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

Authenticating Users

In this section, you will find a list of the parameters you can use to log in/out users, return the name of the currently logged in user, and if the current user belongs to a specific group.

General Information

In this section, you can use the parameters described here to obtain information about your project's active model as well as its datastore classes and the attributes in each one. You can also retrieve information about all the entity sets currently being stored in Wakanda Server's cache.

*Note: If you have multiple models for your project, the active one is the one that has **Active Model** as its role. For more information, refer to the **Contextual menus in the Solution Explorer** section in the **Solution Manager** chapter.*

Manipulating Data

This section lists the parameters you can use to manipulate data in a datastore class. You can query, sort, add, update, and delete entities To optimize the accessing of data, Wakanda also allows you to create and access entity sets, which are stored in Wakanda Server's cache.

REST Requests

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

Request Status

With each REST request, you get the status along with the response. Below are a few of the statuses that can arise:

Status	Description
0	Request not processed (server might not be started).
200 OK	Request processed without error.
401 Unauthorized	Permissions error (check user's permissions).
404 Not Found	The datastore class is not accessible via REST ("Public on Server" as scope) or the entity set doesn't exist.
500 Internal Server Error	Error processing the REST request.

Response

The response (in JSON format) varies depending on the request.

If an error arises, it will be sent along with the response from the server or it will be the response from the server.

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 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

Call $directory/currentUser to find out the current user of your application.

GET **/rest/$directory/currentUser**

Result:

```
{
    "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 by specifying its two parameters: *username* and *password*.

Example

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

POST **/rest/$directory/login**

POST data:

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

Result:

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**.

Example

You call $directory/logout to log the current user out of the application.

GET **/rest/$directory/logout**

Result:

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 **Groups** in the **Directory**).

If we want to check to see if the current user is a member of the Sales group, we must pass either *GroupID* or *GroupName* in the POST.

Example

Below is an example of how to pass either the *GroupID* or *GroupName* in the POST data.

POST **/rest/$directory/currentUserBelongsTo**

POST data:

```
["88BAF858143D4B13B26AF48C7A5A7A68"]
```

or

```
["Sales"]
```

Response:

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

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

Otherwise, it will return:

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

Wakanda Directory

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">
```

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.

For the **$catalog**, **$catalog/{datastoreClass}**, and **$catalog/$all** parameters, the user must be in a group that has **Describe** permissions. For more information, refer to .

$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.
ProgressInfo	Array	An array containing information about progress indicator information.
sessionInfo	Array	An array in which each object contains information about each user session.
jsContextInfo	Array	An array containing one object that returns the information about the JavaScript context pool.

entitySet

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.*

***IMPORTANT NOTE**: If your project is in , you must first log into the project as a user in the Admin group.*

sessionInfo

For each user session, the following information is returned in the *sessionInfo* array:

Property	Type	Description
sessionID	String	A UUID that references the session.
userID	String	A UUID that references the user who runs the session.
userName	String	The name of the user who runs the session.
lifeTime	Number	The lifetime of a user session in seconds (3600 by default).
expiration	Date	The current expiration date and time of the user session.

jsContextInfo

The object in the **jsContextInfo** array details the JavaScript context pool:

Property	Description
contextPoolSize	Maximum number of reusable contexts that can be stored in the JS pool (50 by default)
activeDebugger	Debugger state (false by default)
usedContextCount	Number of used contexts
usedContextMaxCount	Maximum number of contexts that have been used simultaneously
reusableContextCount	Number of reusable contexts (both used and unused)
unusedContextCount	Number of unused contexts
createdContextCount	Number of contexts created since the project was started
destroyedContextCount	Number of contexts destroyed since the project was started

Usage

Retrieve information about the entity sets currently stored in Wakanda Server's cache as well as user sessions:

GET **/rest/$info**

Result:

```
{
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
    }
],
sessionInfo: [
    {
        sessionID: "6657ABBCEE7C3B4089C20D8995851E30",
        userID: "36713176D42DB045B01B8E650E8FA9C6",
        userName: "james",
        lifeTime: 3600,
        expiration: "2013-04-22T12:45:08Z"
    },
    {
        sessionID: "A85F253EDE90CA458940337BE2939F6F",
        userID: "00000000000000000000000000000000",
        userName: "default guest",
        lifeTime: 3600,
        expiration: "2013-04-23T10:30:25Z"
}
],
jsContextInfo: [
    {
        "contextPoolSize": 50,
        "activeDebugger": false,
        "usedContextCount": 1,
        "usedContextMaxCount": 1,
        "reusableContextCount": 1,
        "unusedContextCount": 0,
        "createdContextCount": 4,
        "destroyedContextCount": 3
    }
]
}
```


***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.

The user must have describe access to view this information. For more information, refer to the **Datastore Class Permissions** (Model) and **Datastore Class Permissions** (Datastore class) sections.
For more information, refer to **Permission Actions**.

Example

Retrieve the list of datastore classes from your project.

GET **/rest/$catalog**

Result:

```
{
    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

The user must have describe access to view this information. For more information, refer to the **Datastore Class Permissions** (Model) and **Datastore Class Permissions** (Datastore class) sections.
For more information, refer to **Permission Actions**.

Datastore Class

The following properties are returned for a datastore class:

Property	Type	Description
name	String	Name of the datastore class
collectionName	String	Collection name of the datastore class
scope	String	Scope for the datastore class (note that only datastore classes whose **Scope** is public are displayed)
dataURI	String	A URI to the data in the datastore class
defaultTopSize	Number	The value entered in the **Default Top Size** property for the datastore class (if one was entered)
extraProperties	Object	Information about the datastore class in the **Datastore Model Designer** (panel color, position, or if it is hidden or displayed)

Attribute(s)

Here are the properties for each attribute that are returned:

Property	Type	Description
name	String	Attribute name.
kind	String	Attribute type (storage, calculated, relatedEntity, and alias).
scope	String	Scope of the attribute (only those attributes whose scope is Public will appear).
indexed	String	If any **Index Kind** was selected, this property will return true. Otherwise, this property does not appear.
type	String	Attribute type (bool, blob, byte, date, duration, image, long, long64, number, string, uuid, or word) or the datastore class for a N->1 relation attribute.
minLength	Number	This property returns the value entered for the **Min Length** property, if one was entered.
maxLength	Number	This property returns the value entered for the **Max Length** property, if one was entered.
autoComplete	Boolean	This property returns True if the **Autocomplete** property was checked. Otherwise, this property does not appear.
identifying	Boolean	This property returns True if the **Identifying** property was checked. Otherwise, this property does not appear.
multiLine	Boolean	This property returns True if the **Multiline** property was checked. Otherwise, this property does not appear.
path	String	For an alias attribute, the type is a path (e.g., employer.name)
readOnly	Boolean	This property is True if the attribute is of type calculated or alias.
defaultFormat	Object	If you define a format for the attribute in the **Default Format** property, it will appear in the "format" property.

*Note: Only attributes whose scope is **Public** are returned in any REST request. In an extended datastore class, the attribute must also not be removed. For more information about removed attributes from an extended datastore class, refer to the chapter in the **Datastore Model Designer** chapter.*

Primary Key

The key object returns the **name** of the attribute defined as the **Primary Key** for the datastore class.

For more information on the structure of a datastore class and its attributes, please refer to the **Datastore Classes** and **Attributes** sections in the **Datastore Model Designer** chapter.

Example

You can retrieve the information regarding a specific datastore class.

GET **/rest/$catalog/Employee**

Result:


```
{
    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}**.

The user must have describe access to view this information. For more information, refer to the **Datastore Class Permissions** (Model) and **Datastore Class Permissions** (Datastore class) sections.
For more information, refer to **Permission Actions**.

Example

Retrieve information about all your project's datastore classes and their attributes.

GET **/rest/$catalog/$all**

Result:


```
{

    "dataClasses": [
        {
            "name": "Company",
            "className": "Company",
            "collectionName": "CompanyCollection",
            "scope": "public",
            "dataURI": "/rest/Company",
            "attributes": [
                {
                    "name": "ID",
                    "kind": "storage",
                    "scope": "public",
                    "indexed": true,
                    "type": "long",
                    "identifying": true
                },
                {
                    "name": "name",
                    "kind": "storage",
                    "scope": "public",
                    "type": "string"
                },
                {
                    "name": "revenues",
                    "kind": "storage",
                    "scope": "public",
                    "type": "number"
                },
                {
                    "name": "staff",
                    "kind": "relatedEntities",
                    "matchColumn": "employees,staff",
                    "scope": "public",
                    "type": "EmployeeCollection",
                    "reversePath": true,
                    "path": "employer"
                },
                {
                    "name": "url",
                    "kind": "storage",
                    "scope": "public",
                    "type": "string"
```

```
                }
            ],
            "key": [
                {
                    "name": "ID"
                }
            ]
        },
        {
            "name": "Employee",
            "className": "Employee",
            "collectionName": "EmployeeCollection",
            "scope": "public",
            "dataURI": "/rest/Employee",
            "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": "employer",
                    "kind": "relatedEntity",
                    "scope": "public",
                    "type": "Company",
                    "path": "Company"
                }
            ],
            "key": [
                {
                    "name": "ID"
                }
            ]
        }
    ]
}
```

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:

GET **/rest/Person/?$filter="lastName!=Jones"&$method=entityset&$timeout=600**

***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.

Accessing Data

All datastore classes, datastore class methods, and attributes having a scope of **Public** can be accessed through REST. For more information regarding scope, refer to **Datastore Classes**, **Datastore Class Methods**, and **Attributes** sections of the **Datastore Model Designer** chapter.

All datastore class, datastore class method, and attribute names are case-sensitive; however, the data for queries is not.

If you have created a directory for your Wakanda project, you must also have proper permissions to access the datastore classes, datastore class methods, and attributes. For example, if you don't have permission to update data in a datastore class, you will not be able to modify any of the entities, however, you will be able to add new entities. To log into your Wakanda application, refer to the parameters described in **Authenticating Users**. For more information about creating a directory for your Wakanda project, refer to the section in the **Solution Manager** chapter.

Adding, Modifying, and Deleting Entities

With the REST API, you can perform all the manipulations to data as you can in Wakanda.

To add and modify entities, you can call **$method=update**. Before saving data, you can also validate it beforehand by calling **$method=validate**. If you want to delete one or more entities, you can use **$method=delete**.

Besides retrieving one attribute in a datastore class using **{datastoreClass}({key})**, you can also write a method in your datastore class and call it to return a collection of entities (or an array) by using **{datastoreClass}/{method}**.

Before returning the collection, you can also sort it by using **$orderby** one one or more attributes (even relation attributes).

Navigating Data

Add the **$skip** (to define with which entity to start) and **$top/$limit** (to define how many entities to return) REST requests to your queries or entity sets to navigate the collection of entities.

Creating and Managing Entity Sets

An entity set is a collection of entities obtained through a REST request that is stored in Wakanda Server's cache. Using an entity set prevents you from continually querying your application for the same results. Accessing an entity set is much quicker and can improve the speed of your application.

To create an entity set, call **$method=entityset** in your REST request. As a measure of security, you can also use **$savedfilter** and/or **$savedorderby** when you call **$filter** and/or **$orderby** so that if ever the entity set timed out or was removed from the server, it can be quickly retrieved with the same ID as before.

By default, an entity set is stored for two hours; however, you can change the timeout by passing a new value to **$timeout**. The timeout is continually being reset to the value defined for its timeout (either the default one or the one you define) each time you use it.

If you want to remove an entity set from Wakanda Server's cache, you can use **$method=release**.

If you modify any of the entity's attributes in the entity set, the values will be updated. However, if you modify a value that was a part of the query executed to create the entity set, it will not be removed from the entity set even if it no longer fits the search criteria. Any entities you delete will, of course, no longer be a part of the entity set.

If the entity set no longer exists 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.

Calculating Data

By using **$compute**, you can compute the following items for a specific attribute in a datastore class:

Keyword	Description
$all	A JSON object that defines all the functions for the attribute (average, count, min, max, and sum for attributes of type Number and count, min, and max for attributes of type String
average	Get the average on a numerical attribute
count	Get the total number in the collection or datastore class (in both cases you must specify an attribute)
min	Get the minimum value on a numerical attribute or the lowest value in an attribute of type String
max	Get the maximum value on a numerical attribute or the highest value in an attribute of type String
sum	Get the sum on a numerical attribute

Datastore class and attribute permissions

To manipulate data using REST, the user must have the correct permissions not only for the datastore class, but also the individual attributes. For more information, refer to **Datastore Class Permissions** and **Attribute permissions**.

If the user does not have permission to read an attribute, there is no error, but the value will be null. If the user tries to update a value in an attribute without being in the group that has permission to update, an error is generated and the entire entity is not saved.

Working with Entity Sets

Using the **$entityset/{entitySetID}?$logicOperator... &$otherCollection**, you can combine two entity sets that you previously created. You can either combine the results in both, return only what is common between the two, or return what is not common between the two.

A new collection of entities is returned; however, you can also create a new entity set by calling **$method=entityset** at the end of the REST request.

Viewing an image attribute

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

GET **/rest/Employee(1)/photo?$imageformat=best&$expand=photo**

For more information about the image formats, refer to **$imageformat**.

Retrieving Date attributes

When accessing a Date, it normally is returned in this format:

```
2014-07-03T21:25:05Z
```

If the **Date only** property has been selected, it will be returned as:

```
3!7!2014
```

For more information about the Date only property, refer to **Properties for Date Attributes**.

Saving a BLOB attribute to disk

If you want to save a BLOB stored in your datastore class, you can write the following by also passing "true" to [#cmd id="900959"/]:

GET **/rest/Company(11)/blobAtt?$binary=true&$expand=blobAtt**

{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). 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**.

Usage

Return all the data for a specific datastore class.

GET **/rest/Employee**

Result:


```
{
    "__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:

GET **/rest/Company(1)/name,address**

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

GET **/rest/Company(1)/name,address,staff?$expand=staff**

Usage

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

GET **/rest/Company(1)**

Result:

```
{
    "__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:

GET **/rest/Employee/$entityset/CB1BCC603DB0416D939B4ED379277F02?$skip=20&$top=10**

$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:

GET **/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) for ascending order and DESC (desc) for descending order. By default, the data is sorted in ascending 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:

GET **/rest/Employee/?$filter="salary!=0"&$orderby="salary DESC,lastName ASC,firstName ASC"**

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

GET **/rest/Employee/$entityset/CB1BCC603DB0416D939B4ED379277F02?$orderby="lastName"**

Result:

```
{
    __entityModel: "Employee",
    __COUNT: 10,
    __SENT: 10,
    __FIRST: 0,
    __ENTITIES: [
        {
            __KEY: "1",
            __STAMP: 1,
            firstName: "John",
            lastName: "Smith",
            salary: 90000
        },
        {
            __KEY: "2",
            __STAMP: 2,
            firstName: "Susan",
            lastName: "O'Leary",
            salary: 80000
        },
// more entities
    ]
}
```

$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} {comparator =, !=, >, <...} {val
```

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

Attribute

If the attribute is in the same datastore class, you can just pass it directly (i.e., firstName). However, if you want to query another datastore class, you must include the relation attribute name plus the attribute name (i.e., employer.name). The attribute name is case-sensitive (e.g., firstName is not equal to FirstName).

You can also query attributes of type Object by using dot-notation. For example, if you have an attribute whose name is "objAttribute" with the following structure:

```
{
    prop1: "this is my first property",
    prop2: 9181,
    prop3: ["abc", "def", "ghi"]
}
```

You can search in the Object by writing the following:

GET **/rest/Person/?filter="objAttribute.prop2 == 9181"**

Comparator

The comparator must be one of the following values:

Comparator	Description
=	equals to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
begin	begins with

Value

The value can be in simple quotes and is not case-sensitive, e.g., "bé" is equal to "be".

You can also use the ***** sign to define keyword searches, i.e., $filter="firstName='be*'". You can also omit the quotes for the value you define, especially if it contains an apostrophe (same as single quote), i.e., $filter="lastName=O'Leary".

Conjunction

Use a conjunction to join multiple conditions into the query. You can use one of the following logical operators (pass either the name or the symbol):

Conjunction	Conjunction symbol	Description
AND	&	both query items must be true
OR	|	either of the query items must be true
EXCEPT	^	equivalent to AND NOT

Example

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

GET **/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:

GET **/rest/Employee?$filter="salary>20000 AND employer.name!=acme"&$orderby="lastName,firstName"**

In this example, we search the Person datastore class for all the people whose number property in the anotherobj attribute of type Object is greater than 50:

GET **/rest/Person/?filter="anotherobj.mynum > 50"**

Response:

```
{
    __entityModel: "Person",
    __COUNT: 1,
    __SENT: 1,
    __FIRST: 0,
    __ENTITIES: [
        {
            __KEY: "1",
            __STAMP: 2,
            ID: 1,
            firstName: "john",
            lastName: "smith",
            fullName: "john smith",
            anotherobj: {
                mytext: "this is a quick note",
                myarray: [
                    "aaa",
                    "bbb"
                ],
                mynum: 99
            }
        }
    ]
}
```


$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 paragraph.

Example

If you pass the following query:

GET **/rest/People/$filter="employer.name=acme AND lastName=Jones"&$queryplan=true**

Response:


```
__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 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 passed the following query:

GET **/rest/Employee/$filter="employer.name=acme AND lastName=Jones"&$querypath=true**

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

GET **/rest/$querypath**

Respose:

```
__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:

GET **/rest/Employee/$filter="employer.name=a* AND lastName!=smith"&$querypath=true**

If at least one entity was found, the following query path would be returned, if you write the following:

GET **/rest/$querypath**

Respose:

```
"__queryPath": {
    "steps": [
        {
            "description": "AND",
            "time": 1,
            "recordsfounds": 4,
            "steps": [
                {
                    "description": "Join on Table : Company : Employee.employer = Company.ID",
                    "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 wan to sort the data retrieved for the relation attribute, you can use **$method=subentityset**.

Displaying data from the relation attribute

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

GET **/rest/Employee/$entityset/99B09793950D414A864E6E1F03F0B293?$expand=employer**

employer is the relation attribute that links the Company and Employee datastore classes.

Viewing an image attribute

If you want to view an image attribute in its entirety, write the following:
GET **/rest/Employee(1)/photo?$imageformat=best&$expand=photo**
For more information about the image formats, refer to **$imageformat**.

Saving a BLOB attribute to disk

If you want to save a BLOB stored in your datastore class, you can write the following by also passing "true" to [#cmd id="900959"/]:
GET **/rest/Company(11)/blobAtt?$binary=true&$expand=blobAtt**

Example

If we pass the following REST request for our Company datastore class (which has a relation attribute "staff"):
GET **/rest/Company(1)**
Response:


```
{
    "__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 **$expand** to our request and specify the "staff" relation attribute:
GET **/rest/Company(1)/?$expand=staff**
Response:


```
{
    "__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*} or {*datastoreClass*}/{*method*}/{*att1*,*att2*...}	/People/firstName,lastName/getHighSalaries or /People/getHighSalaries/firstName,lastName

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 about attributes, refer to the **Attribute Categories** paragraph in the **Datastore Model Designer** chapter.

***Note**: You cannot define a property in an attribute of type Object, you can only specify the actual attribute.*

Usage

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

Datastore Class Example

The following requests returns only the first name and last name from the People datastore class (either the entire datastore class or a selection of entities based on the search defined in **$filter**).

GET **/rest/People/firstName,lastName/**

Result:


```
{
    __entityModel: "People",
    __COUNT: 4,
    __SENT: 4,
    __FIRST: 0,
    __ENTITIES: [
        {
            __KEY: "1",
            __STAMP: 1,
            firstName: "John",
            lastName: "Smith"
        },
        {
            __KEY: "2",
            __STAMP: 2,
            firstName: "Susan",
            lastName: "O'Leary"
        },
        {
            __KEY: "3",
            __STAMP: 2,
            firstName: "Pete",
            lastName: "Marley"
        },
        {
            __KEY: "4",
            __STAMP: 1,
            firstName: "Beth",
            lastName: "Adams"
        }
    ]
}
```


GET **/rest/People/firstName,lastName/?$filter="lastName='A*'"/**

Result:


```
{
    __entityModel: "People",
    __COUNT: 1,
    __SENT: 1,
    __FIRST: 0,
    __ENTITIES: [
        {
            __KEY: "4",
            __STAMP: 4,
            firstName: "Beth",
            lastName: "Adams"
        }
    ]
}
```


Entity Example

The following request returns only the first name and last name attributes from a specific entity in the People datastore class:

GET **/rest/People(3)/firstName,lastName/**

Result:


```
 {
     __entityModel: "People",
     __KEY: "3",
     __STAMP: 2,
     firstName: "Pete",
     lastName: "Marley"
 }
```


GET **/rest/People(3)/**

Result:


```
{
    __entityModel: "People",
    __KEY: "3",
    __STAMP: 2,
    ID: 3,
    firstName: "Pete",
    lastName: "Marley",
    salary: 30000,
    employer: {
        __deferred: {
            uri: "http://127.0.0.1:8081/rest/Company(3)",
            __KEY: "3"
        }
    },
    fullName: "Pete Marley",
    employerName: "microsoft"

}
```


Method Example

If you have a datastore class method, you can define which attributes to return as shown below before passing the datastore class method:

GET **/rest/People/firstName,lastName/getHighSalaries**

or

GET **/rest/People/getHighSalaries/firstName,lastName**

Entity Set Example

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

GET **/rest/People/firstName,employer.name/$entityset/BDCD8AABE13144118A4CF8641D5883F5?$expand=employer**

{datastoreClass}/{method}

Description

Datastore class methods must be applied to either a **Class** or **Collection**, and must return either an entity collection or array. When returning an array, however, you cannot define which attributes are returned.

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

GET **/rest/Employee/getHighSalaries**

or

GET **/rest/Employee/firstName/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 Employee",
```

```
            "componentSignature": "dbmg",
            "errCode": 1561
        }
    ]
}
```

Passing Parameters to a Method

You can also pass parameters to a method either in a **GET** or in a **POST**.

In a **GET**, you write the following:

GET **/rest/Employee/addEmployee(John,Smith)**

In a **POST**, you write the following :

POST **/rest**/Employee/addEmployee

POST data:

```
["John","Smith"]
```

Manipulating the Data Returned by a Method

You can define which attributes you want to return, by passing the following:

GET **/rest/Employee/firstName/getEmployees**

Or

GET **/rest/Employee/getEmployees/firstName**

You can also apply any of the following functions to a method: **$expand**, **$filter**, **$orderby**, **$skip**, and **$top/$limit**.

Usage

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

GET **/rest/Employee/lastName,employer?$expand=employer&$top=10&$skip=1/getHighSalaries**

or

GET **/rest/Employee/getHighSalaries/lastName,employer?$expand=employer&$top=10&$skip=1**

If you want to retrieve an attribute and an extended relation attribute, you can write the following REST request:

GET **/rest/Employee/firstName,employer/getHighSalaries?$expand=employer**

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

GET **/rest/Employee/getCities**

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 JS format: YYYY-MM-DDTHH:MM:SSZ (e.g., "2010-10-05T23:00:00Z"). If you have selected the **Date only** property for your Date attribute, the time zone and time (hour, minutes, and seconds) will be removed. In this case, you can also send the date in the format that it is returned to you dd!mm!yyyy (e.g., 05!10!2013).

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:

POST **/rest/Person/?$method=update**

POST data:

```
{
    __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:

POST **/rest/Person/?$method=update**

POST data:

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

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:

POST **/rest/Person/?$method=update**

POST data:

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

Response:

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.*

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 table Employee",
                    "componentSignature": "dbmg",
                    "errCode": 1263
                },
                {
                    "message": "Cannot save record 308 in table Employee of database Widgets",
                    "componentSignature": "dbmg",
                    "errCode": 1046
                },
                {
                    "message": "The entity# 308 of the datastore class \"Employee\" cannot be saved",
                    "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`.

Example

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

`GET` **/rest/Employee(22)/?$method=delete**

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

`GET` **/rest/Employee?$filter="ID=11"&$method=delete**

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

`GET` **/rest/Employee/$entityset/73F46BE3A0734EAA9A33CA8B14433570?$method=delete**

Response:


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


$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.

Creating an Entity Set

To create an entity set, which will be saved in Wakanda Server's cache for two hours, add **$method=entityset** at the end of your REST request:

`GET` **/rest/People/?$filter="ID>320"&$method=entityset**

You can create an entity set that will be stored in Wakanda Server's cache for only ten minutes by passing a new timeout to **$timeout**:

`GET` **/rest/People/?$filter="ID>320"&$method=entityset&$timeout=600**

You can also save the filter and order by, by passing true to **$savedfilter** and **$savedorderby**.

***Note**: **$skip** and **$top/$limit** are not taken into consideration when saving an entity set.*

After you create an entity set, the first element, __ENTITYSET, is added to the object returned and indicates the URI to use to access the entity set:


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


Accessing an Entity Set

To access the entity set, you must use **$entityset/{entitySetID}** with the following syntax:

`GET` **/rest/People/$entityset/0AF4679A5C394746BFEB68D2162A19FF**

Removing an Entity Set from Cache

To remove an entity set from Wakanda Server's cache you must use **$method=release**:

GET **/rest/People/$entityset/0AF4679A5C394746BFEB68D2162A19FF?$method=release**

Viewing the References to the Entity Sets

When you call **$info**, the following information appears:

```
{
    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
        }
    ]

}
```

$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:

GET **/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.

Example

Release an existing entity set:

GET **/rest/Employee/$entityset/4C51204DD8184B65AC7D79F09A077F24?$method=release**

Response:

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\" cannot be found\ncomponent:
            "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:

GET **/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:

GET **/rest/People/?$filter="lastName!=""&$savedfilter="lastName!=""&$orderby="salary"&$savedorderby="salary"&$method=entityset**

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:

GET **/rest/People/$entityset/AEA452C2668B4F6E98B6FD2A1ED4A5A8?$savedfilter="lastName!=""&$savedorderby="salary"**

$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:

GET **/rest/People/?$filter="employer.name=Apple"&$savedfilter="employer.name=Apple"&$method=entityset**

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

GET **/rest/People/$entityset/AEA452C2668B4F6E98B6FD2A1ED4A5A8?$savedfilter="employer.name=Apple"**

$atomic/$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

We call the following REST request in a transaction.

POST **/rest/Employee?$method=update&$atomic=true**

POST data:

```
[
{
    "__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 be saved",
                    "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.

Usage

Here is an example or how to receive the response in an array.

`GET` **/rest/Company/?$filter="name!=Acme"&$top=3&$asArray=true**

Response:


```
[
    {
        __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"
        }
    ]
}
```

$compute

Description

This parameter allows you to do calculations on your data.
You can use any of the following keywords:

Keyword	Description
$all	A JSON object that defines all the functions for the attribute (average, count, min, max, and sum for attributes of type Number and count, min, and max for attributes of type String
average	Get the average on a numerical attribute
count	Get the total number in the collection or datastore class (in both cases you must specify an attribute)
min	Get the minimum value on a numerical attribute or the lowest value in an attribute of type String
max	Get the maximum value on a numerical attribute or the highest value in an attribute of type String
sum	Get the sum on a numerical attribute

Example

If you want to get all the computations for an attribute of type Number, you can write:
GET **/rest/Employee/salary/?$compute=$all**
Response:

```
{
    "salary": {
        "count": 4,
        "sum": 335000,
        "average": 83750,
        "min": 70000,
        "max": 99000
    }
}
```

If you want to get all the computations for an attribute of type String, you can write:
GET **/rest/Employee/firstName/?$compute=$all**
Response:

```
{
    "salary": {
        "count": 4,
        "min": Anne,
        "max": Victor
    }
}
```

If you want to just get one calculation on an attribute, you can write the following:
GET **/rest/Employee/salary/?$compute=sum**
Response:

```
235000
```

$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:

GET **/rest/Employee(1)/photo?$imageformat=jpeg&$expand=photo**

$method=validate

Description

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

Example

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

POST **/rest/Employee/?$method=validate**

POST data:

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

Response:

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

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

Otherwise, you receive an error. In our case, we got 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 be saved",
                    "componentSignature": "dbmg",
                    "errCode": 1534
                }
            ]
        }
    ]
}
```

$distinct

Description

$distinct allows you to return an array containing the distinct values for a query on a specific attribute. Only one attribute in the datastore class 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":

GET **/rest/Company/name?$filter="name=a*"&$distinct=true**

Response:

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

$entityset/{entitySetID}?$logicOperator... &$otherCollection

Description

After creating an entity set (entity set #1) by using **$method=entityset**, you can then create another entity set by using the **$entityset/{entitySetID}?**

$logicOperator... &$otherCollection syntax, the **$logicOperator** property (whose values are shown below), and another entity set (entity set #2) defined by calling the **$otherCollection** property. The two entity sets must be in the same datastore class.

You can also create another entity set containing the results from this call by using the **$method=entityset** at the end of the REST request.

Here are the logical operators:

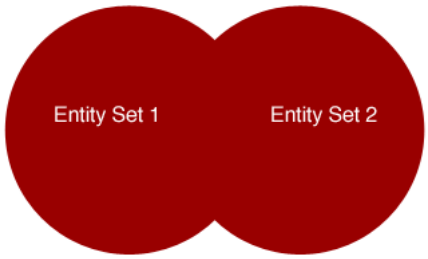
Operator	Description
AND	Returns the entities in common to both entity sets
OR	Returns the entities in both entity sets
EXCEPT	Returns the entities in entity set #1 minus those in entity set #2
INTERSECT	Returns either true or false if there is an intersection of the entities in both entity sets (meaning that least one entity is common in both entity sets)

Note: *The logical operators are not case-sensitive, so you can write "AND" or "and".*

Below is a representation of the logical operators based on two entity sets. The red section is what is returned.

AND

OR



EXCEPT

The syntax is as follows:

GET **/rest**/*datastoreClass*/$entityset/*entitySetID*?**$logicOperator**=*AND*&**$otherCollection**=*entitySetID*

Example

In the example below, we return the entities that are in both entity sets since we are using the **AND** logical operator:

GET **/rest/Employee/$entityset/9718A30BF61343C796345F3BE5B01CE7?$logicOperator=AND&$otherCollection=C05A0D887C664D4DA1B38366DD21629B**

If we want to know if the two entity sets intersect, we can write the following:

GET **/rest/Employee/$entityset/9718A30BF61343C796345F3BE5B01CE7?**
$logicOperator=intersect&$otherCollection=C05A0D887C664D4DA1B38366DD21629B

If there is an intersection, this query returns *true*. Otherwise, it returns *false*.

In the following example we create a new entity set that combines all the entities in both entity sets:

GET **/rest/Employee/$entityset/9718A30BF61343C796345F3BE5B01CE7?**
$logicOperator=OR&$otherCollection=C05A0D887C664D4DA1B38366DD21629B&$method=entityset

$method=subentityset

Description

$method=subentityset allows you to sort the data returned by the relation attribute defined in the REST request.

To sort the data, you use the $subOrderby property. For each attribute, you specify the order as ASC (or asc) for ascending order and DESC (desc) for descending order. By default, the data is sorted in ascending order

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

Usage

If you want to retrieve only the related entities for a specific entity, you can make the following REST request where staff is the relation attribute in the Company datastore class linked to the Employee datastore class:

GET **/rest/Company(1)/staff?$expand=staff&$method=subentityset&$subOrderby=lastName ASC**

Response:


```
{
    "__ENTITYSET": "/rest/Employee/$entityset/FF625844008E430B9862E5FD41C741AB",
    "__entityModel": "Employee",
    "__COUNT": 2,
    "__SENT": 2,
    "__FIRST": 0,
    "__ENTITIES": [
        {
            "__KEY": "4",
            "__STAMP": 1,
            "ID": 4,
            "firstName": "Linda",
            "lastName": "Jones",
            "birthday": "1970-10-05T14:23:00Z",
            "employer": {
                "__deferred": {
                    "uri": "/rest/Company(1)",
                    "__KEY": "1"
                }
            }
        },
        {
            "__KEY": "1",
            "__STAMP": 3,
            "ID": 1,
            "firstName": "John",
            "lastName": "Smith",
            "birthday": "1985-11-01T15:23:00Z",
            "employer": {
                "__deferred": {
                    "uri": "/rest/Company(1)",
                    "__KEY": "1"
                }
            }
        }
    ]

}
```


$binary

Description

$binary allows you to save the BLOB as a document. You must also use the **$expand** in conjunction with it.

When you make the following request:

GET **/rest/Company(11)/blobAtt?$binary=true&$expand=blobAtt**

You will be asked where to save the BLOB to disk: