'ACC%') or (DepartmentNumber = 20 and DepartmentName like 'RES%')

  DepartmentName='ACCOUNTING' or (DepartmentName like 'R%' and Location like '%ALLA%')

  (DepartmentName like 'R%' and Loc like '%ALLA%') or DepartmentName='ACCOUNTING'

  (DepartmentName like 'R%' or Loc like '%ALLA%') or DepartmentName='ACCOUNTING'

  (DepartmentNumber between 20 and 40) and DepartmentNumber is not null

- They may involve attributes of nested child resources:

  Deptno > 5 and Emps.Job = 'MANAGER'

  Emps.Job = 'MANAGER' and Deptno > 5

  Deptno > 5 and (Emps.Job = 'MANAGER')

  (Emps.Job = 'MANAGER') and Deptno > 5

  (Deptno > 5) and (Emps.Job = 'MANAGER')

  (Deptno = 10 and Emps.Job = 'PRESIDENT') or (Deptno = 20 and Emps.Job = 'MANAGER')

  Deptno > 5 and Emps.Job = 'MANAGER' and Emps.Sal >= 2500

  Deptno > 5 and (Emps.Job = 'ANALYST' or Emps.Sal >= 4000)

(Deptno > 5 and Emps.Job = 'ANALYST') or Emps.Sal >= 4000

Emps.Job = 'ANALYST' or Emps.Job = 'SALESMAN'

Deptno > 5 and (Emps.Job = 'ANALYST' or Emps.Job = 'SALESMAN')

Deptno > 5 and Emps.Job = 'MANAGER' and Emps.DirectReports.Sal >= 2000

Deptno > 5 and (Emps.Job = 'MANAGER' or Emps.DirectReports.Sal >= 2000)

Deptno > 10 and (Emps.Job = 'MANAGER' and (Loc = 'NEW YORK' or Emps.Mgr=7698))

Deptno > 10 and (Emps.Job = 'MANAGER' or (Loc = 'NEW YORK' or Emps.Mgr=7698))

Deptno > 10 and (Emps.Job = 'MANAGER' or (Loc = 'NEW YORK' or Emps.Mgr=7698)) or Deptno = 40

Deptno > 10 and (Emps.Job = 'MANAGER' or (Loc = 'NEW YORK' or Emps.Mgr=7698)) or (Deptno = 40)

Deptno > 10 and (Emps.Job = 'MANAGER' or (Emps.DirectReports.Sal > 2000 and (Emps.DirectReports.Comm = 500 or Emps.DirectReports.Deptno > 10)))

Deptno > 10 and (Emps.Job = 'MANAGER' and (Emps.DirectReports.Sal >= 2000 and (Emps.DirectReports.Comm = 500 or Emps.DirectReports.Deptno > 10)))

- They may involve the UPPER function:

  UPPER(DepartmentName) = 'RESEARCH'

  UPPER(DepartmentName) = UPPER('research')

  UPPER(DepartmentName) like 'RES%' and UPPER(Location) like 'DAL%'

  UPPER(DepartmentName) like UPPER('research')

Overview of the Advanced Query Syntax Offered in Framework Version 2 and Later

The following are specific expression use case examples.

- To test whether a value is `null` you must use the `is null` or the `is not null` keywords:

  ```
  AssignedToId is null
  ```

  ```
  AssignedToId is not null
  ```

- For equality use the = sign, and for inequality use either the `!=` or the <> operators.

  ```
  AssignedToId = 100000000089003
  ```

  ```
  Priority != 1
  ```

  ```
  Priority <> 1
  ```

  ```
  ActivityType != 'RS'
  ```

  ```
  ActivityType <> 'RS'
  ```

- For relational comparisons, use the familiar <, <=, >, or <> operators, along with `between` or `not between`.

  ```
  Priority <= 2
  ```

  ```
  Priority < 3
  ```

```
Priority <> 1

Priority > 1

Priority >= 1

TotalLoggedHours >= 12.75

Priority between 2 and 4

Priority not between 2 and 4
```

- For string matching, you can use the `like` operator, employing the percent sign `%` as the wildcard character to obtain "starts with", "contains", or "ends with" style filtering, depending on where you place your wildcard(s):

  ```
  RecordName like 'TT-%'

  RecordName like '%-TT'

  RecordName like '%-TT-%'
  ```

- To test whether a field's value is in a list of possibilities, you can use the `in` operator:

  ```
  ActivityType in ('OC','IC','RS')
  ```

- You can combine expressions using the conjunctions `and` and `or` along with matching sets of parentheses for grouping to create more complex filters like:

  ```
  (AssignedToId is null) or ( (Priority <= 2) and (RecordName like
  'TT-99%'))

  (AssignedToId is not null) and ( (Priority <= 2) or (RecordName like
  'TT-99%'))
  ```

  When using the `between` or `in` clauses, you must surround them by parentheses when you join them with other clauses using `and` or `or` conjunctions.

Testing the REST API

You can test REST APIs to make requests and interact with the business objects outside of your web application development tool.

You use the visual development tool to create the web application and the business objects that your application interacts with. You use the endpoints generated by the development tool to test the REST APIs to access the business objects.

For example, you can use any of the following techniques to test the REST APIs:

- In a 3rd party tool that you display in a browser
- In the cURL command line tool from a command window
- In a web browser (typically limited to GET requests)

Testing the REST APIs requires knowledge of the REST API URI syntax. For details about URI syntax, consult the Oracle documentation for your visual development tool.

2
Working with the Resource Catalog

You can retrieve a description of resources, including the resource collection attributes and available actions, using by a specific media type and describe action.

Topics

About the REST API Catalog Describe

REST APIs support retrieving a describe for all the available resources in the resource catalog, it returns JSON objects that contain the attributes, actions, and links defined in the REST resource definitions for the business object.

The describe for the REST API resource catalog allows you to identify the shape and actions allowed on a REST API defined for the service endpoint. By default the catalog describe request returns a JSON object that contains the information needed to understand all available resources.

REST APIs support the following catalog describe use cases:

- Retrieve a *resource catalog describe*, where the describe details will be limited to resource titles and links only and children, or nested resources, will be excluded.
- Retrieve a resource catalog describe but optionally exclude or include children resources nested within a parent resource and optionally exclude or include all resource annotations.

To retrieve the catalog describe of all the available, parent resources in the application, you append `/describe` to the base URL with the query parameter `metadataMode` set to `minimal`:

```
http://<base_url>/describe?metadataMode=minimal
```

Additionally, you can append URL query parameters on the request for a minimal catalog describe to retrieve specific details in the describe. For example, the following URL with appended query parameters retrieves a minimal catalog describe with all available children resources nested within their parent resources included.

```
http://<base_url>/describe?metadataMode=minimal&includeChildren=true
```

The following table identifies the URL query parameters the may be used with the catalog describe request. These query parameters let you control the amount of detail retrieved in the describe.

Table 2-1 Optional URL Query Parameters for Catalog Describe Requests

URL Query Parameter	Values	Description
`includeChildren`	`true`, `false` (default)	Use to include all available children resources nested within a parent resource describe. You can append `&includeChildren=true` on the describe request. For a resource catalog describe example, see Retrieving the Resource Catalog Describe.
`showAnnotations`	`true`, `false` (default)	To include resource annotations in the catalog describe, you can append `&showAnnotations=true` on the describe request. Note that annotations must be defined by the web developer and may not be present on the resource.

Retrieving the Resource Catalog Describe

REST APIs support describing all available resources while retrieving a reduced amount of information for the application end point using a GET method. The reduced or minimal catalog describe helps improve the readability of the describe by limiting the resource information to resource titles, links, and available annotations.

To examine the minimal describe for all available resources in the resource catalog:

1. Execute the minimal resource catalog describe and locate the names of the resources in the describe. Note that nested resources or children resources are not shown by default.
2. Examine these resource objects `links`.

For example, the minimal describe for a service with a `Department` resource returns the following objects:

```
{
  "Resources" : {
    "Department" : {
      Department,
      "links" : [ {
        "rel" : "self",
        "href" : "<base_url>/Department/describe",
        "name" : "self",
        "kind" : "describe"
      } ]
    },
      ...
  }
}
```

By default children resources are not included in the minimal describe. Use the `includeChildren` query parameter to retrieve the minimal catalog describe with all available children resources nested within the parent resources. For example, to view children resources in the minimal describe, you can use a request like the following:

```
<base_url>/describe?metadataMode=minimal&includeChildren=true
```

The minimal describe with the `includeChildren` query parameter set to `true` for a `Department` resource that includes a child resource `Employee` returns the following objects:

```
{
  "Resources" : {
    "Department" : {
      Department,
     "children" :  {
          "Employee" :  {
            "links" : [ {
              "rel" : "self",
              "href" : "<base_url>/Department/{id}/child/Employee/describe",
              "name" : "self",
              "kind" : "describe"
            } ]
         }
       },
      "links" : [ {
        "rel" : "self",
        "href" : "<base_url>/Department/describe",
        "name" : "self",
        "kind" : "describe"
      } ]
    },
      ...
  }
}
```

The following sample retrieves a minimal resource catalog describe, including children resources, where the `Employee` resource is nested within the `Department` resource.

Request

- **URL**

  `<base_url>/describe?metadataMode=minimal&includeChildren=true`

- **HTTP Method**

  GET

- **Content-Type**

  none

- **Payload**

  none

Response

- **HTTP Code**

  `200`

- **Content-Type**

  `application/vnd.oracle.adf.description+json`

- **Payload**

  ```
  {
    "Resources" :  {
      "Department" :  {
  ```

```
      "children" :  {
        "Employee" :  {
          "links" : [
          {
            "rel" : "self",
            "href" : "<base_url>/Department/{id}/child/Employee/describe",
            "name" : "self",
            "kind" : "describe"
          } ]
        }
      },
      "links" : [
        {
          "rel" : "self",
          "href" : "<base_url>/Department/describe",
          "name" : "self",
          "kind" : "describe"
        }
      ]
    }
  }
}
```

Retrieving a Resource Describe

REST APIs support retrieving a describe for a single resource, all the available resources, or only the nested resources. The describe returns a JSON object that contains the attributes, actions, and links defined in the REST resource definition.

The describe for the resource allows you to identify the shape and actions allowed on the business object. It returns a JSON object that contains the attributes, actions, and links defined in the REST resource definition.

The REST API framework supports the following describe use cases for the service end point:

- Describe a single resource collection.
- Describe a nested resource in a parent-child relationship.
- Describe two or more named resource collections.
- Describe all available resources (resource catalog). For details, see About the REST API Catalog Describe.

To retrieve the describe, invoke an HTTP GET with `/describe` appended to the resource URL.

For example, the following URL returns the describe for the `Department` resource:

`<base_url>/Department/describe`

Describing a Resource Collection

REST APIs support describing resource collections.

To examine a resource collection:

1. Execute the resource collection describe and locate the names of the resources in the describe.

2. Examine these resource objects to understand the shape of each resource:
   - `attributes` specifies the list of available resource collection attributes.
   - `collection` specifies the shape of the collection and specifies `links` and available `actions` (including media types).
   - `item` specifies the shape of the items of the collection and itself specifies `links` and available `actions`.
   - `children` specifies any nested resources (and itself contains `attributes`, `collection`, and `item` objects).

For example, the describe for the `Department` resource returns the following objects:

```
{
  "Resources" : {
    "Department" : {
      "discrColumnType" : false,
      "attributes" : [ {
       ...
      } ],
      "collection" : {
        ...
        } ],
        "links" : [ {
        } ]
        "actions" : [ {
        } ]
      },
      "item" : {
        "links" : [ {
        } ]
        "actions" : [ {
        } ]
      },
      "children" : {
        "Employee"
          ...
        } ],
      ...
      "links" : [ {
      } ]
  }
}
```

The following sample describes the `Department` resource.

Request

- **URL**

  `<base_url>/Department/describe`

- **HTTP Method**

  GET

- **Content-Type**

  none

- **Payload**

  none

Response

- **HTTP Code**

  200

- **Content-Type**

  application/vnd.oracle.adf.description+json

- **Payload**

```
{
  "Resources" : {
    "Department" : {
      "discrColumnType" : false,
      "attributes" : [ {
        "name" : "DepartmentId",
        "type" : "integer",
        "updatable" : true,
        "mandatory" : true,
        "queryable" : true,
        "precision" : 4
      }, {
        "name" : "DepartmentName",
        "type" : "string",
        "updatable" : true,
        "mandatory" : true,
        "queryable" : true,
        "precision" : 30
      }, {
        "name" : "RelState",
        "type" : "integer",
        "updatable" : true,
        "mandatory" : false,
        "queryable" : true
      } ],
      "collection" : {
        "rangeSize" : 25,
        } ],
        "links" : [ {
          "rel" : "self",
          "href" : "<base_url>/Department",
          "name" : "self",
          "kind" : "collection"
        } ],
        "actions" : [ {
          "name" : "get",
          "method" : "GET",
          "responseType" : [ "application/json", "application/
vnd.oracle.adf.resourcecollection+json" ]
        }, {
          "name" : "create",
          "method" : "POST",
          "requestType" : [ "application/vnd.oracle.adf.resourceitem+json" ],
          "responseType" : [ "application/json", "application/
vnd.oracle.adf.resourceitem+json" ]
        }, {
          "name" : "testImpl",
          "parameters" : [ {
            "name" : "testid",
            "type" : "string",
            "mandatory" : false
```

```
          } ],
          "resultType" : "string",
          "method" : "POST",
          "requestType" : [ "application/vnd.oracle.adf.action+json" ],
          "responseType" : [ "application/json", "application/
vnd.oracle.adf.actionresult+json" ]
        } ]
      },
      "item" : {
        "links" : [ {
          "rel" : "child",
          "href" : "<base_url>/Department/{id}/child/Employee",
          "name" : "Employee",
          "kind" : "collection",
          "cardinality" : {
            "value" : "1 to *",
            "sourceAttributes" : "DepartmentId",
            "destinationAttributes" : "DepartmentId"
          }
        }, {
          "rel" : "self",
          "href" : "<base_url>/Department/{id}",
          "name" : "self",
          "kind" : "item"
        } ],
        "actions" : [ {
          "name" : "get",
          "method" : "GET",
          "responseType" : [ "application/json", "application/
vnd.oracle.adf.resourceitem+json" ]
        }, {
          "name" : "update",
          "method" : "PATCH",
          "requestType" : [ "application/vnd.oracle.adf.resourceitem+json" ],
          "responseType" : [ "application/json", "application/
vnd.oracle.adf.resourceitem+json" ]
        }, {
          "name" : "delete",
          "method" : "DELETE"
        } ]
      },
      "children" : {
        "Employee" : {
          "discrColumnType" : false,
          "attributes" : [ {
            "name" : "EmployeeId",
            "type" : "integer",
            "updatable" : true,
            "mandatory" : true,
            "queryable" : true,
            "precision" : 6
          }, {
            "name" : "FirstName",
            "type" : "string",
            "updatable" : true,
            "mandatory" : false,
            "queryable" : true,
            "precision" : 20
          }, {
            "name" : "LastName",
            "type" : "string",
```

```
      "updatable" : true,
      "mandatory" : true,
      "queryable" : true,
      "precision" : 25
    }, {
      "name" : "Email",
      "type" : "string",
      "updatable" : true,
      "mandatory" : true,
      "queryable" : true,
      "precision" : 25
    }, {
      "name" : "JobId",
      "type" : "string",
      "updatable" : true,
      "mandatory" : true,
      "queryable" : true,
      "precision" : 10,
      "controlType" : "choice",
      "maxLength" : "10"
    }, {
      "name" : "DepartmentId",
      "type" : "integer",
      "updatable" : true,
      "mandatory" : false,
      "queryable" : true,
      "precision" : 4
    }, {
      "name" : "Salary",
      "type" : "number",
      "updatable" : true,
      "mandatory" : false,
      "queryable" : true,
      "precision" : 8,
      "scale" : 2
    } ],
    "collection" : {
      "rangeSize" : 0,
      }, {
        "name" : "PrimaryKey",
        "attributes" : [ {
          "name" : "EmployeeId",
          "type" : "integer",
          "updatable" : true,
          "mandatory" : true,
          "queryable" : true,
          "precision" : 6
        } ]
      } ],
      "links" : [ {
        "rel" : "self",
        "href" : "<base_url>/Department/{id}/child/Employee",
        "name" : "self",
        "kind" : "collection"
      } ],
      "actions" : [ {
        "name" : "get",
        "method" : "GET",
        "responseType" : [ "application/json", "application/
vnd.oracle.adf.resourcecollection+json" ]
      }, {
```

```
              "name" : "create",
              "method" : "POST",
              "requestType" : [ "application/vnd.oracle.adf.resourceitem+json" ],
              "responseType" : [ "application/json", "application/
vnd.oracle.adf.resourceitem+json" ]
            } ]
          },
          "item" : {
            "links" : [ {
              "rel" : "self",
              "href" : "<base_url>/Department/{id}/child/Employee/{id}",
              "name" : "self",
              "kind" : "item"
            }, {
              "rel" : "parent",
              "href" : "<base_url>/Department/{id}",
              "name" : "parent",
              "kind" : "item"
            } ],
            "actions" : [ {
              "name" : "get",
              "method" : "GET",
              "responseType" : [ "application/json", "application/
vnd.oracle.adf.resourceitem+json" ]
            }, {
              "name" : "update",
              "method" : "PATCH",
              "requestType" : [ "application/vnd.oracle.adf.resourceitem+json" ],
              "responseType" : [ "application/json", "application/
vnd.oracle.adf.resourceitem+json" ]
            }, {
              "name" : "delete",
              "method" : "DELETE"
            }, {
              "name" : "multiplySalary",
              "parameters" : [ {
                "name" : "multiplicand",
                "type" : "number",
                "mandatory" : false
              } ],
              "resultType" : "number",
              "method" : "POST",
              "requestType" : [ "application/vnd.oracle.adf.action+json" ],
              "responseType" : [ "application/json", "application/
vnd.oracle.adf.actionresult+json" ]
            } ]
          },
          "links" : [ {
            "rel" : "self",
            "href" : "<base_url>/Department/{id}/child/Employee/describe",
            "name" : "self",
            "kind" : "describe"
          } ]
        }
      },
      "links" : [ {
        "rel" : "self",
        "href" : "<base_url>/Department/describe",
        "name" : "self",
        "kind" : "describe"
      } ]
```

```
      }
    }
  }
```

Describing a Nested Resource

REST APIs support describing a nested resource that results from related business objects.

To examine nested resources in the resource catalog:

1. Execute the nested resource describe and locate the names of the resources in the describe. The `children` attribute identifies nested resources.
2. Examine these resource objects to understand the shape of each resource:
   - `attributes` specifies the list of available resource collection attributes.
   - `collection` specifies the shape of the collection and specifies `links` and available `actions` (including media types).
   - `item` specifies the shape of the items of the collection and itself specifies `links` and available `actions`.
   - `children` specifies the nested resources (and itself contains `attributes`, `collection`, and `item` objects).

For example, the describe for the nested resources `Department` and `Employee` returns the following objects:

```
{
  "Resources" : {
    "Employee" : {
      "discrColumnType" : false,
      "attributes" : [ {
       ...
      } ],
      "collection" : {
        ...
        } ],
        "links" : [ {
        } ]
        "actions" : [ {
        } ]
      },
      "item" : {
        "links" : [ {
        } ]
        "actions" : [ {
        } ]
      },
      "children" : {
        "Department"
          ...
        } ],
      ...
      "links" : [ {
      } ]
  }
}
```

The following sample (URL1) describes the `Employee` resource which can be found in the context of a `Department` resource item.

Note: To recursively include all children of the resource item on the requested describe, provide the query parameter `?includeChildren=true` on the describe URL.

Requests

- **URL 1**

  `<base_url>/Department/10/child/Employee/describe`

- **HTTP Method**

  GET

- **Content-Type**

  none

- **Payload**

  none

Responses

- **HTTP Code**

  `200`

- **Content-Type**

  `application/vnd.oracle.adf.description+json`

- **Payload 1**

```
{
  "Resources" : {
    "Employee" : {
      "discrColumnType" : false,
      "attributes" : [ {
        "name" : "EmployeeId",
        "type" : "integer",
        "updatable" : true,
        "mandatory" : true,
        "queryable" : true,
        "precision" : 6
      }, {
        "name" : "FirstName",
        "type" : "string",
        "updatable" : true,
        "mandatory" : false,
        "queryable" : true,
        "precision" : 20
      }, {
        "name" : "LastName",
        "type" : "string",
        "updatable" : true,
        "mandatory" : true,
        "queryable" : true,
        "precision" : 25
      }, {
        "name" : "Email",
        "type" : "string",
        "updatable" : true,
        "mandatory" : true,
```

```
        "queryable" : true,
        "precision" : 25
      }, {
        "name" : "JobId",
        "type" : "string",
        "updatable" : true,
        "mandatory" : true,
        "queryable" : true,
        "precision" : 10,
        "controlType" : "choice",
        "maxLength" : "10"
      }, {
        "name" : "DepartmentId",
        "type" : "integer",
        "updatable" : true,
        "mandatory" : false,
        "queryable" : true,
        "precision" : 4
      }, {
        "name" : "Salary",
        "type" : "number",
        "updatable" : true,
        "mandatory" : false,
        "queryable" : true,
        "precision" : 8,
        "scale" : 2
      } ],
      "collection" : {
        "rangeSize" : 25,
        }, {
          "name" : "PrimaryKey",
          "attributes" : [ {
            "name" : "EmployeeId",
            "type" : "integer",
            "updatable" : true,
            "mandatory" : true,
            "queryable" : true,
            "precision" : 6
          } ]
        } ],
        "links" : [ {
          "rel" : "self",
          "href" : "<base_url>/Department/10/child/Employee",
          "name" : "self",
          "kind" : "collection"
        } ],
        "actions" : [ {
          "name" : "get",
          "method" : "GET",
          "responseType" : [ "application/json", "application/
vnd.oracle.adf.resourcecollection+json" ]
        }, {
          "name" : "create",
          "method" : "POST",
          "requestType" : [ "application/vnd.oracle.adf.resourceitem+json" ],
          "responseType" : [ "application/json", "application/
vnd.oracle.adf.resourceitem+json" ]
        } ]
      },
      "item" : {
        "links" : [ {
```

```
          "rel" : "self",
          "href" : "<base_url>/Department/10/child/Employee/{id}",
          "name" : "self",
          "kind" : "item"
        }, {
          "rel" : "parent",
          "href" : "<base_url>/Department/10",
          "name" : "parent",
          "kind" : "item"
        } ],
        "actions" : [ {
          "name" : "get",
          "method" : "GET",
          "responseType" : [ "application/json", "application/
vnd.oracle.adf.resourceitem+json" ]
        }, {
          "name" : "update",
          "method" : "PATCH",
          "requestType" : [ "application/vnd.oracle.adf.resourceitem+json" ],
          "responseType" : [ "application/json", "application/
vnd.oracle.adf.resourceitem+json" ]
        }, {
          "name" : "delete",
          "method" : "DELETE"
        }, {
          "name" : "multiplySalary",
          "parameters" : [ {
            "name" : "multiplicand",
            "type" : "number",
            "mandatory" : false
          } ],
          "resultType" : "number",
          "method" : "POST",
          "requestType" : [ "application/vnd.oracle.adf.action+json" ],
          "responseType" : [ "application/json", "application/
vnd.oracle.adf.actionresult+json" ]
        } ]
      },
      "links" : [ {
        "rel" : "self",
        "href" : "<base_url>/Department/10/child/Employee/describe",
        "name" : "self",
        "kind" : "describe"
      } ]
    }
  }
}
```

3

Working with REST API Framework Versions

You can specify a REST API framework version to make REST API calls when you want to opt into the features that the framework version defines.

Topics

About REST API Framework Versions

You may utilize a new feature or enhancement that is introduced in a version of the REST API framework if your application opts into the framework version within the visual development tool. A framework version refers to a specific version of the REST API framework that calls to REST APIs will use. Depending on the version, functionality for accessing business objects will vary.

It is important that you know the framework version enabled in your web application since you may need to pass a different payload format to utilize a new feature or enhancement. The framework version used to make calls is under your control so that you may opt into the functionality when you are ready.

These are the way the tooling allows you to specify the framework version to make calls:

- Specify the default framework version to be used.
- Specify the framework version using a custom request header when making the call.

You may pass the custom header `REST-Framework-Version` on the REST API request to specify the framework version to use to execute the request. The REST API framework version passed in the version header overrides the default framework declaration defined by the web application developer.

When your REST API call passes no version header, the calls use the default that you defined in the application. When you did not define the default framework version and do not pass a version header, then the version of the REST API framework is determined by the default setting enabled by the tooling.

For the root resource `/context`, the default REST API framework version for the latest release will be used.

You may want to find out the default framework version for a particular release. To support this use case, REST APIs will return the default framework version in the resource version describe, as the following sample shows. Note that you may override the default framework version with another framework version identifier by specifying

the value in the `REST-Framework-Version` header. The `allowedFrameworkVersions` property lists the values of the available framework versions.

```
{
    "items" : [
        {
            "version" : "18.2",
            "isLatest" : true,
            "adf:extension" : {
                    "defaultFrameworkVersion" : "2",
                    "allowedFrameworkVersions" : [ "1","2","3","4","5","6","7" ]
            },
            "links" : [
             ...
```

For details about the REST API framework functionality supported in each framework version, see Understanding REST API Framework Version Support.

Understanding REST API Framework Version Support

You can specify a REST API framework version for your web application to opt into new functionality offered by a later version of the REST API framework. Currently, Oracle offers the following framework versions.

> **Note:**
>
> Each REST API framework version after version 1 introduces functionality that the previous framework versions does not support. Thus, when you choose to opt into a later framework version, the REST API may introduce backward incompatible changes on the web application consuming the REST API. This topic explains the changes for each framework version.

Framework Version 1

Version 1 is the default version that the REST APIs will use to process requests for web applications when no other version is specified.

Note that the query-by-example resource query syntax supported in the base framework version (version 1) is not compatible with later versions of the REST API framework. Beginning with version 2 of the REST API framework, a more advanced query syntax is offered instead.

Framework Version 2

The purpose of this new version is to introduce an expanded query expression syntax for making REST API calls. Version 2 of the REST API framework will interpret the `q` query parameter value differently than the way framework version 1 does, and therefore introduces a backward incompatible change to web applications that rely on framework version 1. Only when framework version 2 (or later) is specified for the request will the REST API support the use of the expanded expression syntax to process the request.

In version 1, filtering resource collections using the `q` query parameter is limited to a query-by-example syntax, as follows.

```
GET /rest/19.0/Departments?q=Dname SA*;Loc BOSTON
```

Whereas, starting in version 2, the new advanced query syntax supports filtering resource collections using rowmatch query expressions, as follows.

```
GET /rest/19.1/Departments?q=Dname like 'SA*' or Loc = 'BOSTON'
```

> **Note:**
>
> To support both rowmatch expressions (offered in framework version 2 and later) and the query-by-example syntax (version 1 only), you would need to associate framework version 2 (or later) with a new release version identifier that you define in the web application. For example, in the above URL samples, you can preserve the original functionality for release version `19.0` and expose the new functionality for version `19.1`. This assumes that web application has declared framework version 2 the default on release version `19.1`. Alternatively, if your web application will no longer require the functionality of your current framework version, you may associate the new framework version with your existing release version identifier. Therefore, you are not required to increment the release version to make use of a new framework version.

For an explanation of the enhanced query syntax offered by rowmatch expressions, see Understanding Framework Support for Query Syntax.

Framework Version 3

The purpose of this version is to add support for retrieving nested child resources with payload attributes that may be used by the web application to determine whether more resource items would be returned in a subsequent REST API request. To support this functionality, the payload structure in framework version 3 now represents nested child resource as a resource collection, instead of an array of items, as was true in version 1 and 2. Therefore, version 3 introduces a backward incompatible change to web applications that rely on framework version 1 or version 2. If you decide to opt into version 3, you will expose functionality that allows GET operations to use the `?expand` and `?fields` query parameter to return a nested child resource as a resource collection with the `hasMore` attribute. In affect, this change supports pagination of nested child resource that would otherwise require more than one request to fetch.

When you want to add support for framework version 3 to your application, the same guidelines described for framework version 2 (see above section) apply for preserving the existing level of functionality in the web application.

For an example of the new payload structure for nested child resources introduced in version 3, see Fetching a Child Business Object and Fetching a Business Object with a Subset of Items. For details about paginating a resource collection using the `hasMore` attribute, see Paging a Business Object.

Framework Version 4

In addition to HTTP status codes and error messages, it is possible to obtain exception details in the response when your request is enabled to use REST API framework version 4 and the request is made for either `application/vnd.oracle.adf.error+json` or `application/json` media types. With framework version 4, the response will

be in the form an exception detail payload which provides the following benefits to the web application:

- If multiple errors occur in a single request, the details of each error are presented in a hierarchical structure.
- An application-specific error code may be present that identifies the exception corresponding to each error.
- An error path may be present that identifies the location of each error in the request payload structure.

> **Note:**
>
> The exception detail may or may not present certain details, such as the application-specific error code and the request payload's error path.

For example, compare the error response for a POST submitted with a payload that contains the following incorrectly formatted date field when framework version 3 (or earlier) is enable and when framework version 4 (or later) is enabled.

```
{    "EmpNum" : 5027,
     "EmpName" : "John",
     "EmpHireDate" : "not a date"
}
```

Standard Error Response

Without framework version 4, no response payload is generated and instead only a single error message that does not reference the request payload will be returned in the response.

```
"An instance of type oracle.jbo.domain.Date cannot be created from string
not a date. The string value must be in format YYYY-MM-DDTHH:MI:SS.sss
+hh:mm."
```

Exception Payload Error Response

With framework version 4 (or later) enabled, the following exception detail payload is generated for the response. The payload includes the usual HTTP status code and formats the details of one or more exceptions in an array structure.

```
{    "title" : "Bad Request",
    "status" : "400",
    "o:errorDetails" : [ {
     "detail" : "An instance of type oracle.jbo.domain.Date cannot be
created from string not a date.
               The string value must be in format YYYY-MM-DDTHH:MI:SS.sss
+hh:mm.",
       "o:errorCode" : "26099",
       "o:errorPath" : "/EmpHireDate"
    } ]
 }
```

Framework Version 5

Framework version 5 is not supported for users of visual development tools provided by Oracle Cloud services.

Framework Version 6

The purpose of this framework version is to easily differentiate between the resource fields and item information like links and headers. A new element `@context` is introduced in this version and all the information for an item is moved under `@context`. The `changeIndicator` value is moved to `ETag`, which is under `headers`. A new context information key is included under `@context` that contains the unique identifier of the specific resource item as a string.

The new payload for a resource item in a response payload and collection response payload will be similar to the one below:

```
{
     "field1": "value1",
     "field2": "value2",
      ...
     "@context" : {
          "key" : "AB8765BCD",
          "headers" : {
              "ETag" : "ACED00057372001..."
          },
          "links": [
           {
             "rel": "self",
             "href": "https://<baseurl>/accounts/CDRM_53640",
             "name": "accounts",
             "kind": "item"
           },
           { // other links }}
         ]
      }
}
```

Using the Request Header to Specify the REST API Framework Version

REST APIs support executing individual requests on the service endpoint using a custom header to affect the processing of the payload with the functionality specific to a particular REST API framework version. The framework version specified by the custom header overrides the default framework version declaration that may exist in the client application.

> **Note:**
>
> A framework version refers to a specific version of the REST API framework. For details about the REST API framework functionality supported in each framework version, see Understanding REST API Framework Version Support.

To process a request using a specific REST API framework version, the request must pass the custom header `REST-Framework-Version` with the framework version number specified. For example, the following header specifies framework version 2 will be used to process the request that passes this version header.

```
REST-Framework-Version: 2
```

If the custom header is omitted on the request, then REST APIs will use the application's default framework version, as defined in the web application. When the application does not define a default framework version and the request omits the version header, then the base version (version 1) of the REST API framework is assumed.

Part II
Tasks

To use the REST API you should be familiar with the use cases that the REST API supports.

Topics

4

CRUD Tasks

You can access business object using standard HTTP request methods, including GET, POST, PATCH, and DELETE.

Topics

Retrieving Business Objects

REST APIs support the GET method on REST resources to retrieve a resource or nested resource, page a resource, filter a resource collection with a query, or sort a resource collection.

REST APIs support the following GET method use cases:

- Fetching a resource collection payload.
- Fetching a resource collection payload with a subset of resource items.
- Fetching a resource item payload.
- Fetching a paged resource collection.
- Fetching a sorted resource collection.
- Fetching a nested child resource item payload.
- Fetching data only in a resource item payload.
- Filtering a resource item payload with query parameters.

Fetching a Business Object

REST APIs support fetching a resource collection without filtering items.

The following sample fetches the `Department` resource collection and all five items of the collection.

Request

- **URL**

  `<base_url>/Department`

- **HTTP Method**

  GET

- **Content-Type**

  none

- **Payload**

  none

Response

- **HTTP Code**

  200

- **Content-Type**

  application/vnd.oracle.adf.resourcecollection+json

- **Payload**

```
{
  "items" : [ {
    "DepartmentId" : 10,
    "DepartmentName" : "Administration",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/10",
      "name" : "Department",
      "kind" : "item"
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/10/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  }, {
    "DepartmentId" : 20,
    "DepartmentName" : "Marketing",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/20",
      "name" : "Department",
      "kind" : "item"
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/20/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  }, {
    "DepartmentId" : 30,
    "DepartmentName" : "Purchasing",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/30",
      "name" : "Department",
      "kind" : "item"
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/30/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  }, {
```

```
    "DepartmentId" : 40,
    "DepartmentName" : "Human Resources",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/40",
      "name" : "Department",
      "kind" : "item"
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/40/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  }, {
    "DepartmentId" : 50,
    "DepartmentName" : "Shipping",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/50",
      "name" : "Department",
      "kind" : "item"
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/50/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  } ],
  "count" : 5,
  "hasMore" : false,
  "limit" : 5,
  "offset" : 0,
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Department",
    "name" : "Department",
    "kind" : "collection"
  } ]
}
```

Fetching a Business Object with a Subset of Items

REST APIs support fetching a resource collection with a subset of items.

The payload structure of nested child resource differs depending on the REST API framework version that has been registered for the request. For details about the REST API framework versions, see Understanding REST API Framework Version Support.

The following samples are based on different versions of the REST API framework, which reflect the response payload structure change supported in REST API framework version 3 (and later). In both framework scenarios, the samples fetch the `Employee` resource as a child of the `Department` resource.

REST API Framework Version 3 (and later)

Starting with version 3 of the REST API framework, the REST API returns a nested child resource in the response payload as a resource collection, instead of as an array of resource items. This functionality, available in framework version 3 (and later), allows web applications to make a request for additional records after determining how

many items were left unfetched in the initial request. The attributes `hasMore` and `count` on the child resource indicate whether more items may be returned from the resource collection. For details about using the pagination attributes from the response payload when you opt into REST API framework version 3, see Paging a Business Object.

The following sample illustrates functionality for REST API framework version 3 (and later). The response payload represents the nested child resource as a resource collection, where the collection object includes the `hasMore` and `count` attributes. A link is provided should it be necessary to query the child resource for additional resource items. In this sample, the response payload shows the `hasMore` attribute is `false`, suggesting that no items remain unfetched on the `Employee` child resource for either department 10 or department 20.

Request Made With Framework Version 3

- **URL**

  ```
  <base_url>/Department?
  fields=DepartmentId;Employee:FirstName&onlyData=true
  ```

- **HTTP Method**

  GET

- **Query Parameters**

  `fields`

  This parameter filters resource attributes so that only the specified attributes are returned. The parameter value is a comma-separated list of attribute names when filtering a single resource collection. Example: `?fields=FirstName,LastName`. When filtering multiple resources, the resource names are followed by a colon and the comma-separated attribute names list with a semi-colon separating each resource filter list. Example: `?fields=Employee:FirstName;Employee.JobHistory:JobId`. Note that dot notation allows access to nested resources. This parameter cannot be combined with `expand` query parameter. If both are provided, only `fields` will be considered.

  `onlyData`

  This parameter filters the resource item payload to contain only data (no links section, for example).

- **Content-Type**

  none

- **Payload**

  none

Response Made With Framework Version 3

- **HTTP Code**

  `200`

- **Content-Type**

  `application/vnd.oracle.adf.resourcecollection+json`

- **Payload**

  ```
  {
      "items" : [ {
  ```

```
    "DepartmentId" : 10,
    "Employee" : {
         "items" : [ {
            {
                "FirstName" : "Jennifer"
            }
      }  ],
      "count" : 1,
      "hasMore" : false,
      "limit" : 25,
      "offset" : 0,
      "links" : [ {
          "rel" : "self",
          "href" : "<base_url>/Department/10/child/Employee",
          "name" : "Employee",
          "kind" : "collection"
    }, ]
 }, {
    "DepartmentId" : 20,
        "Employee" : {
            "items" : [ {
               {
                   "FirstName" : "Michael"
               },
               {
                   "FirstName" : "Pat"
               }
      }  ],
      "count" : 2,
      "hasMore" : false,
      "limit" : 25,
      "offset" : 0,
      "links" : [ {
          "rel" : "self",
          "href" : "<base_url>/Department/20/child/Employee",
          "name" : "Employee",
          "kind" : "collection"
    }, ]
 }, {
  ...
],
"count" : 25,
"hasMore" : true,
"limit" : 25,
"offset" : 0,
"links" : [
    {
        "rel" : "self",
        "href" : "<base_url>/Department",
        "name" : "Department",
        "kind" : "collection"
    }
]
```

REST API Framework Version 1 or Version 2

Version 1 and version 2 of the REST API framework return the nested child resource expanded in the response payload as an array of resource items. If the resource collection being fetched is large, several requests will be required to fetch all items.

The following samples fetch the attribute values of the `Department` and `Employee` collections. The query parameter `fields` ensures the response payload contains only the specified attributes. Note that a GET request may return no values for any resource in the URL that does not specify an attribute value.

The first request (URL 1) fetches the items for employee `101`. The query parameter `fields` ensures the response payload contains only the specified attributes: `FirstName`, `LastName`, and `Email`.

The second request (URL 2) fetches the department `DepartmentId` and the `FirstName` item for employees of each department. The query parameter `onlyData` filters the response to hide child links.

The third request (URL 3) fetches the department `DepartmentId`, the `FirstName` item for employees of each department, and the `JobId` history for each employee. The query parameter `onlyData` again filters the response to hide child links.

Request 1 Made With Framework Version 1 or 2

- **URL 1**

  `<base_url>/Employee/101?fields=FirstName,LastName,Email`

- **HTTP Method**

  GET

- **Query Parameters**

  `fields`

  This parameter filters resource attributes so that only the specified attributes are returned. The parameter value is a comma-separated list of attribute names when filtering a single resource collection. Example: `?fields=FirstName,LastName`. When filtering multiple resources, the resource names are followed by a colon and the comma-separated attribute names list with a semi-colon separating each resource filter list. Example: `?fields=Employee:FirstName;Employee.JobHistory:JobId`. Note that dot notation allows access to nested resources. This parameter cannot be combined with `expand` query parameter. If both are provided, only `fields` will be considered.

- **Content-Type**

  none

- **Payload**

  none

Response 1 From Framework Version 1 or 2

- **HTTP Code**

  `200`

- **Content-Type**

  `application/vnd.oracle.adf.resourceitem+json`

- **Payload 1**

```
{
  "FirstName" : "Neena",
  "LastName" : "Smith",
  "Email" : "NSMITH",
```

```
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Employee/101",
    "name" : "Employee",
    "kind" : "item"
  } ]
}
```

Request 2 Made With Framework Version 1 or 2

- **URL 2**

  `<base_url>/Department?fields=DepartmentId;Employee:FirstName&onlyData=true`

- **HTTP Method**

  GET

- **Query Parameters**

  `fields`

  This parameter filters resource attributes so that only the specified attributes are returned. The parameter value is a comma-separated list of attribute names when filtering a single resource collection. Example: `?fields=FirstName,LastName`. When filtering multiple resources, the resource names are followed by a colon and the comma-separated attribute names list with a semi-colon separating each resource filter list. Example: `?fields=Employee:FirstName;Employee.JobHistory:JobId`. Note that dot notation allows access to nested resources. This parameter cannot be combined with `expand` query parameter. If both are provided, only `fields` will be considered.

  `onlyData`

  This parameter filters the resource item payload to contain only data (no links section, for example).

- **Content-Type**

  none

- **Payload**

  none

Response 2 From Framework Version 1 or 2

- **HTTP Code**

  `200`

- **Content-Type**

  `application/vnd.oracle.adf.resourcecollection+json`

- **Payload 2**

```
{
    "items" : [

        {
            "DepartmentId" : 10,
            "Employee" : [
                {
                    "FirstName" : "Jennifer"
```

```
                }
            ]
        },
        {
            "DepartmentId" : 20,
            "Employee" : [
                {
                    "FirstName" : "Michael"
                },
                {
                    "FirstName" : "Pat"
                }
            ]
        },
        {
            "DepartmentId" : 30,
            "Employee" : [
                {
                    "FirstName" : "Den"
                },
                {
                    "FirstName" : "Alexander"
                },
                {
                    "FirstName" : "Shelli"
                },
                {
                    "FirstName" : "Sigal"
                },
                {
                    "FirstName" : "Guy"
                },
                {
                    "FirstName" : "Karen"
                }
            ]
        },
        {
            "DepartmentId" : 40,
            "Employee" : [
                {
                    "FirstName" : "Susan"
                }
            ]
        },
      ...
    ],
    "count" : 25,
    "hasMore" : true,
    "limit" : 25,
    "offset" : 0,
    "links" : [
        {
            "rel" : "self",
            "href" : "<base_url>/Department",
            "name" : "Department",
            "kind" : "collection"
        }
    ]
```


Request 3 Made With Framework Version 1 or 2

- **URL 3**

  ```
  <base_url>/Department?
  fields=DepartmentId;Employee:FirstName;Employee.JobHistory:JobId&onlyD
  ata=true
  ```

- **HTTP Method**

  GET

- **Query Parameters**

  `fields`

  This parameter filters resource attributes so that only the specified attributes are returned. The parameter value is a comma-separated list of attribute names when filtering a single resource collection. Example: `?fields=FirstName,LastName`. When filtering multiple resources, the resource names are followed by a colon and the comma-separated attribute names list with a semi-colon separating each resource filter list. Example: `?fields=Employee:FirstName;Employee.JobHistory:JobId`. Note that dot notation allows access to nested resources. This parameter cannot be combined with `expand` query parameter. If both are provided, only `fields` will be considered.

  `onlyData`

  This parameter filters the resource item payload to contain only data (no links section, for example).

- **Content-Type**

  none

- **Payload**

  none

Response 3 From Framework Version 1 or 2

- **HTTP Code**

  `200`

- **Content-Type**

  `application/vnd.oracle.adf.resourcecollection+json`

- **Payload 3**

```
{
    "items" : [

        {
            "DepartmentId" : 10,
            "Employee\" : [
                {
                    "FirstName" : "Jennifer",
                    "JobHistory" : [
                        {
                            "JobId" : "AD_ASST"
                        },
                        {
                            "JobId" : "AC_ACCOUNT"
                        }
                    ]
                }
```

```
    ]
},
{
    "DepartmentId" : 20,
    "Employee" : [
        {
            "FirstName" : "Michael",
            "JobHistory" : [
                {
                    "JobId" : "MK_REP"
                }
            ]
        },
        {
            "FirstName" : "Pat",
            "JobHistory" : [
                {
                    "JobId" : "AD_ASST"
                },
                {
                    "JobId" : "AC_ACCOUNT"
                }
            ]
        }
    ]
},
{
    "DepartmentId" : 30,
    "Employee" : [
        {
            "FirstName" : "Den",
            "JobHistory" : [
                {
                    "JobId" : "ST_CLERK"
                }
            ]
        },
        {
            "FirstName" : "Alexander",
            "JobHistory" : [
                {
                    "JobId" : "AD_ASST"
                },
                {
                    "JobId" : "AC_ACCOUNT"
                }
            ]
        },
        {
            "FirstName" : "Shelli",
            "JobHistory" : [
                {
                    "JobId" : "AD_ASST"
                },
                {
                    "JobId" : "AC_ACCOUNT"
                }
            ]

    ]
  ...
```

```
    ],
    "count" : 25,
    "hasMore" : false,
    "limit" : 25,
    "offset" : 0,
    "links" : [
        {
            "rel" : "self",
            "href" : "<base_url>/Department",
            "name" : "Department",
            "kind" : "collection"
        }
    ]
}
```

Fetching a Business Object Item

REST APIs support fetching a resource item.

The following sample fetches `Department` resource item `50`. The response includes a link to the nested child `Employee` resource.

Request

- **URL**

  `<base_url>/Department/50`

- **HTTP Method**

  GET

- **Content-Type**

  none

- **Payload**

  none

Response

- **HTTP Code**

  `200`

- **Content-Type**

  `application/vnd.oracle.adf.resourceitem+json`

- **Payload**

```
{
  "DepartmentId" : 50,
  "DepartmentName" : "Shipping",
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Department/50",
    "name" : "Department",
    "kind" : "item"
  }, {
    "rel" : "child",
    "href" : "<base_url>/Department/50/child/Employee",
    "name" : "Employee",
    "kind" : "collection"
```

```
  } ]
}
```

Paging a Business Object

REST APIs support retrieving resource collections with pagination to display the resource items in sets. Paging is performed using the following URI query parameters:

- `limit` restricts the number of resources returned inside the resource collection. If the limit exceeds the resource count, then the framework will return all available resources. The value is the maximum number of resources to be returned.
- `offset` defines a zero-based index into the collection (where 0 is the first position). The index identifies the starting position of the resource collection. If offset exceeds the resource count, then no resources are returned.

In the following sample, where a `Department` resource collection has five items, the first request (URL1) retrieves two items (at index 0 and 1), and because `offset` is omitted, the starting position of the response is the first item. To display another set of resource items, a second request (URL2) may be made with an `offset` of `2` to correspond to the third item and a `limit` of `2` to retrieve only two more items (at index 2 and 3), and the last request (URL3) with an `offset` of `4` returns the last item of the five item resource collection (at index 4).

Each time a new set of resource items is retrieved, the `hasMore` attribute of the response indicates whether more items may be returned from the collection. In this example, because the collection contains only five items, the response for URL3 shows `hasMore` set to `false`, indicating that the last set of items had been retrieved.

Note that when the `limit` parameter is omitted from the paging URL, the REST API assumes a `limit` of `25` (as determined by the default `RangeSize` value on the business object definition). In this case, up to twenty-five items will be returned with each request. For this reason, it is a best practice when paging through a collection to always include the `limit` query parameter to ensure only the desired number of resource items are returned and not more.

Requests

- **URL 1**

  `<base_url>/Department?limit=2`

- **URL 2**

  `<base_url>/Department?offset=2&limit=2`

- **URL 3**

  `<base_url>/Department?offset=4&limit=2`

- **HTTP Method**

  GET

- **Query Parameters**

  `limit`

  This parameter is an integer value that restricts the number of resource returned inside the resource collection. If the limit exceeds the resource total results, then the available resources will be returned.

`offset`

This parameter is an integer value that defines the starting position of the resource collection. The default (`0`) specifies the first position. If the offset exceeds the resource count, then no resources are returned.

- **Content-Type**

  none

- **Payload**

  none

Responses

- **HTTP Code**

  `200`

- **Content-Type**

  `application/vnd.oracle.adf.resourcecollection+json`

- **Payload 1**

```
{
  "items" : [ {
    "DepartmentId" : 10,
    "DepartmentName" : "Administration",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/10",
      "name" : "Department",
      "kind" : "item"
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/10/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  }, {
    "DepartmentId" : 20,
    "DepartmentName" : "Marketing",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/20",
      "name" : "Department",
      "kind" : "item"
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/20/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  } ],
  "count" : 2,
  "hasMore" : true,
  "limit" : 2,
  "offset" : 0,
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Department",
    "name" : "Department",
    "kind" : "collection"
```

```
  } ]
}
```

- **Payload 2**

```
{
  "items" : [ {
    "DepartmentId" : 30,
    "DepartmentName" : "Purchasing",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/30",
      "name" : "Department",
      "kind" : "item"
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/30/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  }, {
    "DepartmentId" : 40,
    "DepartmentName" : "Human Resources",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/40",
      "name" : "Department",
      "kind" : "item"
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/40/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  } ],
  "count" : 2,
  "hasMore" : true,
  "limit" : 2,
  "offset" : 2,
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Department",
    "name" : "Department",
    "kind" : "collection"
  } ]
}
```

- **Payload 3**

```
{
  "items" : [ {
    "DepartmentId" : 50,
    "DepartmentName" : "Shipping",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/50",
      "name" : "Department",
      "kind" : "item"
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/50/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
```

```
    } ]
  } ],
  "count" : 1
  "hasMore" : false,
  "limit" : 2
  "offset" : 4,
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Department",
    "name" : "Department,
    "kind" : "collection"
  } ]
}
```

Sorting a Business Object

REST APIs support sorting the fetched resource items.

Sorting is performed using the `orderBy` query string parameter in combination with one or more attribute names. The following optional sort order flags may be associated with each attribute:

- `asc` sorts in ascending order. (Default)
- `desc` sorts in descending order.

The `orderBy` query string parameter format is:

`<orderBy_attribute1_name>:<(asc/desc)>, <orderBy_attribute2_name>:<(asc/desc)>`

Example: `attribute1:desc,attribute2`

The following sample (URL1) fetches the `Department` collection sorted by the `DepartmentName` attribute. The second sample (URL2) fetches the child `Employee` collection sorted by the `salary` attribute. Since the sort order flag is not specified for either request sample, the response is ascending order.

Request

- **URL 1**

  `<base_url>/Department?orderBy=DepartmentName`

- **URL 2**

  `<base_url>/Department/50/child/Employee?orderBy=Salary`

- **HTTP Method**

  GET

- **Query Parameter**

  `orderBy`

  This parameter orders a resource collection based on the specified attributes. The parameter value is a comma-separated string of attribute names, each optionally followed by a colon and `asc` or `desc`. Specify `asc` for ascending and `desc` for descending. The default value is `asc`. For example, ? `orderBy=attr1:asc,attr2:desc`.

- **Content-Type**

none

- **Payload**

  none

Response

- **HTTP Code**

  200

- **Content-Type**

  application/vnd.oracle.adf.resourceitem+json

- **Payload 1**


```
{
  "items" : [ {
    "DepartmentId" : 10,
    "DepartmentName" : "Administration",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/10",
      "name" : "Department",
      "kind" : "item"
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/10/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  }, {
    "DepartmentId" : 40,
    "DepartmentName" : "Human Resources",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/40",
      "name" : "Department",
      "kind" : "item"
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/40/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  }, {
    "DepartmentId" : 20,
    "DepartmentName" : "Marketing",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/20",
      "name" : "Department",
      "kind" : "item"
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/20/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  }, {
    "DepartmentId" : 30,
    "DepartmentName" : "Purchasing",
```

```
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/30",
      "name" : "Department",
      "kind" : "item"
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/30/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  }, {
    "DepartmentId" : 50,
    "DepartmentName" : "Shipping",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/50",
      "name" : "Department",
      "kind" : "item"
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/50/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  } ],
  "count" : 5,
  "hasMore" : false,
  "limit" : 25,
  "offset" : 0,
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Department",
    "name" : "Department",
    "kind" : "collection"
  } ]
}
```


- **Payload 2**


```
{
  "items" : [ {
    "EmployeeId" : 132,
    "FirstName" : "TJ",
    "LastName" : "Olson",
    "Email" : "TJOLSON",
    "JobId" : "ST_CLERK",
    "DepartmentId" : 50,
    "Salary" : 2100,
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Employee/132",
      "name" : "Employee",
      "kind" : "item"
    } ]
  }, {
    "EmployeeId" : 136,
    "FirstName" : "Hazel",
    "LastName" : "Philtanker",
    "Email" : "HPHILTAN",
    "JobId" : "ST_CLERK",
    "DepartmentId" : 50,
```

```
    "Salary" : 3100,
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Employee/136",
      "name" : "Employee",
      "kind" : "item"
    } ]
  } ],
  "count" : 2,
  "hasMore" : false,
  "limit" : 25,
  "offset" : 0,
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Employee",
    "name" : "Employee",
    "kind" : "collection"
  } ]
}
```

Fetching a Child Business Object

REST APIs support retrieving a nested child resource collection.

The payload structure of a nested child resource collection differs depending on the REST API framework version that has been registered for the web application. For details about the REST API framework versions, see Working with REST API Framework Versions.

The following samples are based on two different versions of the `Employee` resource. The URL samples showing resource `1.0` reflect a response payload structure supported by REST API framework versions 1 and 2. While the URL samples showing resource `2.0`, reflect the response payload structure supported in REST API framework version 3 (and later). In both framework scenarios, the samples fetch the `Employee` resource nested within the `Department` resource.

REST API Framework Version 3 (and later)

Starting with version 3 of the REST API framework, the REST API returns a nested child resource in the response payload as a resource collection, instead of as an array of resource items. This functionality, available in framework version 3 (and later), allows web applications to make a request for additional records after determining how many items were left unfetched in the initial request. The attributes `hasMore` and `count` on the nested resource indicate whether more items may be returned from the resource collection. For details about using the pagination attributes from the response payload when you opt into REST API framework version 3, see Paging a Business Object.

The following sample illustrates functionality for REST API framework version 3 (and later). The response payload represents the nested child resource as a resource collection, where the collection object includes the `hasMore` and `count` attributes. A link is provided should it be necessary to query the nested resource for additional resource items. In this sample, items of the `Employee` nested resource are fetched with a `count` of `3` in the payload. The response payload shows the `hasMore` attribute is `false`, suggesting that no items remain unfetched.

Request Made With Framework Version 3

- **URL**

```
<base_url>/Department/50?expand=Employee
```

- **HTTP Method**

  GET

- **Query Parameter**

  `expand`

  When this parameter is provided in combination with REST API framework version 2 or later, the specified children are included in the resource payload (instead of just a link). The value of this query parameter is `all` or `<accessor1>,<accessor2>,...` When `all` is specified, only the top-level children will be included in the resource payload. More than one child can be specified using comma as a separator. Example: `?expand=Employee,Localization`. Nested children can also be provided following the format "`Child.NestedChild`" (Example: `?expand=Employee.Manager`). If a nested child is provided (Example: `Employee.Manager`), the missing children will be processed implicitly. For example, `?expand=Employee.Manager` is the same as `?expand=Employee,Employee.Manager` (which will expand `Employee` and `Manager`).

  Note the `expand` parameter cannot be combined with the `fields` parameter. If both parameters are provided, only `fields` will be considered.

- **Content-Type**

  none

- **Payload**

  none

Response From Framework Version 3

- **HTTP Code**

  `200`

- **Content-Type**

  `application/vnd.oracle.adf.resourceitem+json`

- **Payload**

```
{
  "DepartmentId" : 50,
  "DepartmentName" : "Shipping",
  "Employee" : {
    "items" : [ {
       "EmployeeId" : 120,
       "FirstName" : "Matthew",
       "LastName" : "Weiss",
       "Email" : "MWEISS",
       "JobId" : "ST_MAN",
       "DepartmentId" : 50,
       "Salary" : 8000,
       "links" : [ {
         "rel" : "self",
         "href" : "<base_url>/Department/50/child/Employee/120",
         "name" : "Employee",
         "kind" : "item"
       }, {
         "rel" : "parent",
         "href" : "<base_url>/Department/50",
```

```
            "name" : "Department",
            "kind" : "item"
          } ]
        }, {
          "EmployeeId" : 121,
          "FirstName" : "Adam",
          "LastName" : "Fripp",
          "Email" : "AFRIPP",
          "JobId" : "ST_MAN",
          "DepartmentId" : 50,
          "Salary" : 8200,
          "links" : [ {
            "rel" : "self",
            "href" : "<base_url>/Department/50/child/Employee/121",
            "name" : "Employee",
            "kind" : "item"
          }, {
            "rel" : "parent",
            "href" : "<base_url>/Department/50",
            "name" : "Department",
            "kind" : "item"
          } ]
        }, {
          ...
          } ]
        } ],
        "count" : 3,
        "hasMore" : false,
        "limit" : 25,
        "offset" : 0,
        "links" : [ {
            "rel" : "self",
            "href" : "<base_url>/Department/50/child/Employee",
            "name" : "Employee",
            "kind" : "collection"
        } ]
      },
      "links" : [ {
          "rel" : "self",
          "href" : "<base_url>/Department/50",
          "name" : "Department",
          "kind" : "item"
       } ]
  }
```

REST API Framework Version 1 or Version 2

Version 1 and version 2 of the REST API framework return the nested child resource expanded in the response payload as an array of resource items. If the resource collection being fetched is large, you may have to make several requests since the array of resource items has a limit.

The following samples illustrate functionality for REST API framework version 1 and version 2.

The first request sample (URL 1) retrieves a single child resource item identified by employee `120`. The URL parameter `child` identifies the relationship of the requested resource `Employee`.

The second request (URL 2) shows the use of the query parameter `expand` to ensure that all nested `Employee` resource items will be returned with `Department` resource collection `50`.

The third request (URL 3) shows the use of accessor dot notation (for example, `Employee.JobHistory`) in combination with the query parameter `expand` to ensure that all nested `JobHistory` resource items will be returned with the `Employee` resource items for the `Department` resource collection `80`.

Request Made With Framework Version 1 or Version 2

- **URL 1**

  `<base_url>/Department/50/child/Employee/120`

- **URL 2**

  `<base_url>/Department/50?expand=Employee`

- **URL 3**

  `<base_url>/Department/80?expand=Employee.JobHistory&onlyData=true`

- **HTTP Method**

  GET

- **Query Parameter**

  `expand`

  When this parameter is provided in combination with REST API framework version 1, the specified children are included as links in the resource payload. The value of this query parameter is `all` or `<accessor1>,<accessor2>,...` More than one child can be specified using comma as a separator. Example: `?expand=Employee,Localization`. Nested children can also be provided following the format "`Child.NestedChild`" (Example: `?expand=Employee.Manager`). If a nested child is provided (Example: `Employee.Manager`), the missing children will be processed implicitly. For example, `?expand=Employee.Manager` is the same as `?expand=Employee,Employee.Manager` (which will expand `Employee` and `Manager`).

- **Content-Type**

  none

- **Payload**

  none

Response From Framework Version 1 or Version 2

- **HTTP Code**

  `200`

- **Content-Type**

  `application/vnd.oracle.adf.resourceitem+json`

- **Payload 1**

```
{
  "EmployeeId" : 120,
  "FirstName" : "Matthew",
  "LastName" : "Weiss",
  "Email" : "MWEISS",
```

```
  "JobId" : "ST_MAN",
  "DepartmentId" : 50,
  "Salary" : 8000,
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Department/50/child/Employee/120",
    "name" : "Employee",
    "kind" : "item"
  }, {
    "rel" : "parent",
    "href" : "<base_url>/Department/50",
    "name" : "Department",
    "kind" : "item"
  } ]
}
```

- **Payload 2**

```
{
  "DepartmentId" : 50,
  "DepartmentName" : "Shipping",
  "Employee" : [ {
    "EmployeeId" : 120,
    "FirstName" : "Matthew",
    "LastName" : "Weiss",
    "Email" : "MWEISS",
    "JobId" : "ST_MAN",
    "DepartmentId" : 50,
    "Salary" : 8000,
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/50/child/Employee/120",
      "name" : "Employee",
      "kind" : "item"
    }, {
      "rel" : "parent",
      "href" : "<base_url>/Department/50",
      "name" : "Department",
      "kind" : "item"
    } ]
  }, {
    "EmployeeId" : 121,
    "FirstName" : "Adam",
    "LastName" : "Fripp",
    "Email" : "AFRIPP",
    "JobId" : "ST_MAN",
    "DepartmentId" : 50,
    "Salary" : 8200,
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/50/child/Employee/121",
      "name" : "Employee",
      "kind" : "item"
    }, {
      "rel" : "parent",
      "href" : "<base_url>/Department/50",
      "name" : "Department",
      "kind" : "item"
    } ]
  }, {
     ...
    } ]
```

```
  } ],
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Department/50",
    "name" : "Department",
    "kind" : "item"
  } ]
}
```

- **Payload 3**

```
{
    "DepartmentId" : 80,
    "DepartmentName" : "Sales",
    "RelState" : null,
    "Employee" : [
        ...
        {
            "EmployeeId" : 176,
            "FirstName" : "Jonathon",
            "LastName" : "Taylor",
            "Email" : "JTAYLOR",
            "JobId" : "SA_REP",
            "DepartmentId" : 80,
            "Salary" : 8600,
            "CommissionPct" : 0.2,
            "JobHistory" : [
                {
                    "EmployeeId" : 176,
                    "StartDate" : "2011-03-24",
                    "EndDate" : "2012-12-31",
                    "JobId" : "SA_REP",
                    "DepartmentId" : 80
                },
                {
                    "EmployeeId" : 176,
                    "StartDate" : "2013-01-01",
                    "EndDate" : "2015-03-31",
                    "JobId" : "SA_MAN",
                    "DepartmentId" : 80
                }
            ]
        },
        ...
    ]
}
```

Fetching Data Only for a Business Object

REST APIs support retrieving only the data of a resource collection.

The following sample fetches the values of the `Employee` resource collection attributes. The query parameter `onlyData` ensures the resource is filtered to contain only data in the response payload and no links.

Request

- **URL**

  `<base_url>/Employee?onlyData=true`

- **HTTP Method**

GET

- **Query Parameter**

  `onlyData`

  This parameter filters the resource item payload to contain only data (no links section, for example).

- **Content-Type**

  none

- **Payload**

  none

Response

- **HTTP Code**

  `200`

- **Content-Type**

  `application/vnd.oracle.adf.resourceitem+json`

- **Payload**

```
{
  "items" : [ {
    "EmployeeId" : 101,
    "FirstName" : "Neena",
    "LastName" : "Smith",
    "Email" : "NSMITH",
    "JobId" : "AD_VP",
    "DepartmentId" : 90,
    "Salary" : 2000
  }, {
    "EmployeeId" : 102,
    "FirstName" : "Lex",
    "LastName" : "De Haan",
    "Email" : "LDEHAAN",
    "JobId" : "AD_VP",
    "DepartmentId" : 90,
    "Salary" : 3000
  }, {
    "EmployeeId" : 103,
    "FirstName" : "Alexander",
    "LastName" : "Hunold",
    "Email" : "AHUNOLD",
    "JobId" : "IT_PROG",
    "DepartmentId" : 60,
    "Salary" : 4000
  }, {
    "EmployeeId" : 104,
    "FirstName" : "Bruce",
    "LastName" : "Ernst",
    "Email" : "BERNST",
    "JobId" : "IT_PROG",
    "DepartmentId" : 60,
    "Salary" : 5000
  }, {
    "EmployeeId" : 105,
    "FirstName" : "David",
```

```
    "LastName" : "Austin",
    "Email" : "DAUSTIN",
    "JobId" : "IT_PROG",
    "DepartmentId" : 60,
    "Salary" : 6000
  } ],
  "count" : 5,
  "hasMore" : true,
  "limit" : 25,
  "offset" : 0,
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Employee",
    "name" : "Employee",
    "kind" : "collection"
  } ]
}
```

Filtering a Business Object with a Query Parameter

REST APIs support fetching a resource collection using query expression syntax to filter resource items.

The resource collection may be queried using expressions that differ in syntax depending on the REST API framework version that has been registered for the web application. For details about the REST API framework versions, see Understanding REST API Framework Version Support.

The following samples are based on two different versions of the `Department` resource. The URL sample showing resource `1.0` reflects the query-by-example query parameter syntax supported only by version 1 of the REST API framework. While the URL sample showing resource `2.0`, reflects the rowmatch query parameter syntax supported in REST API framework version 2 (and later). In both framework scenarios, the samples fetch a filtered set of resource items of the `Department` resource.

> **Note:**
>
> For a REST API request, reserved characters that appear in a query parameter value should be encoded. For example, the + character in a timestamp value must be encoded as `%2B`. Additionally, resource and resource items names used in query parameter operations are case sensitive.

REST API Framework Version 2 (and later)

Starting with version 2 of the REST API framework, web applications may use an advanced query syntax, also known as rowmatch expressions, to fetch resources. For a complete description of the query syntax available in version 2 (and later), Understanding Framework Support for Query Syntax.

The following sample fetches all departments with at least one employee whose salary is equal to 10000. This is an example of fetching a parent resource collection (`Department`) and filtering it by a child resource collection attribute (`Employee.Salary`).

Request Example 1 Made With Framework Version 2

- **URL**

  `<base_url>/Department?q=Employee.Salary = 10000`

- **HTTP Method**

  GET

- **Query Parameter**

  `q`

  This parameter filters the resource collection based on one or more attribute value expressions. Starting with REST API framework version 2, complex filters may combine expressions using the `and` and `or` conjunctions with matching sets of parentheses for grouping. For example, `?q=(Deptno>=10 and <= 30) and (Loc!=NY)`.

- **Content-Type**

  none

- **Payload**

  none

Response Example 1 From Framework Version 2

- **HTTP Code**

  `200`

- **Content-Type**

  `application/vnd.oracle.adf.resourcecollection+json`

- **Payload**

```
{
  "items" : [ {
    "DepartmentId" : 70,
    "DepartmentName" : "Public Relations",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/70",
      "name" : "Department",
      "kind" : "item"
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/70/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  }, {
    "DepartmentId" : 80,
    "DepartmentName" : "Sales",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/820",
      "name" : "Department",
      "kind" : "item"
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/80/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
```

```
    } ]
  } ],
  "count" : 2,
  "hasMore" : false,
  "limit" : 25,
  "offset" : 0,
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Department",
    "name" : "Department",
    "kind" : "collection"
  } ]
}
```

The following sample fetches all departments with the numeric ID of `10` or a department name that begins with the letter "H".

Request Example 2 Made With Framework Version 2

- **URL**

  `<base_url>/Department?q=DepartmentId = 10 or DepartmentName like 'H*'`

- **HTTP Method**

  GET

- **Query Parameter**

  `q`

  This parameter filters the resource collection based on one or more attribute value expressions. Starting with REST API framework version 2, complex filters may combine expressions using the `and` and `or` conjunctions with matching sets of parentheses for grouping. For example, `?q=(Deptno>=10 and <= 30) and (Loc!=NY)`.

- **Content-Type**

  none

- **Payload**

  none

Response Example 2 From Framework Version 2

- **HTTP Code**

  `200`

- **Content-Type**

  `application/vnd.oracle.adf.resourcecollection+json`

- **Payload**

```
{
  "items" : [ {
    "DepartmentId" : 10,
    "DepartmentName" : "Administration",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/10",
      "name" : "Department",
      "kind" : "item"
```

```
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/10/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  }, {
    "DepartmentId" : 40,
    "DepartmentName" : "Human Resources",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/40",
      "name" : "Department",
      "kind" : "item"
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/40/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  } ],
  "count" : 2,
  "hasMore" : false,
  "limit" : 25,
  "offset" : 0,
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Department",
    "name" : "Department",
    "kind" : "collection"
  } ]
}
```


REST API Framework Version 1

Version 1 of the REST API framework supports a query-by-example syntax. No application configuration changes are required to use this syntax that is supported only in versions 1. For a description of the query syntax available in version 1 of the REST API framework, see GET Method Endpoints.

> **Note:**
>
> In version 1 of the ADF REST framework, when you create a query with a string matching filter parameter and the string to match contains a query syntax reserved word (such as AND or OR), then the quoted string must be delimited by a space character to separate it from other parameters in the query expression. For example, the following query attempts to filter on the quoted string `'Accounting and Finance'`. Since the string contains the reserved word AND, the string matching filter parameter requires a space before and after the single quotes to be viable in version 1.
>
> ```
> ?q=DepartmentName= 'Accounting and Finance'
> &fields=DepartmentName,Location
> ```
>
> Note that starting in framework version 2, the use of a space character is no longer required to delimit a string matching filter that contains a reserved word.

The following sample fetches departments assigned a `DepartmentId` value less than `30`.

Request Made With Framework Version 1

- **URL**

  `<base_url>/Department?q=DepartmentId<30`

- **HTTP Method**

  GET

- **Query Parameter**

  `q`

  This parameter filters the resource collection based on one or more attribute value expressions. In REST API framework version 1, the value of this query parameter is a list of semi-colon separated query-by-example expressions. For example, `?q=Deptno=10 and <=30;Loc!=NY`.

- **Content-Type**

  none

- **Payload**

  none

Response From Framework Version 1

- **HTTP Code**

  `200`

- **Content-Type**

  `application/vnd.oracle.adf.resourcecollection+json`

- **Payload**

```
{
  "items" : [ {
    "DepartmentId" : 10,
    "DepartmentName" : "Administration",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/10",
      "name" : "Department",
      "kind" : "item"
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/10/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  }, {
    "DepartmentId" : 20,
    "DepartmentName" : "Marketing",
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/20",
      "name" : "Department",
      "kind" : "item"
    }, {
```

```
      "rel" : "child",
      "href" : "<base_url>/Department/20/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  } ],
  "count" : 2,
  "hasMore" : false,
  "limit" : 25,
  "offset" : 0,
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Department",
    "name" : "Department",
    "kind" : "collection"
  } ]
}
```

Creating Business Object Items

REST APIs support the HTTP POST method to create a resource item.

REST APIs support the following creation use cases:

- Creating a resource item in an existing resource collection.
- Creating a child item and parent resource item (where the resource collection of each item form a child-parent relationship) in one roundtrip.

Creating a Business Object Item

REST APIs support using the HTTP POST method to create a resource item in an existing resource collection.

The following sample creates a new resource item in the existing `Department` resource collection.

Request

- **URL**

  `<base_url>/Department`

- **HTTP Method**

  POST

- **Content-Type**

  `application/vnd.oracle.adf.resourceitem+json`

- **Payload**

```
{
  "DepartmentId" : 15,
  "DepartmentName" : "NewDept"
}
```

Response

- **HTTP Code**

  `201`

- **Content-Type**

  `application/vnd.oracle.adf.resourceitem+json`

- **Location**

  `<base_url>/Department/15`

- **Payload**

```
{
  "DepartmentId" : 15,
  "DepartmentName" : "NewDept",
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Department/15",
    "name" : "Department",
    "kind" : "item"
  }, {
    "rel" : "child",
    "href" : "<base_url>/Department/15/child/Employee",
    "name" : "Employee",
    "kind" : "collection"
  } ]
}
```

Creating an Item of a Child Business Object

REST APIs support creating a resource item in the child resource collection of an existing parent resource collection. Alternatively, REST APIs support creating a child item and the parent item in one roundtrip. Creating a child resource item and it's parent resource item will only succeed when both the child resource item and parent resource item do not exist.

The following samples create resource items using a POST method. The first request sample (URL1) creates a child resource item identified by employee `999` in the `Employee` resource collection nested in existing parent resource item `15` of the `Department` collection. The second request (URL2) creates a child resource item identified by employee `99999` in the `Employee` resource collection and also creates the parent resource item `17` of the `Department` collection in one roundtrip.

Request

- **URL 1**

  `<base_url>/Department/15/child/Employee`

- **URL 2**

  `<base_url>/Department`

- **HTTP Method**

  POST

- **Content-Type**

  `application/vnd.oracle.adf.resourceitem+json`

- **Payload 1**

```
{
     "EmployeeId": 999,
     "FirstName": "New",
```

```
    "LastName": "Guy",
    "Email": "NGUY",
    "JobId": "SA_REP",
    "DepartmentId": 15,
    "Salary": 9999
}
```

- **Payload 2**

```
{
    "DepartmentId": 17,
    "DepartmentName": "NewerDept",
    "Employee": [
        {
            "EmployeeId": 99999,
            "FirstName": "Newer",
            "LastName": "Guy",
            "Email": "NRGUY",
            "JobId": "SA_MAN",
            "DepartmentId": 17,
            "Salary": 10001
        }
    ]
}
```

Response

- **HTTP Code**

  201

- **Content-Type**

  application/vnd.oracle.adf.resourceitem+json

- **Location**

  <base_url>/Department/15/child/Employee/999

- **Payload 1**

```
{
  "EmployeeId" : 999,
  "FirstName" : "New",
  "LastName" : "Guy",
  "Email" : "NGUY",
  "JobId" : "SA_REP",
  "DepartmentId" : 15,
  "Salary" : 9999,
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Department/15/child/Employee/999",
    "name" : "Employee",
    "kind" : "item"
  }, {
    "rel" : "parent",
    "href" : "<base_url>/Department/15",
    "name" : "Department",
    "kind" : "item"
  } ]
}
```

- **Location**

  <base_url>/Department/17

- **Payload 2**

```
{
  "DepartmentId" : 17,
  "DepartmentName" : "NewerDept",
  "Employee" : [ {
    "EmployeeId" : 99999,
    "FirstName" : "Newer",
    "LastName" : "Guy",
    "Email" : "NRGUY",
    "JobId" : "SA_MAN",
    "DepartmentId" : 17,
    "Salary" : 10001,
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/17/child/Employee/99999",
      "name" : "Employee",
      "kind" : "item"
    }, {
      "rel" : "parent",
      "href" : "<base_url>/Department/17",
      "name" : "Department",
      "kind" : "item"
    } ]
  } ],
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Department/17",
    "name" : "Department,
    "kind" : "item"
  } ]
}
```

Updating a Business Object Item

REST APIs support the HTTP PATCH method to update the resource item. Update will only succeed when the row already exists.

The following sample updates department 15, where `DepartmentName` is changed in the request payload.

Request

- **URL**

  `<base_url>/Department/15`

- **HTTP Method**

  PATCH

- **Content-Type**

  `application/vnd.oracle.adf.resourceitem+json`

- **Payload**

```
{
  "DepartmentId" : 15,
  "DepartmentName" : "UpdatedDeptName"
}
```

Response

- **HTTP Code**

  `200`

- **Content-Type**

  `application/vnd.oracle.adf.resourceitem+json`

- **Payload**

```
{
  "DepartmentId" : 15,
  "DepartmentName" : "UpdatedDeptName",
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Department/15",
    "name" : "Department",
    "kind" : "item"
  }, {
    "rel" : "child",
    "href" : "<base_url>/Department/15/child/Employee",
    "name" : "Employee",
    "kind" : "collection"
  } ]
}
```

Updating or Creating a Business Object Item (Upsert)

REST APIs support using a POST method with Upsert mode enabled to update a resource item that exists, and if the item does not exist, then the operation will create the item.

You use a POST method and specify `true` for the `Upsert-Mode` header value to enable the Upsert functionality. If the header variable `Upsert-Mode` is not provided, or is set to `false`, then the Upsert functionality is disabled.

The following sample creates (through Upsert) a new `Department` resource item with a child `Employee` item. In this example, department `80` exists but employee `8080` does not. After the request is executed, the existing department `80` is updated, and the resource item, employee `8080`, is created. HTTP 200 is returned because the department item is updated. The response HTTP code 201 is used when the resource collection is created and not updated.

Request

- **URL**

  `<base_url>/Department`

- **HTTP Method**

  `POST`

- **Content-Type**

  `application/vnd.oracle.adf.resourceitem+json`

- **HTTP Header**

  `Upsert-Mode: true`

- **Payload**

```
{
    "Deptno": 80,
    "Dname": "ENG80",
    "Emp": [{
        "Empno": 8080,
        "Ename": "Smith",
        "Mgr": 8080
    }]
}
```

Response

- **HTTP Code**

  `200`

- **Content-Type**

  `application/vnd.oracle.adf.resourceitem+json`

- **Payload**

```
{
    "Deptno": 80,
    "Dname": "ENG80",
    "Emp": [{
        "Empno": 8080,
        "Ename": "Smith",
        "Mgr": 8080
    }]
}
```

Deleting a Business Object Item

REST APIs support the HTTP DELETE method to delete a resource item. REST APIs do not currently support deleting the resource collection.

The following sample (URL1) deletes the resource item, employee `99999` in department `17`. The second request URL deletes the resource item, department `17`.

Request

- **URL 1**

  `<base_url>/Department/17/child/Employee/99999`

- **URL 2**

  `<base_url>/Department/17`

- **HTTP Method**

  DELETE

- **Content-Type**

  none

- **Payload**

  none

Response

- **HTTP Code**

`204`

- **Content-Type**

  none
- **Payload**

  none

5

Data Consistency Tasks

You can perform data consistency checks while making REST API calls. This capability uses version history in the database to enable you to manage HTTP payloads according to updates in the resource itself.

Topics

About Data Consistency

REST APIs support checking for data consistency when updating or retrieving a resource item. Data consistency is enforced by the REST API by generating an entity tag (ETag) with precondition headers so that the resource item matches the server side resource state before updating or retrieving.

REST APIs support generating an entity tag (ETag) in the response header when the requested resource item has data consistency check enabled.

When your visual development tool supports entity change indicators (as it is, for example, in Oracle Visual Builder), the REST API will assign a unique value to indicate the state of each resource item on the server side. At runtime, when the business object item underlying the server side resource item changes, the REST API assigns a new state value to the ETag. The following header shows the ETag returned with a request to retrieve a `Department` resource item.

```
HTTP/1.1 200 OK
Cache-Control: no-cache, no-store, must-revalidate
Location:
Content-Length: 1069
Content-Type: application/json
ETag: "ACED000573720372001362616 36C6520136261636C6523720013626163 6C652"
Link: <base_url>/Department/10>;rel="self";kind="item";name="Department"
```

The web application request can use the ETag value returned in the header response of each resource item to create subsequent requests that contain precondition headers (`If-Match`/`If-None-Match`). Based on the specified ETag and the precondition, the server will evaluate the current resource item state and match against the provided ETag. If the precondition is satisfied, the requested operation is executed; otherwise, a `412` error is returned. The error payload will contain the current resource item in the server side and the header will also reflect the current ETag value.

To support testing ETag values, the REST API provides the following precondition header fields. Usage of these precondition fields forces the REST API to compare a supplied ETag value against the ETag values of previously requested items.

- Verify that the client is providing a state (obtained from a previous resource item response) that matches the current state on the server:

  `If-Match: "<ETag value from resource item response>"`

- Verify that the client is providing a state (obtained from a previous resource item response) that does **not** match the current state on the server.

  `If-None-Match: "<ETag value from resource item response>"`

The following are typical use cases when checking for data consistency:

- Check that the business object item matches the server side resource item state before updating
- Retrieve the business object item using the server side resource item state when none of the requested items match any previously requested items

While these use cases involve GET and PATCH methods, the precondition header and ETag value can be used to check that any HTTP method operation will be applied to the current state of the business object item.

When retrieving a resource collection, an additional custom property `changeIndicator` will appear in the response payload of resource with data consistency enabled. This property contains the current ETag value of each resource item in the requested collection. The following sample illustrates the `changeIndicator` property in the `links` section of a `Department` resource collection. The presence of ETag values in the resource collection payload is a convenience for the web application that can reduce the number of requests to obtain the ETag from individual resource items.

```
{
  "items" : [ {
    "DepartmentId" : 10,
    "DepartmentName" : "Administration",
    "RelState" : 1,
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/10",
      "name" : "Department",
      "kind" : "item",
      "properties" : {
        "changeIndicator" : "ACED0005737200136A6176612E7574696C2E41727261794C69737
        47881D21D99C7619D03000149000473697A657870000000017704000000001737200186F721
        636C652E6A626F2E646F6D61696E2E4E7564A362286F0200015B000464617461740025B45
        27870757200025B42ACF317F8060854E00200007870"
      }
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/10",
      "name" : "Department",
      "kind" : "item",
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/10/child/Employee",
      "name" : "Employee",
      "kind" : "collection",
    } ]
  }, {
    "DepartmentId" : 20,
    "DepartmentName" : "Marketing",
    "links" : [ {
      "rel" : "self",
```

```
      "href" : "<base_url>/Department/20",
      "name" : "Department",
      "kind" : "item",
      "properties" : {
        "changeIndicator" :
"ACED0005737200136A6176612E7574696C2E41727261794C6973747881D21D99C7619D03000149000473
697A657870000000017704000000017372001 86F7261636C652E6A626F2E646F6D61696E2E4E756D62657
2A5B1371914E0BFDA0200014900096D48617368436F6465787200116F7261636C652E73716C2E4E554D42
4552E90466EE632BE1D5020000787200106F7261636C652E73716C2E446174756D4078F514A362286F020
0015B000464617461740002 5B427870757200025B42ACF317F8060854E0020000787000000002C10A0000
000078"
      }
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/20/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
  }, {
      ...
    } ]
  } ],
  "count" : 5,
  "hasMore" : true,
  "limit" : 25,
  "offset" : 0,
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Department",
    "name" : "Department",
    "kind" : "collection"
  } ]
}
```


Checking for Data Consistency When Updating Business Object Items

REST APIs support checking for data consistency when updating resource items.

To check for data consistency using the ETag header and conditional header fields:

1. Query one or more business object items and, for each returned resource item, obtain the ETag value from the `changeIndicator` property in the `properties` section of the response. When querying multiple business object items, there will not be a single ETag response header. Instead, the ETag for each of the items in the response will be in the `properties` section.

   ```
   HTTP/1.1 200 OK
   Cache-Control: no-cache, no-store, must-revalidate
   Location:
   Content-Length: 861
   Content-Type: application/json
   ETag: "responseETag123"
   Link: <<base_url>/Department/10>;rel="self";kind="item";name="Department"
   Set-Cookie: JSESSIONID=jXvsJ1GpdkFJV5Jh0yk7D72vPZ42t8tLYDg74NRKFQzXdnsjG9vv!
   1113104013; path=/; HttpOnly
   X-ORACLE-DMS-ECID: 51f1ff4535af720c:-7e156247:148ec9eeb3b:-8000-00000000000001ad
   X-Powered-By: Servlet/2.5 JSP/2.1
   ```

```
{
  "DepartmentId" : 10,
  "DepartmentName" : "Administration",
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Department/10",
    "name" : "Department",
    "kind" : "item",
    "properties" : {
      "changeIndicator" : "responseETag123"
    }
  }, {
    "rel" : "child",
    "href" : "<base_url>/Department/10/child/Employee",
    "name" : "Employee",
    "kind" : "collection"
  } ]
}
```

2. Update the business object item, using a PATCH request and check for data consistency by supplying the following conditional header field:
   - `If-Match: "<ETag value from resource item response>"` to verify that the state of a requested resource item is current with the previous resource item response.

The following sample updates the `DepartmentName` field of department `10` when the `If-Match` precondition test is satisfied. In the first request (Request 1), the ETag value `responseETag123` is identical to the ETag of the current department `10` on the server side, indicating that the state of the resource item is consistent with the server side. Consequently, the update to `DepartmentName` is allowed.

In the subsequent request (Request 2), however, the ETag supplied in the `If-Match` precondition is unchanged and no longer matches the new ETag value the server has for department `10` resource item. As a consequence of the stale ETag value used in the second request, the update fails with an HTTP code `412`, indicating the precondition test failed, and the current ETag value `responseETag567` is returned in the response header. This occurs in production web applications when multiple users simultaneously access the same business object item. For example, when user 1 and user 2 both query the same item, the item has, for example, ETag value 1. Then, if user 1 successfully updates the item with ETag value 1, and user 2 attempts to update the same item with ETag value 1, the attempt will fail.

Request 1

- **URL 1**

  `<base_url>/Department/10`

- **HTTP Method**

  PATCH

- **Precondition 1**

  `If-Match: "responseETag123"`

- **Content-Type**

  `application/vnd.oracle.adf.resourceitem+json`

- **Payload 1**

```
{
  "DepartmentName" : "FirstAttempt_NewDepartmentName"
}
```

Response 1

- **HTTP Code**

  `200`

- **Content-Type**

  `application/vnd.oracle.adf.resourceitem+json`

- **ETag**

  `responseETag567`

- **Payload 1**

```
{
    "DepartmentId" : 10,
    "DepartmentName" : "FirstAttempt_NewDepartmentName",
    "RelState" : null,
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/10",
      "name" : "Department",
      "kind" : "item",
      "properties" : {
        "changeIndicator" : "responseETag567"
      }
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/10/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
}
```

Request 2

- **URL 2**

  `<base_url>/Department/10`

- **HTTP Method**

  PATCH

- **Precondition 2**

  `If-Match: "staleETag789"`

- **Content-Type**

  `application/vnd.oracle.adf.resourceitem+json`

- **Payload 2**

```
{
  "DepartmentName" : "SecondAttempt_NewDepartmentName"
}
```

Response 2

- **HTTP Code**

412 (Precondition failed)

- **Content-Type**

  `application/vnd.oracle.adf.resourceitem+json`

- **ETag**

  `responseETag567`

- **Payload 2**

```
{
    "DepartmentId" : 10,
    "DepartmentName" : "FirstAttempt_NewDepartmentName",
    "RelState" : null,
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/10",
      "name" : "Department",
      "kind" : "item",
      "properties" : {
        "changeIndicator" : "responseETag567"
      }
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/10/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
}
```

Checking for Data Consistency When Retrieving Business Object Items

REST APIs support checking for data consistency when retrieving resource items.

To check for data consistency using the ETag header and conditional header fields:

1. Query one or more business object items and, for each returned resource item, obtain the ETag value from the `changeIndicator` property in the `properties` section of the response. When querying multiple business object items, there will not be a single ETag response header. Instead, the ETag for each of the items in the response will be in the `properties` section.

```
HTTP/1.1 200 OK
Cache-Control: no-cache, no-store, must-revalidate
Location:
Content-Length: 861
Content-Type: application/json
ETag: "responseETag123"
Link: <base_url>/Department/10>;rel="self";kind="item";name="Department"
Set-Cookie: JSESSIONID=jXvsJ1GpdkFJV5Jh0yk7D72vPZ42t8tLYDg74NRKFQzXdnsjG9vv!
1113104013; path=/; HttpOnly
X-ORACLE-DMS-ECID: 51f1ff4535af720c:-7e156247:148ec9eeb3b:-8000-00000000000001ad
X-Powered-By: Servlet/2.5 JSP/2.1

{
  "DepartmentId" : 10,
  "DepartmentName" : "Administration",
```

```
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Department/10",
    "name" : "Department",
    "kind" : "item",
    "properties" : {
      "changeIndicator" : "responseETag123"
    }
  }, {
    "rel" : "child",
    "href" : "<base_url>/Department/10/child/Employee",
    "name" : "Employee",
    "kind" : "collection"
  } ]
}
```

2. Query one or more business object items and check for data consistency by supplying the following conditional header field:

   - `If-None-Match: "<ETag value from resource item response>"` to verify that the state of none of the previously requested resource items is current with the resource item request.

The following sample retrieves department `10` when the `If-None-Match` precondition test is satisfied. In the first request (Request 1), the ETag value `responseETag123` matches the ETag of the previously requested Department `10` resource item on the server side, indicating that the state of the resource item is consistent with the server side. Consequently, the precondition fails and there is no need to return a newer department `10` resource item. The request returns with an HTTP code `304`, indicating the state on the server has not been modified.

In the subsequent request (Request 2), however, the ETag `unmatchedETagXYZ` supplied in the `If-None-Match` precondition does not exist on the server. As a consequence, the precondition succeeds and department `10` is retrieved. The request returns an HTTP code `200`, indicating the state had changed, and the current (unchanged) ETag value `responseETag123` is returned in the response header.

Request 1

- **URL 1**

  `<base_url>/Department/10`

- **HTTP Method**

  GET

- **Precondition 1**

  `If-None-Match: "responseETag123"`

- **Content-Type**

  none

- **Payload**

  none

Response 1

- **HTTP Code**

  `304` state not modified

- **Content-Type**

  none

- **Payload 1**

  none

Request 2

- **URL 2**

  `<base_url>/Department/10`

- **HTTP Method**

  GET

- **Precondition 2**

  `If-None-Match: "unmatchedETagXYZ"`

- **Content-Type**

  none

- **Payload**

  none

Response 2

- **HTTP Code**

  `200`

- **Content-Type**

  `application/vnd.oracle.adf.resourceitem+json`

- **ETag**

  `responseETag123`

- **Payload 2**

```
{
    "DepartmentId" : 10,
    "DepartmentName" : "Administration",
    "RelState" : null,
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Department/10",
      "name" : "Department",
      "kind" : "item",
      "properties" : {
        "changeIndicator" : "responseETag123"
      }
    }, {
      "rel" : "child",
      "href" : "<base_url>/Department/10/child/Employee",
      "name" : "Employee",
      "kind" : "collection"
    } ]
}
```

6
Advanced Tasks

You can use REST API to perform advanced operations such as obtaining a count of resource items in a resource collection, executing custom actions, and executing batch requests.

Topics

Returning the Estimated Count of Business Object Items

REST APIs support retrieving the estimated item count in the resource collection.

The following sample estimates the total records and queries the first two items in the `Employee` collection. The query parameter `totalResults` ensures the response payload contains the `totalResults` attribute.

Request

- **URL**

  `<base_url>/Employee?totalResults=true&limit=2`

- **HTTP Method**

  GET

- **Query Parameter**

  `totalResults`

  This parameter when set to `true` will include the estimated item count in the response for the resource collection. Otherwise the count is not included. The default value is `false`.

- **Content-Type**

  none

- **Payload**

  none

Response

- **HTTP Code**

  `200`

- **Content-Type**

  `application/vnd.oracle.adf.resourceitem+json`

- **Payload**

```
{
  "items" : [ {
    "EmployeeId" : 101,
    "FirstName" : "Neena",
    "LastName" : "Smith",
    "Email" : "NSMITH",
    "JobId" : "AD_VP",
    "DepartmentId" : 90,
    "Salary" : 2000,
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Employee/NSMITH",
      "name" : "Employee",
      "kind" : "item"
    } ]
  }, {
    "EmployeeId" : 102,
    "FirstName" : "Lex",
    "LastName" : "De Haan",
    "Email" : "LDEHAAN",
    "JobId" : "AD_VP",
    "DepartmentId" : 90,
    "Salary" : 3000,
    "links" : [ {
      "rel" : "self",
      "href" : "<base_url>/Employee/LDEHAAN",
      "name" : "Employee",
      "kind" : "item"
    } ]
  } ],
  "totalResults" : 5,
  "count" : 2,
  "hasMore" : true,
  "limit" : 2,
  "offset" : 0,
  "links" : [ {
    "rel" : "self",
    "href" : "<base_url>/Employee",
    "name" : "Employee",
    "kind" : "collection"
  } ]
}
```


Making Batch Requests

REST APIs support executing multiple operations in a single roundtrip using a batch request. The data is committed at the end of the request. However, if one request part in a batch request fails, then all changes are rolled back and an error response is returned.

A batch request can consist of a combination of create, update, get, delete, invoke custom methods and describe requests. The path parameter and the payload needs to be the same as what you use to invoke the request directly. The get method supports the same URL parameters in the batch request as a separate HTTP request.

The following sample illustrates a successful batch operation that executes operations in four parts: 1) update employee 101, 2) update employee 102, 3) update employee 103, 4) query employee 104.

Request

- **URL**

  `<base_url>`

- **HTTP Method**

  POST

- **Content-Type**

  `application/vnd.oracle.adf.batch+json`

- **Payload**

```
{
    "parts": [{
        "id": "part1",
        "path": "/Employee/101",
        "operation": "update",
        "payload": {
            "Salary": 10000
        }
    }, {
        "id": "part2",
        "path": "/Employee/102",
        "operation": "update",
        "payload": {
            "Salary": 10000
        }
    }, {
        "id": "part3",
        "path": "/Employee/103",
        "operation": "update",
        "payload": {
            "Salary": 10000
        }
    }, {
        "id": "part4",
        "path": "/Employee?q=EmployeeId%3D101",
        "operation": "get"
    }]
}
```

Response

- **HTTP Code**

  200

- **Content-Type**

  `application/vnd.oracle.adf.batch+json`

- **Payload**

```
{
    "parts": [{
        "id": "part1",
        "path": "<base_url>/Employee/101",
        "operation": "update",
        "payload": {
            "EmployeeId": 101,
            "FirstName": "Neena",
            "LastName": "Smith",
```

```
            "Email": "NSMITH",
            "JobId": "AD_VP",
            "DepartmentId": 90,
            "Salary": 10000,
            "links": [{
                "rel": "self",
                "href": "<base_url>/Employee/101",
                "name": "Employee",
                "kind": "item"
            } ]
        }
    }, {
        "id": "part2",
        "path": "<base_url>/Employee/102",
        "operation": "update",
        "payload": {
            "EmployeeId": 102,
            "FirstName": "Lex",
            "LastName": "De Haan",
            "Email": "LDEHAAN",
            "JobId": "AD_VP",
            "DepartmentId": 90,
            "Salary": 10000,
            "links": [{
                "rel": "self",
                "href": "<base_url>/Employee/102",
                "name": "Employee",
                "kind": "item"
            } ]
        }
    }, {
        "id": "part3",
        "path": "<base_url>/Employee/103",
        "operation": "update",
        "payload": {
            "EmployeeId": 103,
            "FirstName": "Alexander",
            "LastName": "Hunold",
            "Email": "AHUNOLD",
            "JobId": "IT_PROG",
            "DepartmentId": 60,
            "Salary": 10000,
            "links": [{
                "rel": "self",
                "href": "<base_url>/Employee/103",
                "name": "Employee",
                "kind": "item"
            } ]
        }
    }, {
        "id": "part4",
        "path": "<base_url>/Employee",
        "operation": "get",
        "payload": {
            "EmployeeId": 101,
            "FirstName": "Neena",
            "LastName": "Smith",
            "Email": "NSMITH",
            "JobId": "AD_VP",
            "DepartmentId": 90,
            "Salary": 10000,
```

```
                "links": [{
                    "rel": "self",
                    "href": "<base_url>/Employee/101",
                    "name": "Employee",
                    "kind": "item"
                } ]
            }
        }]
    }
```

Working with Error Responses

Error responses can be obtained in the form of a JSON payload in the form of HTTP status codes and error messages.

In addition to HTTP status codes and error messages, it is possible to obtain exception details in the response when your request is enabled to use REST API framework version 4 and the request is made for either `application/vnd.oracle.adf.error+json` or `application/json` media types. With framework version 4, the response will be in the form an exception detail payload which provides the following benefits to the web application:

- If multiple errors occur in a single request, the details of each error are presented in a hierarchical structure.
- An application-specific error code may be present that identifies the exception corresponding to each error.
- An error path may be present that identifies the location of each error in the request payload structure.

> **Note:**
>
> The exception detail may or may not present certain details, such as the application-specific error code and the request payload's error path.

For example, compare the error response for a POST submitted with a payload that contains the following incorrectly formatted date field when framework version 3 (or earlier) is enable and when framework version 4 (or later) is enabled.

```
{    "EmpNum" : 5027,
     "EmpName" : "John",
     "EmpHireDate" : "not a date"
}
```

Standard Error Response, Version 3 and earlier

Without framework version 4, no response payload is generated and instead only a single error message that does not reference the request payload will be returned in the response.

```
"An instance of type oracle.jbo.domain.Date cannot be created from string
not a date. The string value must be in format YYYY-MM-DDTHH:MI:SS.sss
+hh:mm."
```

Exception Payload Error Response, Version 4 and later

With framework version 4 enabled, the following exception detail payload is generated for the response. The payload includes the usual HTTP status code and formats the details of one or more exceptions in an array structure.

```
{    "title" : "Bad Request",
    "status" : "400",
    "o:errorDetails" : [ {
     "detail" : "An instance of type oracle.jbo.domain.Date cannot be
created from string not a date.
                The string value must be in format YYYY-MM-DDTHH:MI:SS.sss
+hh:mm.",
       "o:errorCode" : "26099",
       "o:errorPath" : "/EmpHireDate"
    } ]
 }
```

Understanding the Exception Payload Error Response

The exception detail payload will be generated for a REST API error response when the following conditions exist:

- REST API framework version is version 4.
- Either `application/vnd.oracle.adf.error+json` or `application/json` is the media type for the response.

The exception detail payload is a JSON object with the following structure:

```
{   "title" : "Message as per HTTP status code",
    "status" : "HTTP error code",
    "o:errorDetails" : [
     ...
        {
            "detail" : "Message of detail error",
            "o:errorCode" : "error code"
            "o:errorPath" : "JSON pointer to the location of the error in
the request payload"
        },
        ...
    ]
}
```

You opt into the exception payload as the error responses by using framework version 4 and making a request for either the `application/vnd.oracle.adf.error+json` media type or `application/json` media type.

Note that within the exception payload `o:errorDetails` can vary as per the number and the types of errors encountered. Additionally, the error code and error path are not guaranteed to be present in the response payload and should not be relied upon by web applications.

Obtaining an Exception Payload Error Response

REST APIs support obtaining exception details in the response when the request is made with an appropriate media type.

Starting with version 4 of the REST API framework, web applications may obtain an error response with a detailed exception payload.

The following sample attempts to create the department object with a new department item. However, for this example the request fails because the item for the department already exists.

Notice in the exception payload the `o:errorDetails` array provides the error path for where the error occurred in the request object; however, these particular details may not always be available to web applications.

Request Example 1

- **URL**

  `<base_url>/Department`

- **HTTP Method**

  POST

- **Accept Header**

  `application/vnd.oracle.adf.resourceitem+json,application/json`

- **Payload**

```
{
    "DeptNum" : 50,
    "DeptName" : "SALES",
}
```

Response Example 1

- **HTTP Code**

  `400`

- **Content-Type**

  `application/json`

- **Payload**

```
{
   "title" : "Bad Request",
   "status" : "400",
   "o:errorDetails" : [ {
      "detail" : "A department with the same name already exists. Please provide
a different name.",
      "o:errorCode" : "Dept_Rule_0"
  } ]
}
```

The following sample attempts to create the department object with a new department item. However, for this example the request fails because the employee names entered exceed the number of characters allowed by the validation rule defined for the `EmpName` field.

Request Example 2

- **URL**

  `<base_url>/Department`

- **HTTP Method**

  POST

- **Accept Header**

  `application/vnd.oracle.adf.resourceitem+json,application/vnd.oracle.adf.error+json`

- **Payload**

```
{
   "DeptNum" : 52,
   "DeptName" : "newDept522",
   "Employee" : [ {
        "EmpNum" : 501,
        "EmpName" : "MILLERSxxxxxxxxxxxxxxxxx"
    }, {
        "EmpNum" : 502,
        "EmpName" : "JONESPxxxxxxxxxxxxxxxxx"
      } ]
}
```

Response Example 2

- **HTTP Code**

  `400`

- **Content-Type**

  `application/vnd.oracle.adf.error+json`

- **Payload**

```
{
   "title" : "Bad Request",
   "status" : "400",
   "o:errorDetails" : [ {
      "detail" : "Value MILLERSxxxxxxxxxxxxxxxxx for field EmpName exceeds the
maximum length allowed.",
      "o:errorCode" : "27040",
      "o:errorPath" : "/Employee/0/EmpName"
   }, {
      "detail" : "Value JONESPxxxxxxxxxxxxxxxxx for field EmpName exceeds the
maximum length allowed.",
      "o:errorCode" : "27040",
      "o:errorPath" : "/Employee/1/EmpName"
   } ]
}
```

The following sample attempts to perform a batch operation. However, for this example the batch operation fails for the reasons shown in the exception detail payload of the error response.

Request Example 3

- **URL**

  `<base_url>`

- **HTTP Method**

  POST

- **Content-Type Header**

  `application/vnd.oracle.adf.batch+json`

- **Payload**

```
{
    "parts": [
        {
            "id": "part1",
            "path": "/Employee",
            "operation": "create",
        "payload" : {
        "EmpNum" : 1299,
        "EmpJob" : "CLERK",
        "EmpMgr" : 7566,
        "EmpHireDate" : null,
        "EmpSal" : 245,
        "EmpComm" : 0,
        "EmpDeptNum" : 30
        }
        },
        {
            "id": "part2",
            "path": "/Employee",
            "operation": "create",
            "payload": {
                "EmpNum" : 7589,
        "EmpName" : "SampleEmpxxxxxxxxxxxxxxxxxx",
        "EmpJob" : "CLERK",
        "EmpMgr" : 7566,
        "EmpHireDate" : null,
        "EmpSal" : 245,
        "EmpComm" : 0,
        "EmpDeptNum" : 30
            }
        },
        {
            "id": "part3",
            "path": "/Department",
            "operation": "create",
            "payload": {
               "DeptNum" : 52,
               "DeptName" : "newDept522",
               "Employee" : [
                   {
                       "EmpNum" : 7588,
                       "EmpName" : "SampleEmpxxxxxxxxxxxxxxxxxx",
                       "EmpJob" : "CLERK",
                       "EmpMgr" : 7566,
                       "EmpHireDate" : null,
                       "EmpSal" : 245,
                       "EmpComm" : 0,
                       "EmpDeptNum" : 30
                   }
               ]
           }
       },
    {
```

```
            "id": "part4",
            "path": "/Department/10/child/Loc",
            "operation": "get"
        },
    {
            "id": "part5",
            "path": "/Department?invQP=invVal",
            "operation": "get"
        },
    {
            "id": "part6",
            "path": "/Department/54",
            "operation": "delete"
        },
    {
            "id": "part7",
            "path": "/Department/54",
            "operation": "get"
        }
    ]
}
```

Response Example 3

- **HTTP Code**

  `400`

- **Content-Type**

  `application/vnd.oracle.adf.error+json`

- **Payload**

```
{
    "title" : "Bad Request",
    "status" : "400",
    "o:errorDetails" : [ {
        "detail" : "URL request parameter invQP cannot be used in this context.",
        "o:errorCode" : "27520"
    }, {
        "detail" : "Attribute EmpName in Emp is required.",
        "o:errorCode" : "27014",
        "o:errorPath" : "/parts/0"
    }, {
        "detail" : "Value SampleEmpxxxxxxxxxxxxxxxxxx for field EmpName exceeds
the maximum length allowed.",
        "o:errorCode" : "27040",
        "o:errorPath" : "/parts/1/payload/EmpName"
    }, {
        "detail" : "Attribute EmpName in Emp is required.",
        "o:errorCode" : "27014",
        "o:errorPath" : "/parts/1"
    }, {
        "detail" : "Value SampleEmpxxxxxxxxxxxxxxxxxx for field EmpName exceeds
the maximum length allowed.",
        "o:errorCode" : "27040",
        "o:errorPath" : "/parts/2/payload/Employee/0/EmpName"
    }, {
        "detail" : "Attribute EmpName in
AM.Dept_empWorksIn_deptToEmpQA_EmpViewDef is required.",
        "o:errorCode" : "27014",
        "o:errorPath" : "/parts/2"
```

```
    }, {
        "detail" : "Not Found",
        "o:errorCode" : "11404",
        "o:errorPath" : "/parts/3"
    } ]
}
```

Obtaining the Standard Error Message Response

REST APIs support generating an error message that describes the validation or system error when the request is made with REST API framework versions 1 through 3 enabled.

Before version 4 of the REST API framework, the error response returns a single error message and HTTP status code. Version 4 and later allows web applications to obtain an error response with a detailed exception payload.

The following sample attempts to update the `Departments` resource with a new department resource item. However, for this example the update fails because the item for the department already exists. The response is an error message because REST API framework version 4 (or later) is not enabled.

Request Example Made With Framework Version 3

- **URL**

  `http://server/demo/rest/11.2/Departments`

- **HTTP Method**

  POST

- **Content-Type**

  `application/vnd.oracle.adf.resourceitem+json`

- **Accept Header**

  `application/vnd.oracle.adf.resourceitem+json,application/json`

- **Payload**

```
{
    "DeptNum" : 50,
    "DeptName" : "SALES",
}
```

Response Example From Framework Version 3

- **HTTP Code**

  `400`

- **Error Response**

```
A department with the same name already exists. Please provide a
different name.
```

Part III
Reference

To use REST APIs you should be familiar with relevant details and concepts of the REST API.

Topics

A
Links and Relations

Business object relationships are represented in the response as URLs that define the relationship between the current business object and the business object that the link points to.

This appendix includes the following sections:

Describe links Object Structure

`links` is a JSON object where the value is always a URL link and the link name is defined according to the `rel` of the link. The `links` object is generated for each resource collection, item, and for the resource itself.

Note that URL links in the resource describe will be generated using a template placeholder value (`{id}`) when there is not enough information to determine all parts of the URL. For example, the following child link provides a URL with the placeholder for the value of the specific `Department` resource:

```
"item" : {
     "links" : [ {
        "rel" : "child",
        "href" : "<base_url>/Department/{id}/child/Employee",
        "name" : "Employee",
        "kind" : "collection",
        "cardinality" : {
          "value" : "1 to *",
          "sourceAttributes" : "DepartmentId",
          "destinationAttributes" : "DepartmentId"
        }
```

rel Attribute Values

The `rel` attribute defines the type of link relationship between the current resource and the resource which the link points to. Relationships may be specified by any of the values shown in the table below.

Table A-1 Link Relationship in the REST Resource Describe

Link Relationship	Description
`self`	Always generated for a resource. The `href` points to the resource itself or to the resource describe. In the `links` object, the link name is `self` for this `rel`.
`parent`	Always generated for a nested resource. The `href` points to the `self` link of the parent resource. In the `links` object, the link name is `parent` for this `rel`.
`child`	Generated when the resource has nested children. The `href` points to the nested collection. In the `links` object, the link name is the accessor name for this `rel`.
`current`	Generated in the resource version describe when multiple resource version identifiers exist. The `href` points to the most recent version identifier, as defined by the web application's version definition.
`predecessor-version`	Generated in the resource version describe when multiple resource version identifiers exist. The `href` points to the previous version identifier, as defined by the web application's version definition.
`successor-version`	Generated in the resource version describe when multiple resource version identifiers exist. The `href` points to the next most recent version identifier, as defined by the web application's version definition.
`describe`	Generated in the resource version describe. The `href` points to the resource catalog describe for all resources of the same version.

href Attribute Value

The `href` attribute defines the URL to the linked resource or resource describe.

cardinality Attribute Values

The `cardinality` attribute is an optional attribute that defines the cardinality between the source resource and the destination resource. This attribute will be available only when the `rel` attribute value is `child` and the resource type is `vnd.oracle.adf.description+json`. This cardinality attribute has the following attributes.

- `value`: The value of the cardinality. Example: `"1 to *"`
- `sourceAttributes`: The attribute in the source resource used to link to the destination resource.
- `destinationAttributes`: The attribute in the destination resource used to link to the source resource.

B
Framework Versions

REST APIs are supported by the REST API framework to access business objects. The REST API framework supports the exchange of information between the web application and server at runtime.

The REST API runtime supports specifying a framework version that affects the processing of the payload or indicate the default framework version (as configured by the server) to be used. When you specify a framework version to process requests, it allows the API to opt into those features when they are ready.

> **Note:**
>
> Each REST API framework version introduces functionality that the previous framework versions do not support. Thus, when you choose to opt into a later framework version, the REST API of your application may introduce backward incompatible changes on the service client consuming the REST API. In the table below, see the Does Not Support column for backward compatibility issues. See also Understanding REST API Framework Version Support.

The following table explains the changes for each framework version.

Table B-1 REST API Framework Versions

REST API Framework Version	Supports	Does Not Support
1 - Default version. Use to process requests for web applications when no other version is specified	Supports query-by-example resource query syntax Filtering resource collections using the `q` query parameter is limited to a query-by-example.	n/a
2 - You must specify the version for the request. Only then the REST API support the use of expanded expression syntax to process the request.	Supports more advanced query syntax for making REST API calls. Interprets `q` query parameter value differently than Framework version 1. Supports filtering resource collections using `rowmatch` query expressions.	Query-by-example resource query syntax is not compatible. Introduces a backward incompatible change to web application that rely on Framework version 1.

Table B-1 (Cont.) REST API Framework Versions

REST API Framework Version	Supports	Does Not Support
3 - The payload structure represents nested child resource as a resource collection, instead of an array of items as in version 1 and 2	Supports retrieving nested child resources with payload attributes that may be used by the web application to determine whether more resource items would be returned in a subsequent REST API request. Supports pagination of nested child resource that would otherwise require more than one request to fetch. Exposes functionality that allows `GET` operations to use the `?expand` and `?fields` query parameter to return a nested child resource as a resource collection with the `hasMore` attribute	Introduces a backward incompatible change to web application that rely on Framework version 1 or 2.
4 – Possible to obtain exception details in the response when your request is enabled to use REST API framework version 4 and the request is made for either `application/vnd.oracle.adf.error+json` or `application/json` media types.	Supports the response in the form an exception detail payload that provides the following benefits to the web application: • Presents the details of each error in a hierarchical structure if multiple errors occur in a single request. • Identifies the exception corresponding to each error by including an application specific error code. • Presents an error path that identifies the location of each error in the request payload structure.	The exception detail may or may not present certain details, such as the application-specific error code and the request payload's error path.
5 - **Note:** Framework version 5 is not supported for users of visual development tools provided by Oracle Cloud services.	n/a	n/a
6 - Supports differentiation between the resource fields and item information like links and headers.	Non-attribute fields like `links` and `headers` appear within `@context` field in the resource item response object. `links` field in `@context` will no longer have the `properties` field. `headers -> ETag` has the `changeIndicator` value. New field `key` under `@context` contains the unique identifier of the specific resource item as a string.	n/a

C

Media Types

Media types, also called MIME types or content types, define the allowed resource structure of the payload exchanged between the client and server. All REST API media types are based on JSON. Resources accessed in the web application fall under the `application` type and `json` subtype.

REST APIs use one of the media types listed in the table below. The types are defined such that the media type does not vary with the business object backing the resource. Note that the value of the accept header depends on the context of the invocation. Links to the JSON token structure of the REST API media types are provided in the following table.

> **Note:**
>
> As an alternative to specifying the supported media types, request accept headers passed with the REST API call can specify `application/json` when a superset of all supported media types may be accepted in the response.

Table C-1 Media Types Supported by REST APIs

Media Type	Invocation Context	Description
`application/vnd.oracle.adf.resourcecollection+json`	GET method	Represents the format for all resource collections returned by the REST API call. All attributes are automatically generated by the framework. Only the content of the `items` attribute is based on the resource definition. For an example, see Describing a Resource Collection.
`application/vnd.oracle.adf.resourceitem+json`	GET method POST method PATCH method	Represents the format for all resource items returned by the REST API call. Also represents the format for a resource item in a POST or PATCH request payload. Also represents the format for a resource item in a POST or PATCH request payload. Only the attribute `links` is automatically generated by the framework. All the other attributes are based on the resource definition.
`application/vnd.oracle.adf.actionresult+json`	POST method	Describes the result of an action execution.
`application/vnd.oracle.adf.description+json`	GET method	Describes the resource and its elements. For an example, see Retrieving the Resource Catalog Describe
`application/vnd.oracle.adf.batch+json`	POST method	Describes a set of operations to be performed, where the operation consists of a set of parts and each part represents a request. The batch request is executed in one single transaction. For an example, see Making Batch Requests.

Table C-1 (Cont.) Media Types Supported by REST APIs

Media Type	Invocation Context	Description
`application/vnd.oracle.adf.error+json`	any	Describes the exception payload error response for a request made with an error. To use this media type and obtain the exception details in an error response payload, the request must be made with REST API framework version 4 (or later) enabled. For an example, see Obtaining an Exception Payload Error Response.

D
Data Types

REST APIs support data types that are specified by the web application developer when they create the business object. At runtime, the framework exposes the data type of fetched resources as the resource describe attribute `type`.

The following table shows the relationship between the data types supported on business object fields and the corresponding REST data types that the REST API framework defines.

Table D-1 Data Types Supported by REST APIs

Business Object Field Data Type	REST Data Type
`Boolean`	`boolean`
`String`	`string`
`Number`	`number`
`Datetime`	`datetime`
`Date`	`date`
`Time`	`time`
`Reference`	`integer`
`Email`	`string`
`Percentage`	`string`
`Phone`	`string`
`Uri`	`string`

E

Status Codes

REST APIs support HTTP response status codes, where the specific code that is returned depends on the HTTP method invoked on the request.

REST APIs support the HTTP codes listed in the following table.

Table E-1 HTTP Codes Supported by REST APIs

HTTP Code	Description
`200 OK`	Request successfully executed and the response has content.
`201 Created`	Resource successfully created. The response contains the created resource.
`204 No Content`	Request successfully executed and the response doesn't have content.
`304 Not Modified`	According to the provided ETag, the resource was not modified.
`400 Bad Request`	The request could not be understood by the server due to malformed syntax.
`401 Unauthorized`	The server is refusing to service the request because the resource of the request is secured and authentication has not yet been provided.
`404 Not Found`	The requested resource was not found.
`406 Not Acceptable`	The business object identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request.
`412 Precondition failed`	The business object state in the server side doesn't match the provided ETag.
`415 Unsupported Media Type`	The server is refusing to service the request because the entity of the request is in a format not supported by the requested business object for the requested method.
`500 Internal Server Error`	The server encountered an unexpected condition which prevented it from fulfilling the request.

F

Response Headers

REST APIs support a variety of HTTP headers.

The REST API supports the HTTP response headers listed in the following table.

Table F-1 HTTP Headers Supported by REST APIs

HTTP Header Name	Description
`Content-Type`	Use to specify the content-type of the request/response payload. The REST API runtime is able to interpret (request/response) the media types, as described in Media Types.
`Accept`	Use to specify the expected content-type of the response payload. The REST API runtime is able to interpret (request/response) the media types, as described in Media Types.
`REST-Framework-Version`	Use to specify the version of the REST API framework to use to process the request. The REST API framework version passed in the version header overrides the default framework declaration defined by the web application, as described in About REST API Framework Versions.
`Location`	Use to identify the URI of a newly created business object. The REST API framework includes the `Location` header in the response of a POST to create a new business object. For an example, see Creating a Business Object Item.
`ETag`	Use to compare the state of the business object in a request with the state of business object on the server. The REST API framework supports the `ETag` generation for business objects that has been configured to use an change-indicator attribute. See About Data Consistency.
`If-Match`	Use to determine whether the state of the business object in a request is current with the business object on the server. This header is supported in order to execute conditional requests. See About Data Consistency.
`If-None-Match`	Use to determine whether the state of the business object in a request does not match the current state on the server. This header is supported in order to execute conditional requests. See About Data Consistency.
`Upsert-Mode`	Use `Upsert-Mode: true` in a request that uses `POST` to create a business object item if the item does not exist, or update a business object item if the item exists. Note that a POST request with `Upsert-Mode:false` behaves as a POST without the custom header and performs the CREATE operation exclusively. See Updating or Creating a Business Object Item (Upsert).

G

Endpoints

REST APIs support a variety of business object tasks that you can perform using standard HTTP methods in combination with an endpoint that provides details about the resource that you want to return.

This appendix contains the following sections:

GET Method Endpoints

REST APIs support the following operations using a GET method with the URI as shown.

- Describe the resource collection, resource item, or resource catalog (when collection and item are omitted).

  `<base_path>/[{resourceCollectionPath}|{resourceItemPath}]/describe`

- Retrieve the resource collection, optionally using a query string parameter.

  `<base_path>/{resourceCollectionPath}[?{queryStringParam}[&{queryStringParam}]]`

- Retrieve the resource item, optionally using a query string parameter.

  `<base_path>/{resourceItemPath}[?{queryStringParam}[&{queryStringParam}]]`

- Retrieve a specific version of the resource collection, the latest resource version, or all available resources (when version identifier and resource collection are omitted).

  `<base_path>/latest/{resourceCollectionPath}]`

Request Parameters

- The GET method supports query string parameters to query, filter, page, and sort the resource collection. The supported parameters are listed in the following table. All GET method URI parameters can be combined with any other parameter in the table, except where noted on the `expand` and `field` parameters. Note that query string parameters can only be used on resource media types. They cannot, for example, be used when describing the resource.

Note:

The results of the GET method or the query syntax may vary depending on the REST API framework version used for a client request. The following tables specify where the framework version is important to note when using query string parameters. For additional information about framework versions, see Working with REST API Framework Versions.

Table G-1 Supported GET Method Query String Parameters Used Only in Resource Collections

GET URI Parameter	Value	Description
q Starting in **REST API framework version 2**, q supports complex rowmatch expressions.	In framework version 1, the query parameter is used in the WHERE clause and contains one or more query by example-type expressions, separated by a semi-colon. Format: `<exp1>;<exp2>` Example: `?q=Deptno>=10 and <= 30;Loc!=NY` Starting in framework version 2, the query parameter accepts expressions that identify the specific rows to retrieve from the resource. The filter can be as simple as a single expression, or you can create more complex filters by combining expressions using the `and` and `or` conjunctions with matching sets of parentheses for grouping. For example, the following expression uses conjunction to query the resource using three different fields: `(AssignedToId is null) or ( (Priority <= 2) and (RecordName like 'TT-99%'))` If a query parameter value has a special character (like ';', ',', '=' or similar), then the value (in the expression) should be enclosed in quotes to define a literal value. If the literal value contains quotes, then, in addition, the quotes need to be escaped, as defined by the framework version, to be viable: • Version 1 syntax encloses the literal value in double quotes (") and requires a backslash (\) to escape double quote characters contained in the value: ? `q=NickName="\"Billy the Kid\""` • Version 2, or later, syntax encloses the literal value in single quotes (') and requires a single quote to escape a single quote character contained in the value (used as an apostrophe): ? `q=ListName='Bill''s list'` Additionally, in version 1, if the query parameter value contains a reserved word (such as `AND` or `OR`), then the	The resource collection will be queried using the provided expressions. Supported operators in framework version 1 and later: • = (Equal to) • > (Greater than) • < (Less than) • >= (Greater than or equal to) • <= (Less than or equal to) • `!=` (Not equal to) framework version 1 only • `AND` (And) • `OR` (Or) • `NOT` (Not) • `LIKE` (Like) Supported operators in framework version 2 and later also include: • <> (Not equal to) • `BETWEEN` (Between) • `NOT BETWEEN` (Not between) • `IN` (In) • `NOT IN` (Not in) • `IS NULL` (Null) • `IS NOT NULL` (Not Null) Allowed special characters in framework version 1: • `"` (double quotation mark) to define a literal value for use where special characters appear in the value • `'` (single quotation mark) to define a literal value for use where reserved words appear in the value • `\` (backslash) to define an escape character to escape a double quotation mark (") or single quotation mark/ apostrophe (') character used within a literal value • * (asterisk) to define a wildcard character Allowed special characters in framework version 2 and later: • `'` (single quotation mark) to define a literal value or to define an escape character to escape a

Table G-1 (Cont.) Supported GET Method Query String Parameters Used Only in Resource Collections

GET URI Parameter	Value	Description

	value needs to be enclosed in single quotes (') and there must be a space before and after the quoted string. For example, the string matching filter `'Accounting and Finance'` in the following query parameter expression contains the reserved word AND, and therefore requires a space before and after the single quotation marks to be viable in version 1 syntax:
	`?q=DepartmentName= 'Accounting and Finance' &fields=DepartmentName`
	Note that starting in framework version 2, the use of a space character is no longer required to delimit a string matching filter that contains a reserved word.
	For version 1, if the value contains both a special character and a reserved word, then the value needs to be enclosed in single quotes ('), and then enclosed again in double quotes (") , along with escaping the double quotes contained within the value with a backslash (\), and there needs to be a space before and after the twice-quoted string. For example:
	`?q=DepartmentName= "'\"Accounting\" and Finance'" &fields=DepartmentName`
	For a complete explanation of the expression format, see Understanding Framework Support for Query Syntax.
		single quotation mark/ apostrophe (') character used within a literal value
		• % (percent) to define a wildcard character
		For examples, see Filtering a Business Object with a Query Parameter and Understanding Framework Support for Query Syntax.

`totalResults`	boolean Default: `false`	The resource collection will include the estimated row count when `totalResults=true`. For an example, see Returning the Estimated Count of Business Object Items.

Table G-1 (Cont.) Supported GET Method Query String Parameters Used Only in Resource Collections

GET URI Parameter	Value	Description
`limit`	integer	This parameter restricts the number of resource items returned inside the resource collection. If `limit` exceeds the resource item total result, then the framework will return the available resources. For an example, see Paging a Business Object.
`offset`	integer Default: 0 (the first position)	Used to define the starting position of the resource collection. If offset exceeds the resource count, then no resources are returned. For an example, see Paging a Business Object.

Table G-2 Supported GET Method Query String Parameters Used in Resource Collections and Resource Items

GET URI Parameter	Value	Value
`fields` Starting in **REST API framework version 3 and later**, `fields` will return children resource items as a resource collection to support pagination of the collection.	A simple comma-separated list of resource item attributes. Format: `<attr1>,<attr2>` Example: `?fields=Dname,DLoc` May be used on child resources. Format: *`<accessor>:<att1>,<att2>`* Example: ? `fields=Employee:FirstName,LastName` May be used on nested resources using accessor dot notation. Format: *`<accessor1>.<accessor2>:<Attr1>,<Attr2>`* Example: ? `fields=Employee.JobHistory:JobId` Or on both resources in the nested resource. Format: *`<accessor1>:<Attr1>,<Attr2>;<accessor1>.<accessor2>:<Attr1>,<Attr2>`* Example: ? `fields=Name,Location;Employee:FirstName,LastName`	This parameter filters the resource item attributes. Only the specified attributes are returned. Note that if a nested resource is queried using the accessor dot notation (`Employee.JobHistory`), then attributes may be specified on both resources. A resource in the accessor dot notation without a specified attribute will return no resource item attributes. Note this parameter cannot be combined with the `expand` parameter. If both parameters are provided, only `fields` will be considered. For examples, see Fetching a Business Object with a Subset of Items.

Table G-2 (Cont.) Supported GET Method Query String Parameters Used in Resource Collections and Resource Items

GET URI Parameter	Value	Value
`onlyData`	boolean Default: `false`	This parameter filters the resource response in order to contain only data (no `links` objects, for example). For an example, see Fetching Data Only for a Business Object.
`expand` Starting in **REST API framework version 3 and later**, `expand` will return children resource items as a resource collection to support pagination of the collection.	Display all children. Format: `all` Display one or more child resource using a comma-separated list of accessors. Format: <*accessor1*>,<*accessor2*> Example: ? `expand=Employee,Localization` Display nested resources using the accessor dot notation. Format: <*accessor1*>.<*accessor2*> Example: ? `expand=Employee.JobHistory`	When this parameter is provided, the specified children are included in the resource response (instead of the link). Note the `expand` parameter cannot be combined with the `fields` parameter. If both parameters are provided, only `fields` will be considered. Note that if a nested resource is queried using the accessor dot notation (`Employee.JobHistory`), then the missing children will be processed implicitly. For example, ? `expand=Employee.JobHistory` is the same as ? `expand=Employee,Employee.JobHistory` (which will expand `Employee` and `JobHistory`). For examples, see Fetching a Child Business Object.
`dependency`	A set of dependency attributes. Format: `<attr1>=<val1>,<attr2>=<value2>` Example: `dependency=ProductId=2`	The dependencies are attributes that are set before and rolled back after generating the response. The dependencies attributes are always set in the resource item in question. When a child resource collection is requested and the `dependency` parameter is set, the attributes will be set in the parent resource item before generating the resource collection response.
`orderBy`	A comma-separated list of order-by attributes with a sort flag to specify ascending or descending order. Format: `<orderBy_attr1_name>[:<(asc/desc)>],<orderBy_attr2_name>[:<(asc/desc)>]` Example: `?orderBy=DName:desc,DLoc` Default: ORDERBY attributes defined on the business object query will be applied.	Sorts a resource collection based on its attributes. If the `asc`/`desc` are not provided (or an invalid value is provided), `asc` will be used as default. By default, the fetched collection will be sorted in a case insensitive way. For an example, see Sorting a Business Object.

Table G-2 (Cont.) Supported GET Method Query String Parameters Used in Resource Collections and Resource Items

GET URI Parameter	Value	Value
`links`	A comma separated list of `<rel_name>`, where `<rel_name>` is a string representing the relation type of a link. Example: `self, parent`	When a resource item or a resource collection is requested and the `links` query parameter is used, then only those links with relation types ("`rel`") matching the values in the comma-separated parameter value will be shown in the response. Note the `links` parameter cannot be combined with `onlyData` when `onlyData` has a value of `true`, as there will be no links section displayed in the response.

Table G-3 Supported GET Method Query String Parameters Used in Resource Catalog Describe

GET URI Parameter	Value	Value
`metadataMode`	`minimal` `list`	Use to retrieve the resource catalog describe. The URL parameter ? `metadataMode=minimal` is required to retrieve the describe. It returns the titles and links of parent resources (but does not include children resources). If you do not want any metadata in the response but only self links, you can append ?`metadataMode=list` to the describe request. Optionally, additional parameters may be appended to the minimal describe request, to include children resources and / or resource annotations, as explained for the query parameters `showAnnotations` and `includeChildren`. For example, you can append ? `metadataMode=minimal&includeChildren=true` to retrieve a minimal catalog describe with all children resources included. For an example, see Retrieving the Resource Catalog Describe.
`includeChildren`	`true`, `false` (default)	Use to include all available children resources nested within a parent resource describe. You can append `includeChildren=true` on the describe request. For a resource catalog describe example, see Retrieving the Resource Catalog Describe.

Table G-3 (Cont.) Supported GET Method Query String Parameters Used in Resource Catalog Describe

GET URI Parameter	Value	Value
`showAnnotations`	`true`, `false` (default)	To include resource annotations in the catalog describe, you can append `showAnnotations=true` on the describe request. Note that annotations must be defined by the web application developer and may not be present on the resource. You cannot use this parameter with ? `metadataMode=list`.

Query String Operators Supported by REST API Data Types

The following table shows the REST API data types and the valid operators that may be used in query strings with the query (`q`) parameter.

Note that the operators BETWEEN, NOT BETWEEN, IN, NOT IN, and the wildcard character % are available only starting in REST API framework version 2.

Table G-4 Operators Supported by Data Types in Query (q) String Parameter

REST API Data Type	Supported Operator

integer	• = (Equal to)
	`.../Department?q=Deptno = 20`
	• <> (Not equal to)
	`.../Department?q=Deptno <> 20`
	• < (Less than)
	`.../Department?q=Deptno < 20`
	• <= (Less than or equal to)
	`.../Department?q=Deptno <= 20`
	• > (Greater than)
	`.../Department?q=Deptno > 30`
	• >= (Greater than or equal to)
	`.../Department?q=Deptno >= 30`
	• `BETWEEN` (Between)
	`.../Department?q=Deptno BETWEEN 10 AND 30`
	• `NOT BETWEEN` (Not between)
	`.../Department?q=Deptno NOT BETWEEN 10 and 30`
	• `IN` (In)
	`.../Department?q=Deptno IN (10, 30)`
	• `NOT IN` (Not in)
	`.../Department?q=Deptno NOT IN (10, 30)`
	• `IS NULL` (Is null)
	`.../Department?q=Deptno IS NULL`
	• `NOT NULL` (Not null)
	`.../Department?q=Deptno NOT NULL`

Table G-4 (Cont.) Operators Supported by Data Types in Query (q) String Parameter

REST API Data Type	Supported Operator

number	• = (Equal to)
	`.../Department?q=Salary = 3120.99`
	• <> (Not equal to)
	`.../Department?q=Salary <> 3120.99`
	• < (Less than)
	`.../Department?q=Salary < 3120.99`
	• <= (Less than or equal to)
	`.../Department?q=Salary <= 3120.99`
	• > (Greater than)
	`.../Department?q=Salary > 3120.99`
	• >= (Greater than or equal to)
	`.../Department?q=Salary >= 3120.99`
	• `BETWEEN` (Between)
	`.../Department?q=Salary BETWEEN 2000 AND 3120.99`
	• `NOT BETWEEN` (Not between)
	`.../Department?q=Salary NOT BETWEEN 2000 and 3120.99`
	• `IN` (In)
	`.../Department?q=Salary IN (800, 3120.99)`
	• `NOT IN` (Not in)
	`.../Department?q=Salary NOT IN (800, 3120.99)`
	• `IS NULL` (Is null)
	`.../Department?q=Salary IS NULL`
	• `NOT NULL` (Not null)
	`.../Department?q=Salary NOT NULL`

Table G-4 (Cont.) Operators Supported by Data Types in Query (q) String Parameter

REST API Data Type	Supported Operator

string	• = (Equal to)
	`.../Department?q=DeptName = 'SALES'`
	• <> (Not equal to)
	`.../Department?q=DeptName <> 'SALES'`
	• `LIKE` (Like)
	`.../Department?q=DeptName LIKE 'SA%'`
	`.../Department?q=DeptName LIKE '%ES'`
	`.../Department?q=UPPER(DeptName) LIKE UPPER('%e%')`
	• `NOT LIKE` (Not like)
	`.../Department?q=UPPER(DeptName) NOT LIKE UPPER('%c%') IN (In)`
	`.../Department?q=DeptName IN ('SALES', 'RESEARCH')`
	• `NOT IN` (Not in)
	`.../Department?q=DeptName NOT IN ('SALES', 'RESEARCH')`
	• `IS NULL` (Is null)
	`.../Department?q=DeptName IS NULL`
	• `IS NOT NULL` (Is not null)
	`.../Department?q=DeptName IS NOT NULL`

Table G-4 (Cont.) Operators Supported by Data Types in Query (q) String Parameter

REST API Data Type	Supported Operator

date	• = (Equal to)
	`.../Employee?q=HireDate = '1999-01-01'`
	• <> (Not equal to)
	`.../Employee?q=HireDate <> '1999-01-01'`
	• < (Less than)
	`.../Employee?q=HireDate < '1999-01-01'`
	• <= (Less than or equal to)
	`.../Employee?q=HireDate <= '1999-01-01'`
	• > (Greater than)
	`.../Employee?q=HireDate > '1999-01-01'`
	• >= (Greater than or equal to)
	`.../Employee?q=HireDate >= '1999-01-01'`
	• `BETWEEN` (Between)
	`.../Employee?q=HireDate BETWEEN '1999-01-01' AND '2010-01-01'`
	• `NOT BETWEEN` (Not between)
	`.../Employee?q=HireDate NOT BETWEEN '1999-01-01' AND '2010-01-01'`
	• `IS NULL` (Is null)
	`.../Employee?q=HireDate IS NULL`
	• `NOT NULL` (Not null)
	`.../Employee?q=HireDate NOT NULL`

Table G-4 (Cont.) Operators Supported by Data Types in Query (q) String Parameter

REST API Data Type	Supported Operator

time	• = (Equal to)
	`.../Employee?q=HireTime = '08:30:40'`
	• <> (Not equal to)
	`.../Employee?q=HireTime <> '08:30:40'`
	• < (Less than)
	`.../Employee?q=HireTime < '08:30:40'`
	• <= (Less than or equal to)
	`.../Employee?q=HireTime <= '08:30:40'`
	• > (Greater than)
	`.../Employee?q=HireTime > '08:30:40'`
	• >= (Greater than or equal to)
	`.../Employee?q=HireTime >= '08:30:40'`
	• `BETWEEN` (Between)
	`.../Employee?q=HireTime BETWEEN '04:30:00' AND '08:30:40'`
	• `NOT BETWEEN` (Not between)
	`.../Employee?q=HireTime NOT BETWEEN '04:30:00' AND '08:30:40'`
	• `IS NULL` (Is null)
	`.../Employee?q=HireTime IS NULL`
	• `NOT NULL` (Not null)
	`.../Employee?q=HireTime NOT NULL`

Table G-4 (Cont.) Operators Supported by Data Types in Query (q) String Parameter

REST API Data Type	Supported Operator

datetime	• = (Equal to)
	`.../Employee?q=HireDateTime = '1999-01-01T08:30:40Z'`
	• <> (Not equal to)
	`.../Employee?q=HireDateTime <> ’1999-01-01T08:30:40Z’`
	• < (Less than)
	`.../Employee?q=HireDateTime < ’1999-01-01T08:30:40Z’`
	• <= (Less than or equal to)
	`.../Employee?q=HireDateTime <= ’1999-01-01T08:30:40Z’`
	• > (Greater than)
	`.../Employee?q=HireDateTime > ’1999-01-01T08:30:40Z’`
	• >= (Greater than or equal to)
	`.../Employee?q=HireDateTime >= ’1999-01-01T08:30:40Z’`
	• `BETWEEN` (Between)
	`.../Employee?q=HireDateTime BETWEEN ’1999-01-01T08:30:40Z’ AND ’1999-12-01T08:30:40Z’`
	• `NOT BETWEEN` (Not between)
	`.../Employee?q=HireDateTime NOT BETWEEN ’1999-01-01T08:30:40Z’ AND ’1999-12-01T08:30:40Z’`
	• `IS NULL` (Is null)
	`.../Employee?q=HireDateTime IS NULL`
	• `NOT NULL` (Not null)
	`.../Employee?q=HireDateTime NOT NULL`

Table G-4 (Cont.) Operators Supported by Data Types in Query (q) String Parameter

REST API Data Type	Supported Operator

boolean	• `= 'true'` (true)
	`.../Employee?q=Active = 'true'`
	• `= 'false'` (false)
	`.../Employee?q=Active = 'false'`
	• `<> 'true'` (false)
	`.../Employee?q=Active <> 'true'`
	• `<> 'false'` (true)
	`.../Employee?q=Active <> 'false'`
	• `= 'Y'` (true)
	`.../Employee?q=Active = 'Y'`
	• `= 'N'` (false)
	`.../Employee?q=Active = 'N'`
	• `= true` (true)
	`.../Employee?q=Active = true`
	• `= false` (false)
	`.../Employee?q=Active = false`

Media Types Supported

- **Request**
  - None
- **Response**
  - `application/vnd.oracle.adf.resourcecollection+json`: When retrieving a resource collection.
  - `application/vnd.oracle.adf.resourceitem+json`: When retrieving a resource item.
  - `application/vnd.oracle.adf.description+json`: When describing a resource.

Describe Topics

- Retrieving the Resource Catalog Describe
- Retrieving a Resource Describe

Task Topics

- Fetching a Business Object
- Fetching a Business Object with a Subset of Items
- Fetching a Business Object Item
- Paging a Business Object
- Sorting a Business Object
- Fetching Data Only for a Business Object

- Filtering a Business Object with a Query Parameter
- Returning the Estimated Count of Business Object Items
- Checking for Data Consistency When Retrieving Business Object Items

REST API Framework Version Topics

- Working with REST API Framework Versions

POST Method Endpoints

REST APIs support the following tasks using a POST method with the URL as shown.

- Create a resource item.

  `<base_path>/{resourceCollectionPath}`

- Create a parent resource item and create the nested child resource collection in one roundtrip.

  `<base_path>/{resourceCollectionPath}`

- Update or create a resource item resource item in an existing resource using the `Upsert-Mode` header.

  `<base_path>/{resourceCollectionPath}`

- Execute an action on a resource collection or resource item.

  `<base_path>/{resourceCollectionPath}|{resourceItemPath}`

- Execute an a batch request.

  `<base_path>/{version}`

Request Parameters

- none

Media Types Supported

- **Request**
  - `application/vnd.oracle.adf.resourceitem+json`: When creating a resource item.
  - `application/vnd.oracle.adf.resourceitem+json`: When updating or creating a resource item using Upsert.
  - `application/vnd.oracle.adf.action+json`: When executing an action.
  - `application/vnd.oracle.adf.batch+json`: When executing a batch request.
- **Response**
  - `application/vnd.oracle.adf.resourceitem+json`: When executing an action or creating a resource item.
  - `application/vnd.oracle.adf.resourceitem+json`: When updating or creating a resource item using Upsert.
  - `application/vnd.oracle.adf.actionresult+json`: When executing an action that has an object to return.

  - `application/vnd.oracle.adf.batch+json`: When executing an batch request.

Task Topics

- Creating a Business Object Item
- Creating an Item of a Child Business Object
- Updating or Creating a Business Object Item (Upsert)
- Making Batch Requests

PATCH Method Endpoints

REST APIs support the following operation using a PATCH method with the URL as shown.

- Updating a resource item.

  `<base_path>/{resourceItemPath}`

Request Parameters

- none

Media Types Supported

- **Request**
  - `application/vnd.oracle.adf.resourceitem+json`: The resource item to be updated.
- **Response**
  - `application/vnd.oracle.adf.resourceitem+json`: The updated resource item.

Tasks Topics

- Updating a Business Object Item

DELETE Method Endpoints

REST APIs support the following operation using a DELETE method with the URL as shown.

- Deleting a resource item.

  `<base_path>/{resourceItemPath}`

Request Parameters

- none

Media Types Supported

- **Request**
  - none
- **Response**
  - none

Task Topics

- Deleting a Business Object Item