Tools for Creating DeepSee Web Clients

Version 2018.1
2024-05-02

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Tools for Creating DeepSee Web Clients
Caché Version 2018.1 2024-05-02

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

About This Book

This book provides information on the DeepSee JavaScript and REST APIs, which you can use to create web clients for your DeepSee applications. This book contains the following sections:

- Introduction and Samples
- DeepSee JavaScript API
- DeepSee REST API

For a detailed outline, see the table of contents.

The other developer books for DeepSee are as follows:

- *Getting Started with DeepSee* briefly introduces DeepSee and the tools that it provides.
- *DeepSee Developer Tutorial* guides developers through the process of creating a sample that consists of a cube, subject areas, pivot tables, and dashboards.
- *DeepSee Implementation Guide* describes how to implement DeepSee, apart from creating the model.
- *Defining DeepSee Models* describes how to define the basic elements used in DeepSee queries: cubes and subject areas. It also describes how to define listing groups.
- *Advanced DeepSee Modeling Guide* describes how to use the more advanced and less common DeepSee modeling features: computed dimensions, unstructured data in cubes, compound cubes, cube relationships, term lists, quality measures, KPIs, plugins, and other special options.
- *Using MDX with DeepSee* introduces MDX and describes how to write MDX queries manually for use with DeepSee cubes.

For general information, see the *InterSystems Documentation Guide*.

1

Introduction and Samples

This chapter introduces the JavaScript and REST APIs for DeepSee. These APIs let you execute MDX queries and retrieve information about DeepSee model elements. This chapter discusses the following:

- How to create a web application that can use these APIs
- Introduction to the DeepSee JavaScript API
- Introduction to the DeepSee REST API
- Introduction to the samples

1.1 Creating a Web Application

In any scenario (whether you use the JavaScript API or you use the REST services directly), a Caché web application is responsible for handling the requests. You can use the system-defined web application (/api/deepsee) or you can create and use a different web application. The requirements for this web application are as follows:

- You must place your client file or files within the directory structure served by this web application. See "CSP Application Settings" in *Using Caché Server Pages (CSP)*.
- You must specify the **Dispatch Class** option, which specifies how this web application handles REST requests. For DeepSee REST requests, use one of the following:
  - %Api.DeepSee — Use this class if your client application must be able to connect to different namespaces. In this case, when you connect to a Caché server, you must specify the namespace to use.

    The system-defined web application (/api/deepsee) uses this dispatch class.
  - %DeepSee.REST.v1 — Use this class if the REST requests should be tied to a specific namespace (the namespace for the web application).

1.2 Introduction to the DeepSee JavaScript API

The DeepSee JavaScript API is provided by the file DeepSee.js, which is in the *install-dir*/CSP/broker directory. This JavaScript library enables you to interact with DeepSee from a client that is based on JavaScript. The functions in this

library are a wrapper for a REST-based API for DeepSee. (You can also use the REST API directly; it is introduced later in this chapter.)

To use this library:

1. Create a web application as described in the previous section.

   Or use the web application /api/deepsee, which is provided as part of the Caché installation.

2. In your JavaScript client code:

   a. Include the files DeepSee.js and zenCSLM.js. (The latter is a library used by Zen; it is in the same directory.)

   b. Create a DeepSee connection object. This contains information needed to connect to a Caché server.

   c. Create a DeepSee data controller object that uses the connection object.

      The data controller object enables you to interact with a DeepSee data source, which you specify either via an MDX query or via the name of a pivot table.

   d. Use the **runQuery()** method of the data controller. If the data source is an MDX query, DeepSee executes that query. If the data source is a pivot table, DeepSee executes the query defined by the pivot table.

   e. Invoke other methods of the data controller object to examine the query results, to drill down or drill through, and so on.

   The following subsections give the details.

The library DeepSee.js also provides utility functions that provide information about DeepSee model elements. Use these to obtain lists of available cubes, available measures in a cube, and so on.

1.2.1 Creating a DeepSee Connection

To create a DeepSee connection object, use code like the following:

```
connection = new DeepSeeConnection(username,password,host,application,namespace);
```

Where:

- *username* is a Caché username that can access the given host.
- *password* is the associated password.
- *host* is the server name for the machine on which Caché is running.
- *application* is the name of the Caché web application.
- *namespace* is the name of the namespace to access (if this information is needed). If the web application is tied to a namespace, this argument is not needed.

1.2.2 Creating and Using a DeepSee Data Controller

The data controller object enables you to interact with DeepSee data sources. The primary interaction is as follows:

- In a suitable part of the page logic (such as when the page is loaded or when a button is pressed), create a DeepSee data controller and execute a query.

  When you create a data controller, you specify one or two callback functions to be run when data is available; *finalCallback* is required, but *pendingCallback* is optional.

- When pending results are available, DeepSee calls the method specified by *pendingCallback*, if specified.

This method, which you write, uses the results that are available in the data controller object. The method typically draws page contents.

- When the query has completed, DeepSee calls the method specified by *finalCallback*.

  This method, which you write, uses the results that are available in the data controller object. The method typically draws page contents.

Any method that executes a query uses the system described here; see the subsections for details and examples. Other methods return data synchronously.

1.2.2.1 Creating a DeepSee Data Controller and Executing a Query

In a suitable part of the client code (such as within the page initialization logic), do the following:

1. Create a configuration object that has the following properties:
   - *connection* — Specifies the name of a DeepSee data connector object; see the previous section.
   - *widget* — Specifies the id of the HTML element on the page that will use the data controller
   - *type* — Specifies the type of data source; use either `'MDX'` or `'PIVOT'`
   - *initialMDX* — Specifies an MDX SELECT query; use this if type is `'MDX'`
   - *pivotName* — Specifies the logical name of a pivot table; use this if type is `'PIVOT'`
   - *showTotals* — Specifies whether to display totals. Specify either `true` or `false`
2. Create a data controller object with code like the following:

   ```
   var dc = new DeepSeeDataController(configuration,finalCallback,pendingCallback);
   ```

   Where *configuration* is the configuration object from the previous step, *finalCallback* is the name of a callback function on this page, and *pendingCallback* is the name of another callback function on this page. *finalCallback* is required, but *pendingCallback* is optional.
3. Call the **runQuery()** method of the data controller. Or run some other method that executes a query, such as **runDrillDown()** or **runListing()**.

For example:

```
function initializePage() {
   ...
       configuration.connection = new DeepSeeConnection(username,password,host,application,namespace);

        dc = new DeepSeeDataController(configuration,drawChart);
        dc.runQuery();
}
```

1.2.2.2 Using Data Returned by the Data Controller

The page must also implement the callback function or functions referred to in the previous step. These callbacks should update the page as needed, using data obtained from the data controller object.

In each case, the data controller object is passed to the function as the argument.

The following shows a partial example:

```
function drawChart(dataController) {

        var resultSet = dataController.getCurrentData();
...
        var chartDataPoint;
        var chartLabel;
        var chartData = [];

        for (var i = 1; i <= resultSet.getRowCount(); ++i) {
                for (var j = 1; j <= resultSet.getColumnCount(); ++j) {
                        chartDataPoint = resultSet.getOrdinalValue(i,j);
                        chartLabel = resultSet.getOrdinalLabel(2,i);
                     chartData[chartData.length] = { "country":chartLabel[0],"revenue":chartDataPoint};

                }
        }
...
```

The **getCurrentData()** method of the data controller returns another object, the result set object. That object provides methods for examining the results of the query. The example here shows some of them.

1.3 Introduction to the DeepSee REST API

Internally, the JavaScript API described earlier in this chapter uses the DeepSee REST API, which you can also use directly. To use the DeepSee REST API:

1. Create a web application as described in earlier in this chapter.

   Or use the web application /api/deepsee, which is provided as part of the Caché installation.

2. In your JavaScript client code, create and send HTTP requests to the desired target REST services.

   If you are using the dispatch class %Api.DeepSee, use a target URL of the following form:

   ```
   /baseURL/api/deepsee/v1/namespace/RESTcallname
   ```

   Where *baseURL* specifies the server, *namespace* is the target namespace, and *RESTcallname* is the actual rest call (for example, /Info/Cubes). For example:

   ```
   /mycompany/api/deepsee/v1/myapplication/Info/Cubes
   ```

   If you are using the dispatch class %DeepSee.REST.v1, use a target URL of the following form:

   ```
   /baseURL/api/deepsee/v1/RESTcallname
   ```

   For example:

   ```
   /mycompany/api/deepsee/v1/Info/Cubes
   ```

   Note: The client must accept JSON. The Accept header of the request must either specify `application/json` or not declare a format.

3. Examine the response objects and use as applicable.

1.3.1 Use of Slashes in Cube and KPI Names

It is relatively common to use slashes (/) in the logical names of DeepSee cubes and other items, because the slash character is the token that separates a folder name from a short item name. For example, a cube might have the logical name `RelatedCubes/Patients`

You *can* directly use these logical names unmodified in URL parameters (as well as in the request bodies). The applicable DeepSee REST services account for logical names that include slashes. The logic, however, requires you to follow a naming convention (depending on which REST services you plan to use). Specifically, do not have an item with a logical name that is the same as the name of a folder used by another logical name. For example, if you have an item called `mycubes/test/test1`, you should not have an item called `mycubes/test`.

The reason for this restriction is that when you use a REST service that uses another argument after the logical name, part of the name is interpreted as another argument if the first part of the name matches an existing item. Consider the following REST call:

```
http://localhost:57772/api/deepsee/v1/Info/FilterMembers/:mycubename/:filterspec
```

Here *mycubename* is the logical name of a cube and *filterspec* is the specification for a filter provided by that cube. Now consider this REST call with `mycubes/test/test1` as the name of the cube:

```
http://localhost:57772/api/deepsee/v1/Info/FilterMembers/:mycubes/test/test1/:filterspec
```

In order to interpret the slash characters, DeepSee first attempts to find a cube named `mycubes` and then attempts to find a cube named `mycubes/test`, and so on. When DeepSee finds the first item that matches the apparent name, the REST call uses that item, and the remainder of the string is interpreted as the next argument.

1.3.2 Notes on the Response Objects

For most of the REST calls, the response objects contain the property `Info`, which contains information about the request and response. This object contains the property `Error`, which equals one of the following:

- Null — This indicates that no error occurred.
- An object that contains the properties `ErrorCode` and `ErrorMessage` — This object contains details about the error that you can use to determine whether and how to proceed.

If no error occurred, the response object also contains the property `Result`, which is an object containing the requested values.

In general, your client code should first check the `Info.Error` property and then determine how to proceed.

For example, a response object might look like this (with white space added for readability):

```
{"Info":
   {"Error":
      {"ErrorCode":"5001","ErrorMessage":"ERROR #5001: Cannot find Subject Area: 'SampleCube'"}
   }
}
```

In contrast, if no error occurred, the `Info.Error` property is null and the `Result` contains the result that you requested. For example:

```
{"Info":
   {"Error":"",
    "BaseCube":"DemoMDX",
    "SkipCalculated":0},
 "Result":
   {"Measures":
     [
       {"name":"%COUNT","caption":"%COUNT","type":"integer","hidden":0,"factName":""},
       {"name":"Age","caption":"Age","type":"integer","hidden":0,"factName":"MxAge"}
     ...]
    }
}
```

1.4 Samples

DeepSee provides samples for the JavaScript library and for the REST API.

1.4.1 Samples That Use DeepSee.js

For samples that use the DeepSee JavaScript library (DeepSee.js), see the directory *install-dir*/CSP/samples. The sample HTML page testDataController.html demonstrates most of the methods of the JavaScript API. To use this sample, open a URL of the following form in your browser:

```
http://localhost:57772/CSP/samples/testDataController.html
```

Where *localhost:57772* is the server and port on which Caché is running.

The sample HTML page test3rdPartyCharts.html demonstrates how to use the API in combination with a third-party charting library. For this sample, use a URL of the following form:

```
http://localhost:57772/CSP/samples/test3rdPartyCharts.html
```

1.4.2 Sample That Uses REST API

For a sample that uses the REST API, open a URL of the following form in your browser:

```
http://localhost:57772/CSP/samples/DeepSee.RESTClient.cls
```

Where *localhost:57772* is the server and port on which Caché is running.

This displays the web page DeepSee.RESTClient, a class in the SAMPLES namespace. This class is a test page that enables you to exercise each REST call within the context of a Caché web application. You can use this page in two ways:

- You can specify **Application** as /api/deepsee and specify a namespace in **Namespace**. The requests are forwarded to the given namespace. (The SAMPLES namespace is useful because it contains multiple DeepSee models.)
- If you create a custom web application as described earlier in this chapter, specify **Application** as your application name.

  In this case, do not enter anything the **Namespace** field.

Below these options, the page has a table that lists all the DeepSee REST calls. To try a REST call:

1. Select the row that corresponds to the call.

   The lower left part of the page then displays details for that call.

2. If needed, enter values into the labeled input fields (such as **:cube**). These input fields represent values that are required as part of the URL.
3. If needed, type the request body into the larger box. Or select **Browse...** and upload a file that contains the request body.
4. Press **Submit**.
5. If you are prompted for a username and password, enter those and then press **Login**.

The page then displays the response body in the lower right area. Above the response, the page shows the actual URL that was used.

Tip: It is also useful to access the developer tools in your browser and to use them to see the request and response.

DeepSee.js

This reference section provides information on the JavaScript API for DeepSee. This API is provided by the file DeepSee.js.

DeepSeeConnector

Enables you to connect to DeepSee data sources.

Where This Object Is Available

This object is available in client-side JavaScript code, if that code includes DeepSee.js. See “Introduction to DeepSee.js.”

Creating This Object

To create a DeepSee data connector object, use code like the following:

```
var connection = new DeepSeeConnection(username,password,host,application,namespace);
```

Where:

- *username* is a Caché username that can access the given host.
- *password* is the associated password.
- *host* is the server name for the machine on which Caché is running.
- *application* is the name of the Caché web application.
- *namespace* is the name of the namespace to access.

Properties of This Object

This object provides the following properties:

- `username` is a Caché username that can access the given host.
- `password` is the associated password.
- `path` is the base URL for the web services.

Methods of This Object

A data connector object does not provide any methods.

DeepSeeDataController

Enables you to work with a DeepSee data source.

Where This Object Is Available

This object is available in client-side JavaScript code, if that code includes DeepSee.js. See "Introduction to DeepSee.js."

Creating This Object

To create a DeepSee data controller object, use code like the following:

```
var controller = new DeepSeeDataController(configuration,finalCallback,pendingCallback);
```

Where:

- *configuration* is an object that has the following properties:
  - *connection* — Is a DeepSee data connector object.
  - *widget* — Specifies the id of the HTML element on the page that will use the data controller
  - *type* — Specifies the type of data source; use either `'MDX'` or `'PIVOT'` (case-sensitive)
  - *initialMDX* — Specifies an MDX SELECT query; specify this property if *type* is `'MDX'`
  - *pivotName* — Specifies the logical name of a pivot table; specify this property if *type* is `'PIVOT'`
  - *showTotals* — Specifies whether to display totals; specify this property as `true` or `false`
- *finalCallback* is the name of a callback function on this page

  For **runQuery()** and other methods of this object, when the query is completed, DeepSee invokes this function.
- *pendingCallback* (optional) is the name of another callback function on this page.

  For **runQuery()** and other methods of this object, when pending results are available, DeepSee invokes this function, if this argument is supplied.

Methods of This Object

The DeepSee data controller object provides the following JavaScript methods:

applyFilter()

```
applyFilter(filterInfo)
```

Where *filterInfo* is an object that contains the *filterName* and *filterSpec* properties.

This method adds the given filter to the filters used by the data controller, reruns the query, and then invokes the callback functions associated with the data controller object.

This method has no return value.

attachTotals()

```
attachTotals(rowTotals,columnTotals,reattach)
```

Where:

- *rowTotals* is `true` or `false`, depending on whether you want to attach totals for the rows.

- *columnTotals* is `true` or `false`, depending on whether you want to attach totals for the columns.
- *reattach* is `true` or `false`, depending on whether you want to update the most recently saved state.

  If *reattach* is `true`, DeepSee updates the most recently saved state. If *reattach* is `false`, DeepSee adds a new state object to the stack.

This method attaches totals to the data controller object and then invokes the callback functions associated with the data controller object.

This method has no return value.

getCurrentData()

```
getCurrentData()
```

Returns a DeepSee result set object that contains the results from the query currently defined by the data controller object. See the reference for the DeepSeeResultSet object.

getCurrentQueryText()

```
getCurrentQueryText()
```

Returns the text of the query currently defined by the data controller object.

runDrillDown()

```
runDrillDown(axis, position)
```

Where:

- *axis* is the number of the axis on which you want to perform the drilldown action. Specify 1 for column or 2 for rows.
- *position* is the position (1–based) on that axis where you want to perform the drilldown action.

This method executes the given drilldown action, and then invokes the callback functions associated with the data controller object.

This method has no return value.

runListing()

```
runListing(startRow, startCol, endRow, endCol, listingName)
```

Where:

- *startRow* and *startCol* are the first row and column number of the results for which you want a detail listing. Specify 1 for the first row or column.
- *endRow* and *endCol* are the last row and column number for which you want a detail listing.
- *listingName* is the logical name of a detail listing.

This method executes the given detail listing for one or more cells of the results, and then invokes the callback functions associated with the data controller object.

This method has no return value.

runQuery()

```
runQuery()
```

Executes the query currently defined by the data controller object. When the query is pending or completed, DeepSee invokes the callback functions associated with the data controller object.

This method has no return value.

sortResults()

```
sortResults(axis, position, direction, sortType)
```

Where:

- *axis* is the number of the axis you want to sort. Specify 1 for column or 2 for rows.
- *position* is the position (1–based) on that axis where you want to sort.

  For example, to sort by the third column, specify *axis* as 1 and *position* as 3.
- *direction* is the direction in which to sort. Specify 1 for ascending sort or –1 or descending sort.
- *sortType* specifies how to sort. If this is `''` or `'numeric'` (case-insensitive), numeric sorting is used. Otherwise, string sorting is used.

This method sorts the results as requested and then invokes the callback functions associated with the data controller object.

This method has no return value.

undoLastAction()

```
undoLastAction()
```

Undoes the last change and invokes the callback functions associated with the data controller object.

This method has no return value.

DeepSeeResultSet

Enables you to work with the results of a DeepSee query.

Where This Object Is Available

This object is available in client-side JavaScript code, if that code includes DeepSee.js. See "Introduction to DeepSee.js."

Creating This Object

To create a result set, call the **getCurrentData()** method of the data controller object.

Or use code like the following:

```
var resultset = new DeepSeeResultSet(connection,widget,wait,timeout);
```

Where:

- *connection* — Is a DeepSee data connector object.
- *widget* — Specifies the id of the HTML element on the page that will use the data controller.
- *wait* — Boolean. If this parameter is false, the server should respond immediately. If this parameter is true, the server should not respond until the query has completed or timed out.
- *timeout* — Specifies how long (in seconds) the server should wait before returning final but incomplete query results.

Properties of This Object

The DeepSee result set object provides the following properties:

- `connection` — Is a DeepSee data connector object.
- `widget` — Specifies the id of the HTML element on the page that will use the data controller.
- `wait` — Boolean. If this property is false, the server should respond immediately. If this property is true, the server should not respond until the query has completed or timed out.
- `timeout` — Specifies how long (in seconds) the server should wait before returning final but incomplete query results.
- `data` — Contains the query results.
- `pollInterval` — Specifies how long (in milliseconds) to wait before asking the server for updates to pending results. The default is 1000.

Methods of This Object

The DeepSee result set object provides the following JavaScript methods:

getColumnCount()

```
getColumnCount()
```

Returns the number of columns in the result set.

getCubeName()

```
getCubeName()
```

Returns the name of the cube currently in use.

getErrorMessage()

```
getErrorMessage()
```

This method returns the text of the error message contained in the response object, if any.

getOrdinalLabel()

```
getOrdinalLabel(axis, position)
```

Where:

- *axis* is the number of the axis whose labels you want to obtain. Specify 1 for column or 2 for rows.
- *position* is the position on that axis whose labels you want to obtain. Specify 1 for the first position on the axis.

Returns an array of strings, corresponding to the labels at the given position on the given axis.

getOrdinalValue()

```
getOrdinalValue(rowNo,colNo,formatted)
```

Where:

- *rowNo* is the row number (1–based)
- *colNo* is the column number (1–based)

Returns the value in the given cell.

getQueryStatus()

```
getQueryStatus()
```

Returns a numeric value indicating whether the query has completed. This number is 100 if the query has completed. Otherwise, this number is less than 100.

getRowCount()

```
getRowCount()
```

Returns the number of rows in the result set.

isError()

```
isError()
```

This method returns `true` if the response object indicates that an error occurred. Returns `false` otherwise.

runMDXQuery()

```
runMDXQuery(mdx,finalCallback,pendingCallback,filters)
```

Where:

- *mdx* is an MDX SELECT query.
- *finalCallback* is the name of a callback function on this page.

- *pendingCallback* (optional) is the name of another callback function on this page.
- *filters* specifies additional filters to apply to the query.

This method executes the given MDX query. When the query is pending or completed, DeepSee invokes the given callback functions.

This method has no return value.

runMDXDrillQuery()

```
runMDXDrillQuery(mdx,finalCallback,pendingCallback,listing,fieldList,filters)
```

Where:

- *mdx* is an MDX SELECT query.
- *finalCallback* is the name of a callback function on this page.
- *pendingCallback* (optional) is the name of another callback function on this page.
- *listing* is the name of a detail listing.
- *fieldList* specifies a list of listing fields. Specify either *listing* or *fieldList*.
- *filters* specifies additional filters to apply to the query.

This method executes the given drillthrough query. When the query is pending or completed, DeepSee invokes the given callback functions.

This method has no return value.

runPivot()

```
runPivot(pivot,finalCallback,pendingCallback,filters)
```

Where:

- *pivot* is the logical name of pivot table.
- *finalCallback* is the name of a callback function on this page.
- *pendingCallback* (optional) is the name of another callback function on this page.
- *filters* specifies additional filters to apply to the query.

This method executes the MDX query defined by the given pivot table. When the query is pending or completed, DeepSee invokes the given callback functions.

This method has no return value.

DeepSeeUtils

Provides additional methods for working with DeepSee.

Where This Object Is Available

This object is available in client-side JavaScript code, if that code includes DeepSee.js. See "Introduction to DeepSee.js."

Creating This Object

When you include DeepSee.js in your client code, the *DeepSeeUtils* object is automatically available.

Methods of This Object

The *DeepSeeUtils* object provides the following methods:

getCubeList()

```
getCubeList(connection,finalFunc)
```

Where:

- *connection* — Is a DeepSee data connector object.
- *finalFunc* — Specifies the name of the function to execute when results are available

This method retrieves information about the available cubes. It calls the POST /Info/Cubes REST call, which is described in the next reference in this book. The response object from that call is available to the function specified by *finalFunc*.

getCubeListings()

```
getCubeListings(connection,finalFunc)
```

Where:

- *connection* — Is a DeepSee data connector object.
- *cubename* — Specifies the logical name of a DeepSee cube.
- *finalFunc* — Specifies the name of the function to execute when results are available

This method retrieves information about the listings available to the given cube. It calls the POST /Info/Listings/:cube REST call, which is described in the next reference in this book. The response object from that call is available to the function specified by *finalFunc*.

getDashboardList()

```
getDashboardList(connection,finalFunc)
```

This method retrieves information about the available dashboards. It calls the POST /Info/Dashboards REST call, which is described in the next reference in this book. The response object from that call is available to the function specified by *finalFunc*.

getErrorMessage()

```
getErrorMessage(data)
```

Where *data* is a response object returned by any of the DeepSee REST services, which are described in the next reference in this book.

This method returns the text of the error message contained in that object, if any.

getFiltersForDataSource()

```
getFiltersForDataSource(connection,cubename,finalcallback)
```

Where:

- *connection* — Is a DeepSee data connector object.
- *cubename* — Specifies the logical name of a DeepSee cube.
- *finalcallback* — Specifies the name of the function to execute when results are available

This method retrieves information about the available filters for the given cube. It calls the POST /Info/Filters/:data-source REST call, which is described in the next reference in this book. The response object from that call is available to the function specified by *finalFunc*.

getMembersForFilter()

```
getMembersForFilter(connection,cubeName,filterSpec,finalCallback)
```

Where:

- *connection* — Is a DeepSee data connector object.
- *cubename* — Specifies the logical name of a DeepSee cube.
- *filterSpec* — Specifies the filter whose members you want to retrieve.
- *finalcallback* — Specifies the name of the function to execute when results are available

This method retrieves information about members of the given filter. It calls the POST /Info/FilterMembers/:data-source/:filterSpec REST call, which is described in the next reference in this book. The response object from that call is available to the function specified by *finalFunc*.

getPivotList()

```
getPivotList(connection,finalFunc)
```

Where:

- *connection* — Is a DeepSee data connector object.
- *finalFunc* — Specifies the name of the function to execute when results are available

This method retrieves information about the available pivot tables. It calls the POST /Info/Pivots REST call, which is described in the next reference in this book. The response object from that call is available to the function specified by *finalFunc*.

getResultsAsArray()

```
getResultsAsArray(data)
```

Where *data* is a response object returned by any of the DeepSee REST services, which are described in the next reference in this book.

This method returns an array of results from that object as follows:

1. If the response object contains an `Result.Filters` array, the method returns that array.
2. Otherwise, if the response object contains an `Result.FilterMembers` array, the method returns that array.
3. Otherwise, if the response object contains an `Result.Listings` array, the method returns that array.

Otherwise the method returns nothing.

isError()

```
isError(data)
```

Where *data* is a response object returned by any of the DeepSee REST services, which are described in the next reference in this book.

This method returns `true` if the response object indicates that an error occurred. Returns `false` otherwise.

DeepSee REST API

This reference section provides information on the REST services for DeepSee. These services are provided by the class %DeepSee.REST.v1. See “Using the DeepSee REST API,” earlier in this book.

GET /Data/GetDSTIME

Retrieves the last ^OBJ.DSTIME timestamp that the server processed for a given cube.

URL Parameters

sourceClass	*Required.* Full name of the source class of the cube.

Request Body Details

This service ignores the request body.

Example Request

- *Request Method:*

  `GET`

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Data/GetDSTime/HoleFoods.Transation`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

Example Response

```
{
    "Status": "OK",
    "DispatchClass": "%DeepSee.REST.v1.InfoServer",
    "NameSpace": "SAMPLES"
}
```

GET /Data/TestConnection

Tests the connection to the server.

URL Parameters

None.

Request Body Details

This service ignores the request body.

Example Request

- *Request Method:*

  `GET`

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Data/TestConnection`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

Example Response

```
{
    "Status": "OK",
    "DispatchClass": "%DeepSee.REST.v1.DataServer",
    "NameSpace": "SAMPLES"
}
```

GET /Info/TestConnection

Tests the connection to the server.

URL Parameters

None.

Request Body Details

This service ignores the request body.

Example Request

- *Request Method:*

  `GET`

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Info/TestConnection`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

Example Response

```
{
    "Status": "OK",
    "DispatchClass": "%DeepSee.REST.v1.InfoServer",
    "NameSpace": "SAMPLES"
}
```

POST /Data/KPIExecute

Execute the query defined by a KPI.

URL Parameters

None. Note that a request body is required; see the next heading.

Request Body Details

This service uses the following properties of the request body:

`KPI`	*Required.* Logical name of the KPI.
`FILTERS`	Optional. Array of filter objects that specify how the KPI is filtered. Each filter object must provide the following properties: • `name` — a filter specification such as `[aged].[h1].[age group]` • `value` — logical name of a member of a filter, such as `&[0 to 29]`

Example Request

- *Request Method:*

  `POST`

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Data/KPIExecute`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

- *Request Body:*

```
{
  "KPI": "DemoMDX",
  "FILTERS": [ {"name" : "[aged].[h1].[age group]","value" : "&[0 to 29]" } ]
}
```

Example Response

```
{
   "Info":{"Error":"","KpiName":"DemoMDX"},
   "Result":
      {"Series":
         [
          {"PatCount":49,"AvgTestScore":77.14705882352941176},
          {"PatCount":46,"AvgTestScore":77.41025641025641026},
          {"PatCount":40,"AvgTestScore":74.37931034482758621},
          {"PatCount":45,"AvgTestScore":74.59459459459459459},
          {"PatCount":53,"AvgTestScore":71.2},
          {"PatCount":51,"AvgTestScore":74.05},
          {"PatCount":44,"AvgTestScore":75.66666666666666667},
          {"PatCount":48,"AvgTestScore":74.86486486486486486},
          {"PatCount":42,"AvgTestScore":74}
         ],
        "Properties":
          [
           {"name":"PatCount","caption":"PatCount","columnNo":1},
           {"name":"AvgTestScore","caption":"AvgTestScore","columnNo":2}
          ]
      }
}
```

In the response object, the `Result` property contains the properties `Series` and `Properties`. The `Series` property contains an array of objects, one for each series (or row) in the KPI. The `Properties` property contains an array of objects, one for each row in the KPI.

For information that applies to all response objects, see the discussion at the start of this reference.

POST /Data/MDXCancelQuery/:queryID

Cancels a previously started query, given the ID of the query.

URL Parameters

queryID	*Required.* ID of the query to cancel. If you started the query with the POST /Data/MDXExecute service, obtain the ID of the query from the `Info.QueryID` property of the response object returned by that service. If you started the query with the POST /Data/PivotExecute service, obtain the ID of the query from the `Info.QueryID` property of the response object returned by that service.

Request Body Details

This service ignores the request body.

Example Request

- *Request Method:*

  `POST`

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Data/MDXCancelQuery/:patients||en2515296118`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

POST /Data/MDXDrillthrough

Executes a detail listing.

URL Parameters

None. Note that a request body is required; see the next heading.

Request Body Details

This service uses the following properties of the request body:

`MDX`	*Required.* MDX SELECT QUERY, preceded by either `DRILLTHROUGH` or `DRILLFACTS`. Use `DRILLTHROUGH` to use a named detail listing or to retrieve fields from the source class of the cube. Use `DRILLFACTS` to retrieve fields from the fact table. If the base SELECT query returns more than one cell, only the top left cell is used for the detail listing.
`LISTING`	Optional. Logical name of the detail listing to use. You must specify either `LISTING` or `RETURN`, but not both.
`WAIT`	Optional. Specify 0 or 1 (the default). If this property is 0, the server sends partial results. If this property is 1, the server assumes the client wishes to wait for complete results before sending a response.
`RETURN`	Optional. List of fields in the applicable table, depending on the value in `MDX`. If you specify this, the listing consists of these fields.

Example Request

- *Request Method:*

  `POST`

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Data/MDXDrillthrough`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

- *Request Body:*

```
{
  "MDX" : "DRILLTHROUGH SELECT FROM patients WHERE AGED.60",
  "LISTING" : "Patient details"
}
```

  For another example:

```
{
  "MDX" : "DRILLTHROUGH SELECT FROM patients WHERE AGED.60",
  "RETURN":"Age,BirthDate"
}
```

  For another example:

```
{
  "MDX" : "DRILLFACTS SELECT FROM patients WHERE AGED.60",
  "RETURN":"MxAge,MxTestScore"
}
```

Example Response


```
{
 "Info":
        {"Error":"",
         "MDXText":"DRILLTHROUGH SELECT FROM [PATIENTS] WHERE [AGED].[60]",
         "QueryKey":"en2156087935","CubeKey":"PATIENTS",
         "QueryID":"PATIENTS||en2156087935","Cube":"patients",
         "ResultsComplete":1,"Pivot":"","QueryType":"DRILLTHROUGH","ListingSource":"source",
         "ColCount":5,"RowCount":0,"Error":"","TimeStamp":"2016-08-14 15:43:04"},
 "AxesInfo":
      [{"%ID":"SlicerInfo","Text":"[AGED].[60]"},
       {"%ID":"AxisInfo_1","Text":"[%SEARCH]"},
       {"%ID":"AxisInfo_2","Text":"[%SEARCH]"}],
 "Result":
   {"children":
      [
       {"PatientID":"SUBJ_100508","Age":60,"Gender":"Female","Home City":"Elm Heights","Test Score":81},

       {"PatientID":"SUBJ_100539","Age":60,"Gender":"Female","Home City":"Elm Heights","Test Score":90},

        {"PatientID":"SUBJ_100701","Age":60,"Gender":"Female","Home City":"Redwood","Test Score":61},
        {"PatientID":"SUBJ_100829","Age":60,"Gender":"Female","Home City":"Juniper","Test Score":98},
        ...]}
}
```


For information that applies to all response objects, see the discussion at the start of this reference.

POST /Data/MDXExecute

Executes an MDX query and obtains the results.

URL Parameters

None. Note that a request body is required; see the next heading.

Request Body Details

This service uses the following properties of the request body:

`MDX`	*Required.* MDX SELECT QUERY.
`FILTERS`	Optional. Any additional filters to add to the query. If specified, this property must be an array of strings, each of which specifies a filter value.
`WAIT`	Optional. Specify 0 or 1 (the default). If this property is 0, the server sends partial results. If this property is 1, the server assumes the client wishes to wait for complete results before sending a response.
`TIMEOUT`	Optional. Timeout for waiting for query results, in seconds. The default timeout for this wait is 2 seconds less than the session's timeout setting.

Example Request

- *Request Method:*

  `POST`

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Data/MDXExecute`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

- *Request Body:*

```
{"MDX": "SELECT aged.[age group].MEMBERS ON 0 FROM PATIENTS"}
```

  For another example:

```
{"MDX": "SELECT FROM PATIENTS"}
```

  For another example:

```
{"MDX": "SELECT birthd.date.members on 0 FROM PATIENTS", "WAIT":1, "TIMEOUT":30}
```

  For another example:

```
{
 "MDX": "SELECT FROM PATIENTS",
 "FILTERS": [ "[HomeD].[H1].[ZIP].&[32006]","[HomeD].[H1].[ZIP].&[32007]"],
 "WAIT":1,
 "TIMEOUT":30
}
```

Example Response

Response for MDX Query

Note that POST /Data/PivotExecute and POST /Data/MDXUpdateResults return the same response body.

The `Info.QueryID` property contains the query ID, which you need as input for the POST /Data/MDXCancelQuery and POST /Data/MDXUpdateResults services. An `Info.ResultsComplete` property with a value of 1 indicates that the MDX query has completed. Note that if the `Info.PendingResults` property has a value of 1, plugins are still computing, although the rest of the query may have completed. An `Info.PendingResults` property with a value of 0 indicates that any plugins have finished computing.

For information that applies to all response objects, see the discussion at the start of this reference.


```
{
   "Info":{
      "Error":"",
      "MDXText":"SELECT [AGED].[AGE GROUP].MEMBERS ON 0 FROM [PATIENTS]",
      "QueryKey":"en2772997983",
      "CubeKey":"PATIENTS",
      "QueryID":"PATIENTS||en2772997983",
      "Cube":"PATIENTS",
      "ResultsComplete":1,
      "Pivot":"",
      "QueryType":"SELECT",
      "ListingSource":"",
      "ColCount":3,
      "RowCount":0,
      "TimeStamp":"2016-08-14 16:05:16"}
      ,"AxesInfo":[
         {"%ID":"SlicerInfo","Text":""},
         {"%ID":"AxisInfo_1","Text":"[AGED].[AGE GROUP].MEMBERS"},
         {"%ID":"AxisInfo_2","Text":"[%SEARCH]"}
      ],
 "Result":
   {"Axes":[
       {"%ID":"Axis_1","Tuples":
         [{"%ID":"Tuple_1",
           "Members":[{"%ID":"Member_1",
                       "Name":"0 to 29"}],
                       "MemberInfo":
                          [{"%ID":"MemberInfo_1",
                            "nodeNo":3,"text":"0 to 29",
                            "dimName":"AgeD",
                            "hierName":"H1",
                            "levelName":"Age Group",
                            "memberKey":"0 to 29",
                            "dimNo":2,
                            "hierNo":1,
                            "levelNo":2,
                            "aggregate":"",
                            "orSpec":""}]},
       {"%ID":"Tuple_2",...},
       {"%ID":"Tuple_3",...}],
    "TupleInfo":[
          {"%ID":"TupleInfo_1","childSpec":"[AgeD].[H1].[Age Group].&[0 to 29].children"},
          {"%ID":"TupleInfo_2","childSpec":"[AgeD].[H1].[Age Group].&[30 to 59].children"}
           ...],
   "CellData":[
         {"%ID":"Cell_1","ValueLogical":418,"Format":"","ValueFormatted":"418"},
         {"%ID":"Cell_2","ValueLogical":421,"Format":"","ValueFormatted":"421"},
...]}
}
```


If the response is incomplete, it includes cell data objects like the following:


```
{"%ID":"Cell_9","ValueLogical":"@Computing 0.00%","Format":"","ValueFormatted":"@Computing 0.00%"}
```


Response for MDX DRILLTHROUGH Query


```
{
  "Info": {
    "Error": "",
    "TimeStamp": "2017-09-26 15:31:23",
    "ResultsComplete": 1,
```

```
    "MDXText": "DRILLTHROUGH SELECT [AGED].[AGE GROUP].[0 TO 29] ON 0 FROM [PATIENTS]",
    "QueryKey": "en2983351588",
    "CubeKey": "PATIENTS",
    "QueryID": "PATIENTS||en2983351588",
    "Cube": "PATIENTS",
    "Pivot": "",
    "QueryType": "DRILLTHROUGH",
    "ListingSource": "source",
    "ColCount": 5,
    "RowCount": 0
  },
  "AxesInfo": [
    {
      "%ID": "SlicerInfo",
      "Text": ""
    },
    {
      "%ID": "AxisInfo_1",
      "Text": "[AGED].[AGE GROUP].[0 TO 29]"
    },
    {
      "%ID": "AxisInfo_2",
      "Text": "[%SEARCH]"
    }
  ],
  "Result": {
    "children": [
      {
        "PatientID": "SUBJ_100786",
        "Age": 0,
        "Gender": "Female",
        "Home City": "Centerville",
        "Test Score": 77
      },
      {
        "PatientID": "SUBJ_100960",
        "Age": 0,
        "Gender": "Female",
        "Home City": "Elm Heights",
        "Test Score": 62
      },
      {
        "PatientID": "SUBJ_100977",
        "Age": 0,
        "Gender": "Female",
        "Home City": "Elm Heights",
        "Test Score": 54
      },
      ...]}
}
```

POST /Data/MDXUpdateResults/:queryID

Retrieves updated results for a given query that was previously incomplete.

URL Parameters

queryID	*Required.* ID of the query. If you started the query with the POST /Data/MDXExecute service, obtain the ID of the query from the `Info.QueryID` property of the response object returned by that service. If you started the query with the POST /Data/PivotExecute service, obtain the ID of the query from the `Info.QueryID` property of the response object returned by that service.

Request Body Details

This service ignores the request body.

Example Request

- *Request Method:*

  `POST`

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Data/MDXCancelQuery/:patients||en2515296118`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

Example Response

This service returns the same response object as POST /Data/MDXExecute.

POST /Data/PivotExecute

Executes the MDX query defined by a pivot table and obtains the results.

URL Parameters

None. Note that a request body is required; see the next heading.

Request Body Details

This service uses the following properties of the request body:

`PIVOT`	*Required.* Full logical name of the pivot table, including the name of the folder that contains it.
`FILTERS`	Optional. Specifies any additional filters to add to the query. If specified, this property must be an array of strings, each of which specifies a filter value.
`WAIT`	Optional. Specify 0 or 1 (the default). If this property is 0, the server sends partial results. If this property is 1, the server assumes the client wishes to wait for complete results before sending a response.
`TIMEOUT`	Optional. Timeout for waiting for query results, in seconds. The default timeout for this wait is 2 seconds less than the session's timeout setting.
`VARIABLES`	Optional. Specifies the values for any pivot variables used in the pivot table. Specify this as an array of objects. Each object must specify the `name` and `value` properties.

Example Request

- *Request Method:*

  `POST`

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Data/PivotExecute`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

- *Request Body:*

```
{ "PIVOT":"Use in Dashboards/Product Info" }
```

  For another example:

```
{ "PIVOT":"Pivot Variables/Commission Calculator",
  "VARIABLES": [{"name":"commissionpercentage", "value":15}]
}
```

Example Response

This service returns the same response object as POST /Data/MDXExecute.

POST /Info/Cubes

Returns information about the cubes and subject areas available in the Caché namespace that you access via this REST call.

URL Parameters

None.

Request Body Details

This service uses the following properties of the request body:

TYPE	Optional. This property can be `cubes` or `subjectareas`. If this property is `cubes`, the server sends information only about cubes. If this property is `subjectareas`, the server sends information only about subject areas. If this property is not specified, the server sends information about both cubes and subject areas.
BASECUBE	Optional. If specified, this property should equal the logical name of a cube. In this case, the server sends information only about cubes and subject areas based on this cube.

Example Request

- *Request Method:*

  POST

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Info/Cubes`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

- *Request Body:*

```
{"BASECUBE":"patients"}
```

Example Response

```
{
 "Info":
   {"Error":"","Type":"all","BaseCube":"patients"},
 "Result":
   {"Cubes":[
      {"name":"AsthmaPatients","displayName":"AsthmaPatients","modTime":"2016-11-14
20:49:14","type":"subjectArea"},
     {"name":"DemoMDX","displayName":"DemoMDX","modTime":"2016-11-14 20:49:14","type":"subjectArea"},

      {"name":"YoungPatients","displayName":"YoungPatients","modTime":"2016-11-14
20:49:14","type":"subjectArea"}
      ]
   }
}
```

In the response object, the `Result` property contains a property called `Cubes`, which contains an array of objects, one for each cube or subject area. In these objects, the `type` property indicates whether the item is a cube or a subject area.

For information that applies to all response objects, see the discussion at the start of this reference.

POST /Info/Dashboards

Returns information about the dashboards available in the Caché namespace that you access via this REST call.

URL Parameters

None.

Request Body Details

This service ignores the request body.

Example Request

- *Request Method:*

  `POST`

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Info/Dashboards`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

Example Response

```
{
   "Info":
       {"Error":""},
   "Result":
       {"Dashboards":
          [
           {"fullName":"Basic Dashboard Demo",
            "name":"Basic Dashboard Demo",
            "lastModified":"2016-11-14 19:39:14",
            "itemType":"dashboard"},
           {"fullName":"Custom Drilldown Spec",
            "name":"Custom Drilldown Spec",
            "lastModified":"2016-11-14 19:39:14",
            "itemType":"dashboard"}
          ...]
       }
}
```

In the response object, the `Result` property contains a property called `Dashboards`, which contains an array of objects, one for each dashboard.

For information that applies to all response objects, see the discussion at the start of this reference.

POST /Info/FilterMembers/:datasource/:filterSpec

Returns information about the members of the given filter, as defined by the given data source (which is either a cube or a KPI).

URL Parameters

datasource	*Required.* Name of the data source. This is one of the following: • *cubename* — a logical cube name • *cubename*.cube — a logical cube name, followed by .cube • *kpiname*.kpi — a logical KPI name, followed by .kpi This name can include slashes; see "Use of Slashes in Cube and KPI Names," earlier in this book.
filterSpec	*Required.* Filter specification.

Request Body Details

This service uses the following properties of the request body. These properties both specify filters that are applied to the data source, thus limiting the list of members returned by the service.

RELATED	Optional. If specified, this property is an array of objects, each of which contains the spec property (a filter specification) and the value property (value of that filter). A value property should be an MDX set expression and should use member keys.
SEARCHKEY	Optional.

Example Request

- *Request Method:*

  POST

- *Request URL:*

  http://localhost:57772/api/deepsee/v1/Info/FilterMembers/:demomdx.kpi/:homed.h1.zip

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

- *Request Body:*

```
{
  "RELATED":[{"spec":"gend.h1.gender","value":"&[female]"}],
        "SEARCHKEY":"Jan"
}
```

Example Response

```
{
   "Info":

{"Error":"","DataSource":"demomdx.cube","DataSourceType":"cube","Default":"","Filter":"[HomeD].[H1].[ZIP]"},

   "Result":
      {"FilterMembers":
         [
          {"text":"32006","value":"&[32006]","description":""},
          {"text":"32007","value":"&[32007]","description":""},
          {"text":"34577","value":"&[34577]","description":""},
          {"text":"36711","value":"&[36711]","description":""},
          {"text":"38928","value":"&[38928]","description":""}
          ]
       }
}
```

In the response object, the `Result` property contains a property called `FilterMembers`, which contains an array of objects, one for each filter member. The object for a given filter member contains the following properties:

- `description` contains the text description of the member, if any
- `text` contains the display text for the member
- `value` contains the logical value of the member (typically the MDX key)

For information that applies to all response objects, see the discussion at the start of this reference.

POST /Info/Filters/:datasource

Returns information about the filters available for the given data source (which is either a cube or a KPI).

URL Parameters

datasource	*Required.* Name of the data source. This is one of the following: • *cubename* — a logical cube name • *cubename*.`cube` — a logical cube name, followed by `.cube` • *kpiname*.`kpi` — a logical KPI name, followed by `.kpi` This name can include slashes; see "Use of Slashes in Cube and KPI Names," earlier in this book.

Request Body Details

This service ignores the request body.

Example Request

- *Request Method:*

  `POST`

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Info/Filters/:aviationevents`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

Example Response

```
{
   "Info":
       {"Error":"","DataSource":"aviationevents.cube","DataSourceType":"cube"},
   "Result":
       {"Filters":
          [
           {"caption":"Year","value":"[EventDateD].[H1].[Year]","type":"year"},
           {"caption":"Month","value":"[EventDateD].[H1].[Month]","type":""},
           {"caption":"Day","value":"[EventDateD].[H1].[Day]","type":""}
           ...]
       }
}
```

In the response object, the `Result` property contains a property called `Filters`, which contains an array of objects, one for each filter. Each object has the following properties:

- `caption` contains the display value for the filter.
- `type` contains the filter type, if it exists. This can be `"calc"` or can be the name of a time function. In other cases it is empty.
- `value` contains the filter specification, which is the logical identifier for the filter. For information on the filter specification, see POST /Info/FilterMembers/:datasource/:filterSpec.

For information that applies to all response objects, see the discussion at the start of this reference.

POST /Info/KPIs

Returns information about the KPIs available in the Caché namespace that you access via this REST call.

URL Parameters

None.

Request Body Details

This service ignores the request body.

Example Request

- *Request Method:*

  `POST`

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Info/KPIs`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

Example Response

```
{  "Info":
      {"Error":""},
   "Result":
      {"KPIs":
         [
           {"name":"Aviation Actions","caption":"Aviation Actions",
                                      "moddate":"2016-11-14 11:22:08","type":"kpi"},
           {"name":"AviationTopConcepts","caption":"AviationTopConcepts",
                                      "moddate":"2016-11-14 11:22:08","type":"kpi"},
           {"name":"BPDiastolic","caption":"BPDiastolic",
                                      "moddate":"2016-11-14 11:22:08","type":"kpi"}
          ...]
      }
}
```

In the response object, the `Result` property contains a property called `KPIs`, which contains an array of objects, one for each KPI.

For information that applies to all response objects, see the discussion at the start of this reference.

POST /Info/ListingFields/:cube

Returns information about the <listingField> elements available in the Caché namespace that you access via this REST call.

URL Parameters

cube	*Required.* Logical name of the cube. This name can include slashes; see "Use of Slashes in Cube and KPI Names," earlier in this book.

Request Body Details

This service ignores the request body.

Example Request

- *Request Method:*

  `POST`

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Info/ListingFields/:holefoods`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

Example Response

```
{
   "Info":
      {"Error":"","BaseCube":"holefoods"},
   "Result":
      {"ListingFields":
         [
           {"caption":"Channel","expression":"%EXTERNAL(Channel) Channel"},
           {"caption":"City","expression":"Outlet->City"},
           {"caption":"Comment","expression":"Comment"},
           ...]
      }
}
```

In the response object, the `Result` property contains a property called `ListingFields`, which contains an array of objects, one for each <listingField> element.

For information that applies to all response objects, see the discussion at the start of this reference.

POST /Info/Listings/:cube

Returns information about the detail listings available for the given cube.

URL Parameters

cube	*Required.* Logical name of the cube. This name can include slashes; see "Use of Slashes in Cube and KPI Names," earlier in this book.

Request Body Details

This service uses the following property of the request body:

TYPE	Optional. This property can be `map` or `table`. If this property is `map`, the server sends information only about map listings. If this property is `table`, the server sends information only about non-map listings. If this property is not specified, the server sends information about all kinds of listings.

Example Request

- *Request Method:*

  POST

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Info/Listings/:holefoods`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

- *Request Body:*

```
{"TYPE":"map"}
```

Example Response

For another example:

```
{
   "Info":
      {"Error":"","DataSource":"holefoods","Type":"all"},
   "Result":
      {"Listings":
         [
          {"name":"Custom Listing"},
          {"name":"Another Sample Listing by Date",
           "fields":"%ID As \"ID #\",DateOfSale As \"Sale Date\"",
           "order":"DateOfSale,%ID",
           "type":"table",
           "source":"listingGroup",
           "edit":1},
          {"name":"Another Sample Listing with Customer Info",
           "fields":"%ID,Outlet->City \"Store Location\",Outlet->Country->Name Country,Product->Name
Product,
                    ZipCode \"Customer ZipCode\",Latitude,Longitude","order":"",
           "type":"map",
           "source":"listingGroup",
           "edit":1},
          {"name":"Customer Info",
           "fields":"%ID,Outlet->City \"Store Location\",Outlet->Country->Name Country,Product->Name
Product,
                    ZipCode \"Customer ZipCode\",Latitude,Longitude","order":"",
           "type":"map",
```

```
            "source":"cube",
            "edit":0},
           ...]
    }
}
```

In the response object, the `Result` property contains a property called `Listings`, which contains an array of objects, one for each detail listing.

For information that applies to all response objects, see the discussion at the start of this reference.

POST /Info/Measures/:cube

Returns information about the measures available to the given cube.

URL Parameters

cube	*Required.* Logical name of the cube. This name can include slashes; see "Use of Slashes in Cube and KPI Names," earlier in this book.

Request Body Details

This service uses the following property of the request body:

`SKIPCALCULATED`	Optional.

Example Request

- *Request Method:*

  `POST`

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Info/Measures/:demomdx`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

Example Response

```
{
   "Info":
      {"Error":"",
       "BaseCube":"DemoMDX",
       "SkipCalculated":0},
   "Result":
      {"Measures":
       [
         {"name":"%COUNT","caption":"%COUNT","type":"integer","hidden":0,"factName":""},
         {"name":"Age","caption":"Age","type":"integer","hidden":0,"factName":"MxAge"}
        ...]
   }
}
```

In the response object, the `Result` property contains a property called `Measures`, which contains an array of objects, one for each measure.

For information that applies to all response objects, see the discussion at the start of this reference.

POST /Info/NamedFilters/:cube

Returns information about the named filters available to the given cube.

URL Parameters

cube	*Required.* Logical name of the cube. This name can include slashes; see "Use of Slashes in Cube and KPI Names," earlier in this book.

Request Body Details

This service ignores the request body.

Example Request

- *Request Method:*

  `POST`

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Info/NamedFilters/:holefoods`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

Example Response

```
{
   "Info":
      {"Error":"","BaseCube":"holefoods"},
   "Result":
      {"NamedFilters":
         [
           {"name":"Sample Named Filter",
            "description":"",
            "spec":"[Product].[P1].[Product Category].&[Seafood]","cube":"HOLEFOODS"}
         ]
      }
}
```

In the response object, the `Result` property contains a property called `NamedFilters`, which contains an array of objects, one for each named filter.

For information that applies to all response objects, see the discussion at the start of this reference.

POST /Info/Pivots

Returns information about the pivot tables available in the Caché namespace that you access via this REST call.

URL Parameters

None.

Request Body Details

This service uses the following property of the request body:

BASECUBE	Optional. If specified, this property should equal the logical name of a cube. In this case, the server sends information only about pivot tables based on this cube.

Example Request

- *Request Method:*

  POST

- *Request URL:*

  http://localhost:57772/api/deepsee/v1/Info/Pivots

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

- *Request Body:*

```
{"BASECUBE":"PATIENTS"}
```

Example Response

```
{
   "Info":
      {"Error":"","BaseCube":""},
   "Result":
      {"Pivots":
        [
          {"fullName":"Calculated Members\/Alternative Avg Allergy Count",
           "name":"Alternative Avg Allergy Count",
           "lastModified":"2016-11-14 11:22:08",
           "itemType":"pivot"},
          {"fullName":"Calculated Members\/Average Patient Count per Decade",
           "name":"Average Patient Count per Decade",
           "lastModified":"2016-11-14 11:22:08",
           "itemType":"pivot"}
         ...]
      }
}
```

In the response object, the `Result` property contains a property called `Pivots`, which contains an array of objects, one for each pivot table.

For information that applies to all response objects, see the discussion at the start of this reference.

POST /Info/PivotVariableDetails/:cube/:variable

Returns detailed information about the given pivot variable.

URL Parameters

cube	*Required.* Logical name of a cube that has access to the given pivot variable. This name can include slashes; see "Use of Slashes in Cube and KPI Names," earlier in this book.
variable	*Required.* Logical name of the pivot variable.

Request Body Details

This service ignores the request body.

Example Request

- *Request Method:*

  `POST`

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Info/PivotVariableDetails/:holefoods/:Year`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

Example Response

```
{
   "Info":
      {"Error":"","BaseCube":"holefoods","Variable":"Year"},
   "Result":
      {"Details":
        {"context":"literal","defaultValue":"NOW","description":"","displayList":"","displayName":"Year",

"name":"Year","sourceName":"HoleFoodsYears.kpi","sourceType":"kpi","type":"string","valueList":""}
       }
}
```

In the response object, the `Result` property contains a property called `Details`, which contains the details for the pivot variable.

For information that applies to all response objects, see the discussion at the start of this reference.

POST /Info/PivotVariables/:cube

Returns information about the pivot variables available to the given cube.

URL Parameters

cube	*Required.* Logical name of the cube. This name can include slashes; see "Use of Slashes in Cube and KPI Names," earlier in this book.

Request Body Details

This service ignores the request body.

Example Request

- *Request Method:*

  `POST`

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Info/PivotVariables/:holefoods`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

Example Response

```
{
   "Info":
      {"Error":"","BaseCube":"holefoods"},
   "Result":
      {"PivotVariables":
         [
           {"name":"CommissionPercentage","caption":"Commission Percentage","defValue":"0",
                                          "context":"literal","desc":""},
           {"name":"Year","caption":"Year","defValue":"NOW",
                                          "context":"literal","desc":""}
         ]
      }
}
```

In the response object, the `Result` property contains a property called `PivotVariables`, which contains an array of objects, one for each pivot variable.

For information that applies to all response objects, see the discussion at the start of this reference.

POST /Info/QualityMeasures/:cube

Returns information about the quality measures available to the given cube.

URL Parameters

cube	*Required.* Logical name of the cube. This name can include slashes; see "Use of Slashes in Cube and KPI Names," earlier in this book.

Request Body Details

This service ignores the request body.

Example Request

- *Request Method:*

  `POST`

- *Request URL:*

  `http://localhost:57772/api/deepsee/v1/Info/QualityMeasures/:holefoods`

  For comments on the possible forms of the URL, see "Introduction to the DeepSee REST API" in the chapter "Introduction and Samples."

Example Response

```
{
   "Info":
      {"Error":"","BaseCube":"holefoods"},
   "Result":
      {"QualityMeasures":
         [
          {"name":"TestCatalog\/TestSet\/TestQM","caption":"Sample Quality Measure","description":""}]}
}
```

In the response object, the `Result` property contains a property called `QualityMeasures`, which contains an array of objects, one for each quality measure.

For information that applies to all response objects, see the discussion at the start of this reference.