HTTP REST

Using standard HTTP requests, this API allows you to retrieve information about the datastore classes in your project, manipulate data, log into your web application, and much more. This manual is organized in three categories:

- Authenticating Users
- General Information
- Manipulating Data

REST Requests

With each REST request, the server returns the status and a response (with or without an error).

Authenticating Users

Once you have set up users and groups in your solution's directory, you will need to have users log into the project to access and manipulate data.

You can log in a user to your application by passing the user's name and password to **$directory/login**. Once logged in, you can retrieve the user's name by using **$directory/currentUser** and can find out if he/she belongs to a specific group by using **$directory/currentUserBelongsTo**. To log out the current user, call **$directory/logout**.

For more information about your solution's directory, refer to the **Directory** section in the **Solution Manager** chapter.

$directory/currentUser

Description

By calling **$directory/currentUser** after a user has logged in, you can retrieve the following information:

Property	Type	Description
userName	String	Username used to log into the application.
fullName	String	Full name of the user.
ID	String	UUID referencing the user.

Example

If you call **$directory/currentUser** as shown below once a user has logged in:

```
http://127.0.0.1:8081/rest/$directory/currentUser
```

The following result is returned:

```
{
    "result": {
        "userName": "jsmith",
        "fullName": "John Smith",
        "ID": "12F169764253481E89F0E4EA8C1D791A"
    }
}
```

If no user has been logged in, the result is:

```
{
    "result": null
}
```

$directory/login

Description

Use **$directory/login** to login a user into your web application through REST. The two parameters to login a user are *username* and *password*.

In a GET, you pass the *username* and *password* as shown below:

```
http://127.0.0.1:8081/rest/$directory/login(jsmith,johnny1)
```

In a POST, you pass an array containing two elements, *username* and *password*:

```
["jsmith","johnny1"]
```

If the login was successful, the result will be:

```
{
    "result": true
}
```

Otherwise, the response will be:

```
{
     "result": false
}
```

$directory/logout

Description

To log out the current user from your application, use **$directory/logout**.

If the logout was successful, the result will be:

```
{
    "result": true
}
```

Otherwise, the response will be:

```
{
     "result": false
}
```

$directory/currentUserBelongsTo

Description

To find out if the currently logged in user belongs to a specific group, use **$directory/currentUserBelongsTo**. You can pass either the group ID (which is the group's UUID reference number) or its name as defined in the solution's directory (see **Directory** in the **Solution Manager**).

To obtain the internal ID that Wakanda generated for your group, you can open your solution's Directory with a Text editor. You will see an entry similar to this one for each group:

```
<group ID="88BAF858143D4B13B26AF48C7A5A7A68" name="Sales">
```

Example

If we want to check to see if the current user is a member of the Sales group, we could write the following:

```
http://127.0.0.1:8081/rest/$directory/currentUserBelongsTo(88BAF858143D4B13B26AF48C7A5A7A
```

Or the following:

```
http://127.0.0.1:8081/rest/$directory/currentUserBelongsTo("Sales")
```

If the current user is in the group, the response will be:

```
{
    "result": true
}
```

Otherwise, it will return:

```
{
     "result": false
}
```

General Information

The parameters in this section allow you to retrieve information about one or all of the datastore classes in your project's active model as well as the entity sets currently stored in Wakanda Server's cache.

$info

Description

When you call this request for your project, you retrieve information in the following properties:

Property	Type	Description
cacheSize	Number	Wakanda Server's cache size.
usedCache	Number	How much of Wakanda Server's cache has been used.
entitySetCount	Number	Number of entity sets.
entitySet	Array	An array in which each object contains information about each entity set.

For each entity set currently stored in Wakanda Server's cache, the following information is returned:

Property	Type	Description
id	String	A UUID that references the entity set.
tableName	String	Name of the datastore class.
selectionSize	Number	Number of entities in the entity set.
sorted	Boolean	Returns true if the set was sorted (using **$orderby**) or false if it's not sorted.
refreshed	Date	When the entity set was created or the last time it was used.
expires	Date	When the entity set will expire (this date/time changes each time when the entity set is refreshed). The difference between refreshed and expires is the timeout for an entity set. This value is either two hours by default or what you defined using **$timeout**.

For information about how to create an entity set, refer to **$method=entityset**. If you want to remove the entity set from Wakanda Server's cache, use **$method=release**.

***Note**: Wakanda also creates its own entity sets for optimization purposes, so the ones you create with **$method=entityset** are not the only ones returned.*

Example

By making the following call:

```
http://127.0.0.1:8081/rest/$info
```

Here is an example of the information returned regarding our project's entity sets that are currently being stored in Wakanda Server's cache:

```
{
     cacheSize: 209715200,
     usedCache: 3136000,
     entitySetCount: 4,
     entitySet: [
         {
             id: "1418741678864021B56F8C6D77F2FC06",
             tableName: "Company",
             selectionSize: 1,
             sorted: false,
             refreshed: "2011-11-18T10:30:30Z",
             expires: "2011-11-18T10:35:30Z"
         },
```

```
            {
                id: "CAD79E5BF339462E85DA613754C05CC0",
                tableName: "People",
                selectionSize: 49,
                sorted: true,
                refreshed: "2011-11-18T10:28:43Z",
                expires: "2011-11-18T10:38:43Z"
            },
            {
                id: "F4514C59D6B642099764C15D2BF51624",
                tableName: "People",
                selectionSize: 37,
                sorted: false,
                refreshed: "2011-11-18T10:24:24Z",
                expires: "2011-11-18T12:24:24Z"
            }
        ],
        ProgressInfo: [
            {
                UserInfo: "flushProgressIndicator",
                sessions: 0,
                percent: 0
            },
            {
                UserInfo: "indexProgressIndicator",
                sessions: 0,
                percent: 0
            }
        ]
    }
```

***Note**: The progress indicator information listed after the entity sets is used internally by Wakanda.*

$catalog

Description

When you call **$catalog**, a list of the datastore classes is returned along with two URIs for each datastore class in your project's active model.

Only the datastore classes with the scope of **Public** are shown in this list for your project's active model. For more information on the **Scope** of a datastore class, please refer to the **Datastore Class Properties** section in the **Datastore Model Designer** chapter.

Here is a description of the properties returned for each datastore class in your project's active model:

Property	Type	Description
name	String	Name of the datastore class.
uri	String	A URI allowing you to obtain information about the datastore class and its attributes.
dataURI	String	A URI that allows you to view the data in the datastore class.

Example

Here is an example of what can be returned when you call **$catalog**:

```
{
    dataClasses: [
        {
            name: "Company",
            uri: "http://127.0.0.1:8081/rest/$catalog/Company",
            dataURI: "http://127.0.0.1:8081/rest/Company"
        },
        {
            name: "Employee",
            uri: "http://127.0.0.1:8081/rest/$catalog/Employee",
            dataURI: "http://127.0.0.1:8081/rest/Employee"
        }
```

```
    ]
}
```

$catalog/{datastoreClass}

Description

Calling **$catalog/{datastoreClass}** for a specific datastore class will return the following information about the datastore class and the attributes it contains. If you want to retrieve this information for all the datastore classes in your project's active model, use **$catalog/$all**.

The information you retrieve concerns the following:

- Datastore class
- Attribute(s)
- Primary key

Example

If you make the following request:

```
http://127.0.0.1:8081/rest/$catalog/Employee
```

The following example indicates the information that can be returned for the datastore class:

```
{
    name: "Employee",
    className: "Employee",
    collectionName: "EmployeeCollection",
    scope: "public",
    dataURI: "http://127.0.0.1:8081/rest/Employee",
    defaultTopSize: 20,
    extraProperties: {
        panelColor: "#76923C",
        __CDATA: "\n\n\t\t\n",
        panel: {
            isOpen: "true",
            pathVisible: "true",
            __CDATA: "\n\n\t\t\t\n",
            position: {
                X: "394",
                Y: "42"
            }
        }
    },
    attributes: [
        {
            name: "ID",
            kind: "storage",
            scope: "public",
            indexed: true,
            type: "long",
            identifying: true
        },
        {
            name: "firstName",
            kind: "storage",
            scope: "public",
            type: "string"
        },
        {
            name: "lastName",
            kind: "storage",
            scope: "public",
            type: "string"
        },
        {
            name: "fullName",
            kind: "calculated",
            scope: "public",
```

```
            type: "string",
            readOnly: true
        },
        {
            name: "salary",
            kind: "storage",
            scope: "public",
            type: "number",
            defaultFormat: {
                format: "$###,###.00"
            }
        },
        {
            name: "photo",
            kind: "storage",
            scope: "public",
            type: "image"
        },
        {
            name: "employer",
            kind: "relatedEntity",
            scope: "public",
            type: "Company",
            path: "Company"
        },
        {
            name: "employerName",
            kind: "alias",
            scope: "public",
            type: "string",
            path: "employer.name",
            readOnly: true
        },
        {
            name: "description",
            kind: "storage",
            scope: "public",
            type: "string",
            multiLine: true
        },
    ],
     key: [
        {
            name: "ID"
        }
     ]
}
```

$catalog/$all

Description

Calling **$catalog/$all** allows you to receive detailed information about the attributes in each of the datastore classes in your project's active model. Remember that the scope for the datastore classes and their attributes must be **Public** for any information to be returned.

For more information about what is returned for each datastore class and its attributes, refer to **$catalog/{datastoreClass}**.

Manipulating Data

The structure for a REST request is as follows:

URI	Resource	{Subresource}	Querystring
http://{*servername*}:{*port*}/rest/	**{datastoreClass}**/	**{attribute1, attribute2, ...}**/	
	{datastoreClass} ({key})/	**{attribute1, attribute2, ...}**/	
	{datastoreClass}/	**{attribute1, attribute2, ...}**/	{*method*}
			$entityset/{entitySetID}
			?**$filter**

While all REST requests must contain the URI and Resource parameters, the Subresource (which filters the data returned) is optional.

As with all URIs, the first parameter is delimited by a "?" and all subsequent parameters by a "&". For example:

```
http://127.0.0.1:8082/rest/Person/?$filter="lastName!=Jones"&$method=entityset&$timeout=6
```

***Note**: You can place all values in quotes in case of ambiguity. For example, in our above example, we could've put the value for the last name in quotes "Jones".*

The parameters in this chapter allow you to manipulate data in datastore classes in your Wakanda project. Besides retrieving data, you can also add, update, and delete entities in a datastore class.

If you want the data to be returned in an array instead of JSON, use the **$asArray** parameter.

{datastoreClass}

Description

When you call this parameter in your REST request, the first 100 entities are returned unless you have specified a value in the **Default Top Size** property (see **Datastore Class Properties**). You can also modify the number of entities by passing another value to **$top/$limit**.

Here is a description of the data returned:

Property	Type	Description
__entityModel	String	Name of the datastore class.
__COUNT	Number	Number of entities in the datastore class.
__SENT	Number	Number of entities sent by the REST request.This number can be the total number of entities if it is less than the value defined in the **Default Top Size** property (in the Properties for the datastore class) or **$top/$limit** or the value in **$top/$limit**.
__FIRST	Number	Entity number that the selection starts at. Either 0 by default or the value defined by **$skip**.
__ENTITIES	Array	This array of objects contains an object for each entity with all the Public attributes. All relational attributes are returned as objects with a URI to obtain information regarding the parent.

For each entity, there is a **__KEY** and a **__STAMP** property. The **__KEY** property contains the value of the primary key defined for the datastore class. The **__STAMP** is an internal stamp that is needed when you modify any of the values in the entity when using **$method=update**.

Example

When calling only **{datastoreClass}**:

```
http://127.0.0.1:8081/rest/Employee
```

The following data for the datastore class is returned:


```
{
    "__entityModel": "Company",
    "__COUNT": 250,
    "__SENT": 100,
    "__FIRST": 0,
    "__ENTITIES": [
        {
            "__KEY": "1",
            "__STAMP": 1,
            "ID": 1,
            "name": "Adobe",
            "address": null,
            "city": "San Jose",
            "country": "USA",
            "revenues": 500000,
            "staff": {
                "__deferred": {
                    "uri": "http://127.0.0.1:8081/rest/Company(1)/staff?$expand=staff"
                }
            }
        },
        {
            "__KEY": "2",
            "__STAMP": 1,
            "ID": 2,
            "name": "Apple",
            "address": null,
            "city": "Cupertino",
            "country": "USA",
            "revenues": 890000,
            "staff": {
                "__deferred": {
                    "uri": "http://127.0.0.1:8081/rest/Company(2)/staff?$expand=staff"
                }
            }
        },
        {
            "__KEY": "3",
            "__STAMP": 2,
            "ID": 3,
            "name": "4D",
            "address": null,
            "city": "Clichy",
            "country": "France",
            "revenues": 700000,
            "staff": {
                "__deferred": {
                    "uri": "http://127.0.0.1:8081/rest/Company(3)/staff?$expand=staff"
                }
            }
        },
        {
            "__KEY": "4",
            "__STAMP": 1,
            "ID": 4,
            "name": "Microsoft",
            "address": null,
            "city": "Seattle",
            "country": "USA",
            "revenues": 650000,
            "staff": {
                "__deferred": {
                    "uri": "http://127.0.0.1:8081/rest/Company(4)/staff?$expand=staff"
                }
```

```
            }
        }
.....//more entities here
    ]
}
```

{datastoreClass}({key})

Description

By passing the *datastoreClass* and a *key*, you can retrieve all the public information for that entity. The *key* is the value in the attribute defined as the Primary Key for your datastore class. For more information about defining a primary key, refer to the **Modifying the Primary Key** section in the **Datastore Model Designer**.

For more information about the data returned, refer to **{datastoreClass}**.

If you want to specify which attributes you want to return, define them using the following syntax **{attribute1, attribute2, ...}**. For example:

```
http://127.0.0.1:8081/rest/Company(1)/name,address
```

If you want to expand a relation attribute using **$expand**, you do so by specifying it as shown below:

```
http://127.0.0.1:8081/rest/Company(1)/name,address,staff?$expand=staff
```

Example

The following request returns all the public data in the Company datastore class whose *key* is 1.

```
http://127.0.0.1:8081/rest/Company(1)
```

The following data is returned:

```
{
    "__entityModel": "Company",
    "__KEY": "1",
    "__STAMP": 1,
    "ID": 1,
    "name": "Apple",
    "address": Infinite Loop,
    "city": "Cupertino",
    "country": "USA",
    "url": http://www.apple.com,
    "revenues": 500000,
    "staff": {
        "__deferred": {
            "uri": "http://127.0.0.1:8081/rest/Company(1)/staff?$expand=staff"
        }
    }
}
```

$top/$limit

Description

$top/$limit defines the limit of entities to return. By default, the number is limited to 100 or to the value specified in the **Default Top Size** property for your datastore class (see **Datastore Class Properties**). You can use either keyword: **$top** or **$limit**.

When used in conjunction with **$skip**, you can navigate through the entity collection returned by the REST request.

Example

In the following example, we request the next ten entities after the 20th entity:

```
http://127.0.0.1:8081/rest/Employee/$entityset/CB1BCC603DB0416D939B4ED379277F02?$skip=20&
```

$skip

Description

$skip defines which entity in the collection to start with. By default, the collection sent starts with the first entity. To start with the 10th entity in the collection, pass 10.

$skip is generally used in conjunction with **$top/$limit** to navigate through an entity collection.

Example

In the following example, we go to the 20th entity in our entity set:

```
http://127.0.0.1:8081/rest/Employee/$entityset/CB1BCC603DB0416D939B4ED379277F02?$skip=20
```

$orderby

Description

$orderby orders the entities returned by the REST request.

For each attribute, you specify the order as ASC (or asc), by default, for ascending order and DESC (desc) for descending order.

If you want to specify multiple attributes, you can delimit them with a comma, e.g., $orderby="lastName desc, firstName asc".

Example

In this example, we retrieve entities and sort them at the same time:

```
127.0.0.1:8081/rest/Employee/?$filter="salary!=0"&$orderby="salary, DESC,lastName ASC,fir
```

The example below sorts the entity set by lastName attribute in ascending order:

```
http://127.0.0.1:8081/rest/Employee/$entityset/CB1BCC603DB0416D939B4ED379277F02?$orderby=
```

$filter

Description

This parameter allows you to define the filter for your datastore class or method.

Simple Filter

A filter is composed of the following elements:

```
{attribute} {comparator =, !=, >, <...} {value}
```

For example: $filter="firstName=john AND salary>20000" where firstName and salary are attributes in the Employee datastore class.

Complex Filter

A more compex filter is composed of the following elements, which joins two queries:

```
{attribute} {comparator =, !=, >, <...} {value} {conjunction AND/OR/EXCEPT} {attribute} {
```

For example: $filter="firstName=john AND salary>20000" where firstName and salary are attributes in the Employee datastore class.

Example

In the following example, we look for all employees whose last name begins with a "J":

```
http://127.0.0.1:8081/rest/Employee?$filter="lastName begin j"
```

In this example, we search the Employee datastore class for all employees whose salary is greater than 20,000 and who do not work for a company named Acme:

```
http://127.0.0.1:8081/rest/Employee?$filter="salary>20000 AND employer.name!=acme"&$order
```

$queryplan

Description

$queryplan returns the query plan as it was passed to Wakanda Server.

Property	Type	Description
item	String	Actual query executed
subquery	Array	If there is a subquery, an additional object containing an item property (as the one above)

For more information about query plans, refer to the **queryPlan and queryPath** paragraph.

Example

If you pass the following query:

```
$filter="employer.name=acme AND lastName=Jones"&$queryplan=true
```

The following query plan would be returned:

```
__queryPlan: {
    And: [
        {
            item: "Join on Table : Company : People.employer = Company.ID",
            subquery: [
                {
                    item: "Company.name = acme"
                }
            ]
        },
        {
            item: "People.lastName = Jones"
        }
    ]
}
```

$querypath

Description

$querypath returns the query as it was executed by Wakanda Server. If, for example, a part of the query passed returns no entities, the rest of the query is not executed. The query requested is optimized as you can see in this **$querypath**.

For more information about query paths, refer to the **queryPlan and queryPath** paragraph.

In the steps array, there is an object with the following properties defining the query executed:

Property	Type	Description
description	String	Actual query executed or "AND" when there are multiple steps
time	Number	Number of milliseconds needed to execute the query
recordsfounds	Number	Number of records found
steps	Array	An array with an object defining the subsequent step of the query path

Example

If you pass the following query:

```
$filter="employer.name=acme AND lastName=Jones"&$querypath=true
```

And no entities were found, the following query path would be returned:

```
__queryPath: {

    steps: [
        {
            description: "AND",
            time: 0,
            recordsfounds: 0,
            steps: [
                {
```

```
                    description: "Join on Table : Company : People.employer = Company.ID"
                    time: 0,
                    recordsfounds: 0,
                    steps: [
                        {
                            steps: [
                                {
                                    description: "Company.name = acme",
                                    time: 0,
                                    recordsfounds: 0
                                }
                            ]
                        }
                    ]
                }
            ]
        }
    ]

}
```

If, on the other hand, the first query returns more than one entity, the second one will be executed. If we execute the following query:

```
$filter="employer.name=a@ AND lastName!=smith"&$querypath=true
```

The result will be as follows:

```
"__queryPath": {
    "steps": [
        {
            "description": "AND",
            "time": 1,
            "recordsfounds": 4,
            "steps": [
                {
                    "description": "Join on Table : Company : Employee.employer = Company
                    "time": 1,
                    "recordsfounds": 4,
                    "steps": [
                        {
                            "steps": [
                                {
                                    "description": "Company.name LIKE a@",
                                    "time": 0,
                                    "recordsfounds": 2
                                }
                            ]
                        }
                    ]
                },
                {
                    "description": "Employee.lastName # smith",
                    "time": 0,
                    "recordsfounds": 4
                }
            ]
        }
    ]
}
```

$expand

Description

When you have relation attributes in a datastore class, you can use **$expand** to retrieve all the attributes for the related entity or entities.

You can apply **$expand** to an entity (e.g., People(1)), an entity collection, or an entity set.

If you want to expand a relation attribute that is in an entity set, you can use the following syntax:

```
http://127.0.0.1:8081/rest/Employee/$entityset/99B09793950D414A864E6E1F03F0B293?$expand=e
```

If you want to view an image in its entirety, write the following:

```
http://127.0.0.1:8081/rest/Employee(1)/photo?$imageformat=best&$expand=photo
```

Example

If we pass the following REST request for our Company datastore class (which has a relation attribute "staff"):

```
http://127.0.0.1:8081/rest/Company(1)
```

The following data is returned:

```
{
    "__entityModel": "Company",
    "__KEY": "1",
    "__STAMP": 2,
    "ID": 1,
    "name": "Adobe",
    "address": null,
    "city": "San Jose",
    "country": "USA",
    "url": "http://www.adobe.com",
    "revenues": 500000,
    "staff": {
        "__deferred": {
            "uri": "http://127.0.0.1:8081/rest/Company(1)/staff?$expand=staff"
        }
    }
}
```

If we add the **$expand** to our request and specify the "staff" relation attribute:

```
http://127.0.0.1:8081/rest/Company(1)/?$expand=staff
```

The following data is returned:

```
{
    "__entityModel": "Company",
    "__KEY": "1",
    "__STAMP": 2,
    "ID": 1,
    "name": "Adobe",
    "address": null,
    "city": "San Jose",
    "country": "USA",
    "url": "http://www.adobe.com",
    "revenues": 500000,
    "staff": {
        "__COUNT": 2,
        "__SENT": 2,
        "__FIRST": 0,
        "__ENTITIES": [
            {
                "__KEY": "1",
                "__STAMP": 5,
                "ID": 1,
                "firstName": "John",
                "lastName": "Smith",
                "fullName": "John Smith",
                "telephone": "4085551111",
                "salary": 34000,
                "employer": {
                    "__deferred": {
                        "uri": "http://127.0.0.1:8081/rest/Company(1)",
                        "__KEY": "1"
                    }
                },
                "employerName": "Adobe"
            },
```

```
            {
                "__KEY": "2",
                "__STAMP": 2,
                "ID": 2,
                "firstName": "Paula",
                "lastName": "Miller",
                "fullName": "Paula Miller",
                "telephone": "4085559999",
                "salary": 36000,
                "employer": {
                    "__deferred": {
                        "uri": "http://127.0.0.1:8081/rest/Company(1)",
                        "__KEY": "1"
                    }
                },
                "employerName": "Adobe"
            }
        ]
    }
}
```

{attribute1, attribute2, ...}

Description

You can apply this filter to your entities in the following ways:

Method	Syntax	Example
Datastore class	{*datastoreClass*}/{*att1*,*att2*...}	/People/firstName,lastName
Collection of entities	*{datastoreClass}/{att1,att2...} /?$filter="{filter}"*	/People/firstName,lastName /?$filter="lastName='a@'"
Specific entity	{*datastoreClass*} ({*ID*})/{*att1*,*att2...*}	/People(1)/firstName,lastName
Entity set	*{datastoreClass}/{att1,att2...} /$entityset/{entitySetID}*	/People/firstName/$entityset /528BF90F10894915A4290158B4281E61
Datastore class method	{*datastoreClass*}/{*att1*,*att2*...} /{*method*}	/People/firstName,lastName /getHighSalaries

The attributes must be delimited by a comma, i.e., /Employee/firstName,lastName,salary.

If you want specific information from the related datastore class, you must first specify the relation attribute in the datastore class by using **$expand**. For example, you could write /Employee /firstName,lastName,employer.name,employer.city/?$expand="employer".

All types of attributes can be passed: storage, calculated, alias, inherited, or relational. For more information on attributes, refer to the **Attribute Categories** paragraph in the "**Datastore Model Designer**" chapter.

Example

Here are a few examples, showing you how to specify which attributes to return depending on the technique used to retrieve entities.

You can apply this technique to:

- Datastore classes (all or a collection of entities in a datastore class)
- Specific entities
- Datastore class methods
- Entity sets

Entity Set Example

Once you have created an entity set, you can filter the information in it by defining which attributes to display:

```
http://127.0.0.1:8081/rest/Employee/firstName,employer.name/$entityset/BDCD8AABE13144118A
```

{datastoreClass}/{method}

Description

Datastore class methods must be applied to either a **Class** or **Collection**, and must return either an entity collection or array.

The scope for a datastore class method must be **Public** for you to be able to call it in a REST request:

```
http://127.0.0.1:8082/rest/People/getHighSalaries
```

If you do not have the permissions to execute the datastore class method, you will receive the following error:

```
{
    "__ERROR": [
        {
            "message": "No permission to execute method getHighSalaries in dataClass Empl
            "componentSignature": "dbmg",
            "errCode": 1561
        }
    ]
}
```

Example

In the example below, we call our method, but also browse through the collection by returning the next ten entities from the sixth one:

```
http://127.0.0.1:8081/rest/Employee/getHighSalaries/lastName,employer?$expand=employer&$t
```

In the example below, the getCities datastore class method returns an array of cities:

```
http://127.0.0.1:8081/rest/Employee/getCities
```

Here is the result:

```
{
    "result": [
        "Paris",
        "Florence",
        "New York"
    ]
}
```

$method=update

Description

$method=update allows you to update and/or create one or more entities in a single **POST**. If you update and/or create one entity, it is done in an object with each property an attribute with its value, e.g., { lastName: "Smith" }. If you update and/or create multiple entities, you must create an array of objects.

To update an entity, you must pass the **__KEY** and **__STAMP** parameters in the object along with any modified attributes. If both of these parameters are missing, an entity will be added with the values in the object you send in the body of your **POST**.

All triggers, calculated attributes, and events are executed immediately when saving the entity to the server. The response contains all the data as it exists on the server.

Dates must be expressed in this format: YYYY-MM-DD*T*HH:MM:SS*Z* (e.g., "2010-10-05T23:00:00Z"). Booleans are either true or false.

You can also put these requests to create or update entities in a transaction by calling **$atomic/$atonce**. If any errors occur during data validation, none of the entities are saved. You can also use **$method=validate** to validate the entities before creating or updating them.

If a problem arises while adding or modifying an entity, an error will be returned to you with that information.

Example

To update a specific entity, you use the following URL:

```
http://127.0.0.1:8082/rest/Person/?$method=update
```

And POST the following in the request body:

```
{
    __KEY: "340",
    __STAMP: 2,
    firstName: "Pete",
    lastName: "Miller"
}
```

The firstName and lastName attributes in the entity indicated above will be modified leaving all other attributes (except calculated ones based on these attributes) unchanged.

If you want to create an entity, you can POST the attributes using this URL:

```
http://127.0.0.1:8082/rest/Person/?$method=update
```

With their values without including the **__KEY** and **__STAMP** parameters. For example:

```
{
    firstName: "John",
    lastName: "Smith"
}
```

When you add or modify an entity, it is returned to you with the attributes that were modified. For example, if you create the new employee above, the following will be returned:

```
{
    "__KEY": "622",
    "__STAMP": 1,
    "uri": "http://127.0.0.1:8081/rest/Employee(622)",
    "ID": 622,
    "firstName": "John",
    "firstName": "Smith",
    "fullName": "John Smith"
}
```

***Note**: The only reason the fullName attribute is returned is because it is a calculated attribute based on both firstName and lastName.*

You can also create and update multiple entities at the same time using the same URL above by passing multiple objects in an array to the POST:

```
[{
    "__KEY": "309",
    "__STAMP": 5,
    "ID": "309",
    "firstName": "Penelope",
    "lastName": "Miller"
}, {
    "firstName": "Ann",
    "lastName": "Jones"
}]
```

If, for example, the stamp is not correct, the following error is returned:

```
{
    "__ENTITIES": [
        {
            "__KEY": "309",
            "__STAMP": 1,
            "ID": 309,
            "firstName": "Betty",
            "lastName": "Smith",
            "fullName": "Betty Smith",
            "__ERROR": [
                {
                    "message": "Given stamp does not match current one for record# 308 of
                    "componentSignature": "dbmg",
                    "errCode": 1263
                },
                {
                    "message": "Cannot save record 308 in table Employee of database Widg
                    "componentSignature": "dbmg",
```

```
                    "errCode": 1046
                },
                {
                    "message": "The entity# 308 of the datastore class \"Employee\" cannc
                    "componentSignature": "dbmg",
                    "errCode": 1517
                }
            ]
        },
        {
            "__KEY": "612",
            "__STAMP": 4,
            "uri": "http://127.0.0.1:8081/rest/Employee(612)",
            "ID": 612,
            "firstName": "Ann",
            "lastName": "Jones",
            "fullName": "Ann Jones"
        }
    ]
}
```


If, for example, the user does not have the appropriate permissions to update an entity, the following error is returned:


```
{
    "__KEY": "2",
    "__STAMP": 4,
    "ID": 2,
    "firstName": "Paula",
    "lastName": "Miller",
    "fullName": "Paula Miller",
    "telephone": "408-555-5555",
    "salary": 56000,
    "employerName": "Adobe",
    "employer": {
        "__deferred": {
            "uri": "http://127.0.0.1:8081/rest/Company(1)",
            "__KEY": "1"
        }
    },
    "__ERROR": [
        {
            "message": "No permission to update for dataClass Employee",
            "componentSignature": "dbmg",
            "errCode": 1558
        },
        {
            "message": "The entity# 1 of the datastore class \"Employee\" cannot be saved
            "componentSignature": "dbmg",
            "errCode": 1517
        }
    ]
}
```


$method=delete

Description

With **$method=delete**, you can delete an entity or an entire entity collection. You can define the collection of entities by using, for example, **$filter** or specifying one directly using **{datastoreClass}({key})** (e.g., /Employee(22)).

You can also delete the entities in an entity set, by calling **$entityset/{entitySetID}**.

$method=delete can either be processed in a GET or POST.

If the request was successful, the following response is returned:


```
{
    "ok": true
}
```


Example

You can then write the following REST request to delete the entity whose key is 22:

```
http://127.0.0.1:8081/rest/Employee(22)/?$method=delete
```

You can also do a query as well using **$filter**:

```
http://127.0.0.1:8081/rest/Employee?$filter="ID=11"&$method=delete
```

You can also delete an entity set using **$entityset/{entitySetID}**:

```
http://127.0.0.1:8081/rest/Employee/$entityset/73F46BE3A0734EAA9A33CA8B14433570?$method=d
```

$method=entityset

Description

When you create a collection of entities in REST, you can also create an entity set that will be saved in Wakanda Server's cache. The entity set will have a reference number that you can pass to **$entityset/{entitySetID}** to access it. By default, it is valid for two hours; however, you can modify that amount of time by passing a value (in seconds) to **$timeout**.

If you have used **$savedfilter** and/or **$savedorderby** (in conjunction with **$filter** and/or **$orderby**) when you created your entity set, you can recreate it with the same reference ID even if it has been removed from Wakanda Server's cache.

$timeout

Description

To define a timeout for an entity set that you create using **$method=entityset**, pass the number of seconds to **$timeout**. For example, if you want to set the timeout to 20 minutes, pass 1200. By default, the timeout is two (2) hours.

Once the timeout has been defined, each time an entity set is called upon (by using **$method=entityset**), the timeout is recalculated based on the current time and the timeout.

If an entity set is removed and then recreated using **$method=entityset** along with **$savedfilter**, the new default timeout is 10 minutes regardless of the timeout you defined when calling **$timeout**.

Example

In our entity set that we're creating, we define the timeout to 20 minutes:

```
http://127.0.0.1:8081/rest/Employee/?$filter="salary!=0"&$method=entityset&$timeout=1200
```

$method=release

Description

You can release an entity set, which you created using **$method=entityset**, from Wakanda Server's cache, by executing the following:

```
http://127.0.0.1:8081/rest/Employee/$entityset/4C51204DD8184B65AC7D79F09A077F24?$method=r
```

If the request was successful, the following response is returned:

```
{
    "ok": true
}
```

If the entity set wasn't found, an error is returned:

```
{
    "__ERROR": [
        {
            "message": "Error code: 1802\nEntitySet \"4C51204DD8184B65AC7D79F09A077F24\"
            "componentSignature": "dbmg",
            "errCode": 1802
        }
    ]
}
```

$entityset/{entitySetID}

Description

After creating an entity set by using **$method=entityset**, you can then use it subsequently.

Because entity sets have a time limit on them (either by default or after calling **$timeout** with your own limit), you can call **$savedfilter** and **$savedorderby** to save the filter and order by statements when you create an entity set.

When you retrieve an existing entity set stored in Wakanda Server's cache, you can also apply any of the following to the entity set: **$expand**, **$filter**, **$orderby**, **$skip**, and **$top/$limit**.

Example

After you create an entity set, the entity set ID is returned along with the data. You call this ID in the following manner:

```
http://127.0.0.1:8081/rest/Employee/$entityset/9718A30BF61343C796345F3BE5B01CE7
```

$savedorderby

Description

When you create an entity set, you can save the sort order along with the filter that you used to create it as a measure of security. If the entity set that you created is removed from Wakanda Server's cache (due to the timeout, the server's need for space, or your removing it by calling **$method=release**).

You use **$savedorderby** to save the order you defined when creating your entity set, you then pass **$savedorderby** along with your call to retrieve the entity set each time.

If the entity set is no longer in Wakanda Server's cache, it will be recreated with a new default timeout of 10 minutes. If you have used both **$savedfilter** and **$savedorderby** in your call when creating an entity set and then you omit one of them, the new entity set, having the same reference number, will reflect that.

Example

You first call **$savedorderby** with the initial call to create an entity set:

```
http://127.0.0.1:8082/rest/People/?$filter="lastName!=''"&$savedfilter="lastName!=''"&$or
```

Then, when you access your entity set, you write the following (using both **$savedfilter** and **$savedorderby**) to ensure that the filter and its sort order always exists:

```
http://127.0.0.1:8082/rest/People/$entityset/AEA452C2668B4F6E98B6FD2A1ED4A5A8?$savedfilte
```

$savedfilter

Description

When you create an entity set, you can save the filter that you used to create it as a measure of security. If the entity set that you created is removed from Wakanda Server's cache (due to the timeout, the server's need for space, or your removing it by calling **$method=release**).

You use **$savedfilter** to save the filter you defined when creating your entity set and then pass **$savedfilter** along with your call to retrieve the entity set each time.

If the entity set is no longer in Wakanda Server's cache, it will be recreated with a new default timeout of 10 minutes. The entity set will be refreshed (certain entities might be included while others might be removed) since the last time it was created, if it no longer existed before recreating it.

If you have used both **$savedfilter** and **$savedorderby** in your call when creating an entity set and then you omit one of them, the new entity set, which will have the same reference number, will reflect that.

Example

In our example, we first call **$savedfilter** with the initial call to create an entity set as shown below:

```
http://127.0.0.1:8082/rest/People/?$filter="employer.name=Apple"&$savedfilter="employer.n
```

Then, when you access your entity set, you write the following to ensure that the entity set is always valid:

```
http://127.0.0.1:8082/rest/People/$entityset/AEA452C2668B4F6E98B6FD2A1ED4A5A8?$savedfilte
```

$atomic/$atonce

Description

When you have multiple actions together, you can use **$atomic/$atonce** to make sure that none of the actions are completed if one of them fails. You can use either **$atomic** or **$atonce**.

Example

If we call the following REST request:

```
http://127.0.0.1:8081/rest/Employee?$method=update&$atomic=true
```

Along with this array of objects to POST in the request body:

```
[
{
    "__KEY": "1",
    "__STAMP": 5,
    "salary": 45000
},
{
    "__KEY": "2",
    "__STAMP": 10,
    "salary": 99000
}
]
```

We get the following error in the second entity and therefore the first entity is not saved either:

```
{
    "__ENTITIES": [
        {
            "__KEY": "1",
            "__STAMP": 5,
            "uri": "http://127.0.0.1:8081/rest/Employee(1)",
            "ID": 1,
            "firstName": "John",
            "lastName": "Smith",
            "fullName": "John Smith",
            "gender": false,
            "telephone": "4085551111",
            "salary": 45000,
            "employerName": "Adobe",
            "employer": {
                "__deferred": {
                    "uri": "http://127.0.0.1:8081/rest/Company(1)",
                    "__KEY": "1"
                }
            }
        },
        {
            "__KEY": "2",
            "__STAMP": 2,
            "ID": 2,
            "firstName": "Paula",
            "lastName": "Miller",
            "fullName": "Paula Miller",
            "telephone": "4085559999",
            "salary": 36000,
            "employerName": "Adobe",
            "employer": {
                "__deferred": {
                    "uri": "http://127.0.0.1:8081/rest/Company(1)",
                    "__KEY": "1"
                }
            },
            "__ERROR": [
                {
                    "message": "Value cannot be greater than 60000",
                    "componentSignature": "dbmg",
                    "errCode": 1569
```

```
                },
                {
                    "message": "Entity fails validation",
                    "componentSignature": "dbmg",
                    "errCode": 1570
                },
                {
                    "message": "The entity# 1 of the datastore class \"Employee\" cannot
                    "componentSignature": "dbmg",
                    "errCode": 1517
                }
            ]
        }
    ]
}
```

***Note**: Even though the salary for the first entity has a value of 45000, this value was not saved to the server and the timestamp (__STAMP) was not modified either. If we reload the entity, we will see the previous value.*

$asArray

Description

If you want to receive the response in an array, you just have to add **$asArray** to your REST request:

```
http://127.0.0.1:8082/rest/Company/?$filter="name!=Acme"&$top=3&$asArray=true
```

The REST request will be returned as an array:

```
[
    {
        __KEY: {
            ID: 1,
            __STAMP: 3
        },
        ID: 1,
        name: "Apple",
        revenues: 500000,
        staff: {
            __COUNT: 1
        },
        country: "US"
    },
    {
        __KEY: {
            ID: 2,
            __STAMP: 3
        },
        ID: 2,
        name: "4D",
        revenues: 300000,
        staff: {
            __COUNT: 2
        },
        country: "France"
    },
    {
        __KEY: {
            ID: 3,
            __STAMP: 3
        },
        ID: 3,
        name: "Microsoft",
        revenues: 400000,
        staff: {
            __COUNT: 0
        },
        country: "US"
    }
]
```

The same data in its default JSON format:


```
{
    __entityModel: "Company",
    __COUNT: 7,
    __SENT: 3,
    __FIRST: 0,
    __ENTITIES: [
        {
            __KEY: "1",
            __STAMP: 3,
            ID: 1,
            name: "Apple",
            revenues: 500000,
            staff: {
                __deferred: {
                    uri: "http://127.0.0.1:8082/rest/Company(1)/staff?$expand=staff"
                }
            },
            country: "US"
        },
        {
            __KEY: "2",
            __STAMP: 3,
            ID: 2,
            name: "4D",
            revenues: 300000,
            staff: {
                __deferred: {
                    uri: "http://127.0.0.1:8082/rest/Company(2)/staff?$expand=staff"
                }
            },
            country: "France"
        },
        {
            __KEY: "3",
            __STAMP: 3,
            ID: 3,
            name: "Microsoft",
            revenues: 400000,
            staff: {
                __deferred: {
                    uri: "http://127.0.0.1:8082/rest/Company(3)/staff?$expand=staff"
                }
            },
            country: "US"
        }
    ]
}
```


$imageformat

Description

Define which format to use to display images. By default, the **best** format for the image will be chosen. You can, however, select one of the following formats:

Type	Description
GIF	GIF format
PNG	PNG format
JPEG	JPEG format
TIFF	TIFF format
best	Best format based on the image

Once you have defined the format, you must pass the image attribute to **$expand** to load the photo completely.

If there is no image to be loaded or the format doesn't allow the image to be loaded, the response will be empty.

Example

The following example defines the image format to JPEG regardless of the actual type of the photo:

```
http://127.0.0.1:8081/rest/Employee(1)/photo?$imageformat=jpeg&$expand=photo
```

$method=validate

Description

Before actually saving a new or modified entity with , you can first try to validate the actions first with **$method=validate**.

If the request is successful, the following response is returned:

```
{
    "ok": true
}
```

Otherwise, the errors that occurred are returned.

Example

In this example, we POST the following request to **$method=validate**:

```
http://127.0.0.1:8081/rest/Employee/?$method=validate
```

With the following data in the request body:

```
[{
    "__KEY": "1",
    "__STAMP": 8,
    "firstName": "Pete",
    "lastName": "Jones",
    "salary": 75000
}, {
    "firstName": "Betty",
    "lastName": "Miller",
}]
```

We get an error because our salary field must be inferior to 60000:

```
{
    "__ENTITIES": [
        {
            "__ERROR": [
                {
                    "message": "Value cannot be greater than 60000",
                    "componentSignature": "dbmg",
                    "errCode": 1569
                },
                {
                    "message": "Entity fails validation",
                    "componentSignature": "dbmg",
                    "errCode": 1570
                },
                {
                    "message": "The new entity of the datastore class \"Employee\" cannot
                    "componentSignature": "dbmg",
                    "errCode": 1534
                }
            ]
        }
    ]
}
```

$distinct

Description

$distinct allows you to return an array containing the distinct values for a query on a specific attribute for a datastore class. Only one attribute can be specified. Generally, the String type is best; however, you can also use it on any attribute type that could contain multiple values.

You can also use **$skip** and **$top/$limit** as well, if you'd like to navigate the selection before it's placed in an array.

Example

In our example below, we want to retrieve the distinct values for a company name starting with the letter "a":

```
http://127.0.0.1:8081/rest/Company/name?$filter="name=a@"&$distinct=true
```

Our response was the following:

```
[
    "Adobe",
    "Apple"
]
```