IBM Cognos Software Development Kit
Version 11.0.0

Dynamic Cubes Developer Guide

Note

Before using this information and the product it supports, read the information in “Notices” on page 83.

Product Information

This document applies to IBM Cognos Software Development Kit Version 11.0.0 and may also apply to subsequent releases.

Contents

Introduction

This document is intended for use with IBM® Cognos® Dynamic Cubes. It describes the API available to create applications that model dimensional metadata and create dynamic cubes for use as data sources in the Content Manager.

Audience

To use the *IBM Cognos Dynamic Cubes Developer Guide* effectively, you must be familiar with the following items:

- Cognos Dynamic Cubes, IBM Cognos Cube Designer, and the *IBM Cognos Dynamic Cubes User Guide*.
- The representational state transfer (REST) web services architecture.
- A programming language that has libraries for making HTTP requests.

Finding information

To find product documentation on the web, including all translated documentation, access IBM Knowledge Center (http://www.ibm.com/support/knowledgecenter).

Forward-looking statements

This documentation describes the current functionality of the product. References to items that are not currently available may be included. No implication of any future availability should be inferred. Any such references are not a commitment, promise, or legal obligation to deliver any material, code, or functionality. The development, release, and timing of features or functionality remain at the sole discretion of IBM.

Samples disclaimer

The Sample Outdoors Company, Great Outdoors Company, GO Sales, any variation of the Sample Outdoors or Great Outdoors names, and Planning Sample depict fictitious business operations with sample data used to develop sample applications for IBM and IBM customers. These fictitious records include sample data for sales transactions, product distribution, finance, and human resources. Any resemblance to actual names, addresses, contact numbers, or transaction values is coincidental. Other sample files may contain fictional data manually or machine generated, factual data compiled from academic or public sources, or data used with permission of the copyright holder, for use as sample data to develop sample applications. Product names referenced may be the trademarks of their respective owners. Unauthorized duplication is prohibited.

Accessibility features

Consult the documentation for the tools that you use to develop applications to determine their accessibility level. These tools are not a part of this product.

IBM Cognos HTML documentation has accessibility features. PDF documents are supplemental and, as such, include no added accessibility features.

Chapter 1. What's new?

This topic contains a list of new features for this release of the IBM Cognos Dynamic Cubes API. It helps you plan your upgrade and application deployment strategies and the training requirements for your users.

New features in version 10.2.2

New features have been added to the IBM Cognos Dynamic Cubes API and are described here.

Support for in-memory aggregates

This release of the Dynamic Cubes API adds support for in-memory aggregates.

The following commands have been added to support modeling in-memory aggregates.

- aggregate
- aggregate_dimension
- aggregate_hierarchy
- aggregate_level
- aggregate_measure

The following command has been updated to support modeling in-memory aggregates: cube

The following properties were added:

- `inDatabaseAggregates`
- `inMemoryAggregates`

You can see usage examples for aggregate modeling in Chapter 5, “Aggregate modeling using the Cognos Dynamic Cubes API,” on page 33.

Support for calculated members and measures

This release of the Dynamic Cubes API adds support for calculated members and measures.

The following command has been added to support modeling calculated members and measures: calculated_member

The following commands have been updated to support modeling calculated members and measures:

- measure_dimension
  - The `defaultMeasure` property can be a calculated measure.
  - Added the `calculatedMeasures` property.
- measure_folder
  - Added the `calculatedMeasures` property.
- relational_hierarchy
  - Added the `calculatedMembers` property.
- virtual_hierarchy
  - Added the `calculatedMembers` property.
- virtual_measure_dimension
  - The `defaultVirtualMeasure` property can be a calculated measure.
  - Added the `calculatedMeasures` property.

You can see usage examples for creating calculated members and measures in “Create calculated members and measures” on page 38.

Support for slices

This release of the Dynamic Cubes API adds support for slices.

The following command has been updated to support modeling slices: cube

- Added the `slices` property.

You can see usage examples for modeling slices in “Filter data using an aggregate slicer” on page 37.

Support for relative time members

This release of the Dynamic Cubes API adds support for relative time members.

The following command has been added to support modeling relative time members: relative_time_member

The following commands have been updated to support modeling relative time members:

- relational_hierarchy
  - The `levelType` property can take the following additional values:
    - `time_holidays`
    - `time_quarters`
    - `time_seasons`
    - `time_semesters`
    - `time_trimesters`
- virtual_hierarchy
  - Added the `calculatedMembers` property.
- virtual_measure_dimension
  - Added the following properties:
    - `generateNextPeriodsMembers`
    - `generatePriorPeriodsMembers`
    - `includeRelativeTimeSubtree`
    - `relativeTimeMembers`

You can see usage examples for modeling relative time members in “Use relative time functionality” on page 38.

Support for named sets

This release of the Dynamic Cubes API adds support for named sets.

The following commands have been added to support modeling named sets:

- named_set
- named_set_folder

The following commands have been updated to support modeling named sets:

- cube
  - The following properties were added:
    - `namedSetFolders`
    - `namedSets`
- virtual_cube

  - The following properties were added:
    - `namedSetFolders`
    - `namedSets`

You can see usage examples for modeling named sets in "Create named sets" on page 40.

Support for parameter maps

This release of the Dynamic Cubes API adds support for parameter maps.

The following commands have been added to support modeling parameter maps:

- parameter_map
- realtional_parameter_map

The following commands have been updated to support modeling parameter maps:

- model
  - Added the `parameterMaps` property.
- query_item
  - The `parent` object can be a relational parameter map.
- relational_filter
  - The `parent` object can be a relational parameter map.

You can see usage examples for modeling parameter maps in "Create parameter maps" on page 41.

Support for virtual measure folders

This release of the Dynamic Cubes API adds support for virtual measure folders.

The following command has been added to support modeling virtual measure folders:
virtual_measure_folder

The following commands have been updated to support virtual measure folders:

- virtual_measure
  - The `parent` object can be a virtual measure folder.
- virtual_measure_dimension
  - Added the `folders` property.

Support for new expression types

This release of the Dynamic Cubes API adds support for new expression types.

The following expression types are now available:

- A member unique name.
- An `id` property of a model object.

For more information, see "Expressions" on page 82.

New features in version 10.2.1 interim fix 3

New features have been added to the IBM Cognos Dynamic Cubes API and are described here. These features have also been rolled up into version 10.2.1 fix pack 3.

Support for virtual cubes

This release of the Dynamic Cubes API adds support for virtual cubes.

The following commands have been added to support modeling virtual cubes:

- virtual_cube
- virtual_dimension
- virtual_hierarchy
- virtual_level
- virtual_measure
- virtual_measure_dimension
- virtual_source

You can see usage examples for virtual cube modeling in Chapter 4, “Virtual cube modeling using the Cognos Dynamic Cubes API,” on page 25.

Chapter 2. Overview of the Cognos Dynamic Cubes API

The IBM Cognos Dynamic Cubes API automates the functions available in the IBM Cognos Cube Designer. The API uses the representational state transfer (REST) web services architecture.

Use the Cognos Dynamic Cubes API to send HTTP requests to the IBM Cognos Analytics server to work with dynamic cubes, including creation, modification, deployment, and startup. The HTTP response from the server contains data about any actions that are run, along with any error or warning messages that are generated by the request. Data sent to and retrieved from the server is in the JavaScript Object Notation (JSON) format.

Chapter 3, "Sample Cognos Dynamic Cubes model creation," on page 9 describes the steps in the creation of a simple model using the `StartToFinish.java` sample program as an example. The commands that are used and the input and output JSON objects are described. This topic shows you how the results of operations are used to do subsequent operations.

Chapter 7, "Cognos Dynamic Cubes command reference," on page 43 documents the commands available in the Cognos Dynamic Cubes API.

Cognos Dynamic Cubes HTTP request structure

The IBM Cognos Dynamic Cubes HTTP request can be generated by the web libraries available with most programming languages. The parts of the HTTP request are described here.

HTTP method

The IBM Cognos Dynamic Cubes API HTTP request uses the 4 common HTTP methods:

GET
Used to retrieve the properties of an object.

POST
Used to create an object or execute a command.

PUT
Used to update the properties of an object.

DELETE
Used to delete an object.

Important: Only GET or POST can be specified as the HTTP method. To use PUT or DELETE, specify POST as the method and use the `X-HTTP-Method-Override` request header with a value of PUT or DELETE.

HTTP path

The HTTP path is based on the dispatcher URL of the Cognos Analytics server and has one of the two forms:

- *<dispatcher_url>*/FmCommand/*<command>*
- *<dispatcher_url>*/FmCommand/*<command>*/*<object_id>*

where

- *<dispatcher_url>* is the **External dispatcher URI** of the Cognos Analytics server as specified in IBM Cognos Connection.
- *<command>* is the Cognos Dynamic Cubes command.
- *<object_id>* is an object `id` that is required for some commands.

Request headers

Some commands require that you include request headers. If you are sending data in the entity-body, you must include a `Content-Type: application/json` header. Other headers are required on occasion and they are mentioned as appropriate.

Entity-body

Many commands require that you submit data in the entity-body. This data is packaged as a JSON object. The data that is required is described in the documentation for the individual commands. You should consider the following when creating the JSON object:

- The names in the name-value pairs are case-sensitive.
- The name-value pairs can be entered in any order.
- Localized text and expressions have special structures. See the linked topics for details.

Cognos Dynamic Cubes HTTP response structure

The IBM Cognos Dynamic Cubes HTTP response is described here.

Response code

The response code from a Cognos Dynamic Cubes request is always `200` or `201`. A different response code may be returned if the request path is malformed or the Cognos Dynamic Cubes API is not available. All information, including errors or warnings, is included in the entity-body of the response.

Response headers

The response headers contain the standard information about the server, cookies, and so on. There is also a `Content-Type: application/json;charset=utf-8` header.

Entity-body

The entity-body consists of a JSON object that contains the response to a request, including any error or warning messages.

If the request is to create an object, then the response consists of an identifier that can be used to refer to the object in subsequent requests, such as requests that create child objects or updating the properties of the object. A sample response is shown here.

```
{
        "id": "a36907c1f796426a96b6749d0651c0cd"
}
```

Important: These identifiers do not persist and should be assumed only to be valid for the period after a model is created or opened, and before the model is closed.

A request to retrieve the properties of a data source returns a JSON object such as the following.

```
{
    "schema": "gosales",
    "queryProcessing": "databaseOnly",
    "queryType": "relational",
    "functionSetId": "V_SQLServer",
    "catalog": "GOSALES",
    "rollupProcessing": "unspecified",
    "name": "great_outdoors_sales",
    "cmDataSource": "great_outdoors_sales",
    "cube": "",
```

```
    "interface": "SS"
}
```

The data that is returned is described in the documentation for the individual commands.

A request returns an empty JSON object if the command ran and no response is needed.

If errors are encountered when you attempt to run a request, the response contains an array that is called `errors` that contains any error messages, such as the following.

```
{
    "errors": [
        "Unable to find model object for ID: 62613913c45e47be80de79340effde9"
    ]
}
```

If there any warnings that are generated by the request, they are contained in a `warnings` array.

Cognos Dynamic Cubes sample programs

There are two sample programs, which are written in the Java™ programming language, that you can use to explore the IBM Cognos Dynamic Cubes API.

The sample programs are in the *<installation_location>*\sdk\fmdsdk\java folder. The sample programs are described here:

StartToFinish.java
: This sample program creates a simple model that is based on the `great_outdoors_sales` sample data source, and creates, deploys, registers, and starts a cube that is based on the model.

PseudoTranslate.java
: This sample program demonstrates model modifications. It finds an English name of a cube and creates a pseudo-translated name by adding lead-in and lead-out characters.

  This sample program requires two command line arguments, which are the file paths and names of the input and output model files. The file paths are relative to the location of the sample program.

To use the sample programs, you must

- Ensure the IBM Cognos Analytics server supports anonymous access.
- Replace the string BI_SERVER_URL in each Java program with the **External dispatcher URI** of the server.
- Add *<installation_location>*\webapps\p2pd\WEB-INF\lib\JSON4J.jar to the class path, along with the standard Java libraries.

Command overview

Each command in the IBM Cognos Dynamic Cubes API performs a specific action, such as create a relational dimension, or update the properties of a measure. Creating a model involves running a series of linked commands that create the model, along with its child objects, such as dimensions and measures.

Objects in Cognos Dynamic Cubes have parent-child relationships with other objects. To create two objects with a parent-child relationship, do the following steps.

1. Create the parent object. The output JSON object contains an identifier for the object, as shown here.

```
{
  "id": "1f95121f943b44ff8eabb25c392ab7b1"
}
```

2. Create the child object. In the input JSON object, include a `parent` name-value pair with this identifier, along with any other required properties as shown here.

```
{
  "parent": "1f95121f943b44ff8eabb25c392ab7b1",
  other properties...
}
```

If you get the properties of a parent object, the properties include arrays of child items so you can determine, from the properties of the parent object, which child objects it has. These properties are read-only. You create a parent-child relationship when you create the child object as previously demonstrated.

Chapter 3. Sample Cognos Dynamic Cubes model creation

The `StartToFinish.java` sample program is an example of creating a simple model, and then creating, deploying, registering, and starting a cube, using the IBM Cognos Dynamic Cubes API.

Each step in the model creation process is briefly explained, and the input and output JSON objects are displayed. The identifiers that are created by the Cognos Dynamic Cubes API are shown as variables (such as *<modelId>*) since the values differ every time the sample program is run. Unless otherwise specified, the HTTP POST method is used for the Cognos Dynamic Cubes API calls described in this sample program.

Creating a model

The first step when you are creating a model is to create a model object. Then, the child objects of the model are also created and the relationships between the components are defined.

Creating a model - `Project`

The `model_new` command creates a new model called `Project`. The identifier that is contained in the response is used in subsequent commands to refer to the model.

Input JSON data

```
{
  "name": "Project"
  "locale": "en",
  "namespace": "Model",
}
```

Output JSON data

```
{
  "id": "<modelId>"
}
```

Creating a data source for the model

A data source is created that is used by the model. This data source is based on the GO Sales sample database.

Creating a data source - `great_outdoors_sales`

The `datasource` command creates a data source for the model. The identifier that is contained in the response is used in subsequent commands to refer to the data source.

Input JSON data

```
{
  "parent": "<modelId>",
  "schema": "gosales",
  "queryType": "relational",
  "functionSetId": "V_SQLServer",
  "interface": "SS",
```

```
  "catalog": "GOSALES",
  "name": "great_outdoors_sales",
  "cmDataSource": "great_outdoors_sales"
}
```

Output JSON data

```
{
  "id": "<idDataSource>"
}
```

Creating a relational dimension

A relational dimension is created for the model.

Creating a relational dimension - `Products`

The `relational_dimension` command creates a relational dimension for the model called `Products`. The identifier contained in the response is used in subsequent commands to refer to the relational dimension.

Input JSON data

```
{
  "parent": "<modelId>",
  "name": [
    {
      "locale": "en",
      "text": "Products"
    }
  ]
}
```

Output JSON data

```
{
  "id": "<idDimension>"
}
```

Creating a level for the relational dimension

A level called `Line` is created for the relational dimension. In addition, attributes `Product Line Code` and `Product Line En` are created for the level.

Creating a level - `Line`

The `level` command creates a level called `Line`. The identifier contained in the response is used in subsequent commands to refer to the level.

Input JSON data

```
{
  "parent": "<idDimension>",
  "name": [
    {
```

```
      "locale": "en",
      "text": "Line"
    }
  ]
}
```

Output JSON data

```
{
  "id": "<idLevel>"
}
```

Creating an attribute - `Product Line Code`

The `query_item` command creates an attribute called `Product Line Code` for the `Line` level. The identifier contained in the response is used in subsequent commands to refer to the attribute.

Input JSON data

```
{
  "parent": "<idLevel>",
  "usage": "identifier",
  "datatype": "int32",
  "isLevelKey": "true",
  "scale": "0",
  "parent": "<idLevel>",
  "name": [
    {
      "locale": "en",
      "text": "Product Line Code"
    }
  ],
  "precision": "10"
}
```

Output JSON data

```
{
  "id": "<idProductLineCode>"
}
```

Creating an attribute - `Product Line En`

The `query_item` command creates an attribute called `Product Line En` for the `Line` level. The identifier contained in the response is used in subsequent commands to refer to the attribute.

Input JSON data

```
{
  "parent": "<idLevel>",
  "size": "30",
  "usage": "identifier",
  "datatype": "nVarChar",
  "name": [
    {
      "locale": "en",
      "text": "Product Line En"
    }
  ]
```

```
}
```

Output JSON data

```
{
  "id": "<idProductLine>"
}
```

Creating the member caption role for `Product Line En`

The `query_item_role` command creates the member caption role for the `Product Line En` attribute.

Input JSON data

```
{
  "parent": "<idProductLine>",
  "name": [
    {
      "locale": "en",
      "text": "_memberCaption"
    }
  ]
}
```

Output JSON data

```
{
  "id": "<query_item_role>"
}
```

Creating a second level for the relational dimension

A second level called `Line` is created for the relational dimension. In addition, attributes `Product Type Code` and `Product Type En` are created for the level.

Creating a level - Type

The `level` command creates a level called `Type`. The identifier contained in the response is used in subsequent commands to refer to the level.

Input JSON data

```
{
  "parent": "<idDimension>",
  "name": [
    {
      "locale": "en",
      "text": "Type"
    }
  ]
}
```

Output JSON data

```
{
  "id": "<idTypeLevel>"
```

```
}
```

Creating an attribute - Product Type Code

The query_item command creates an attribute called Product Type Code for the Type level. The identifier contained in the response is used in subsequent commands to refer to the attribute.

Input JSON data

```
{
  "parent": "<idTypeLevel>",
  "usage": "identifier",
  "datatype": "int32",
  "isLevelKey": "true",
  "scale": "0",
  "name": [
    {
      "locale": "en",
      "text": "Product Type Code"
    }
  ],
  "precision": "10"
}
```

Output JSON data

```
{
  "id": "<idProductTypeCode>"
}
```

Creating an attribute - Product Type En

The query_item command creates an attribute called Product Type En for the Type level. The identifier contained in the response is used in subsequent commands to refer to the attribute.

Input JSON data

```
{
  "parent": "<idTypeLevel>",
  "size": "40",
  "usage": "identifier",
  "datatype": "nVarChar",
  "name": [
    {
      "locale": "en",
      "text": "Product Type En"
    }
  ]
}
```

Output JSON data

```
{
  "id": "<idProductType>"
}
```

Creating the member caption role for Product Type En

The query_item_role command creates the member caption role for the Product Type En attribute.

Input JSON data

```
{
  "parent": "<idProductType>",
  "name": [
    {
      "locale": "en",
      "text": "_memberCaption"
    }
  ]
}
```

Output JSON data

```
{
  "id": "<query_item_role>"
}
```

Creating a relational hierarchy for the model

A relational hierarchy is added to the model.

Creating a relational hierarchy

The relational_hierarchy command creates a relational hierarchy that contains the Line and Type levels.

Input JSON data

```
{
  "parent": "<idDimension>",
  "defaultHierarchy": "true",
  "rootCaption": "All",
  "isParentChild": "false",
  "multiRoot": "false",
  "name": [
    {
      "locale": "en",
      "text": "ByType"
    }
  ],
  "rootMember": "All",
  "levels": ["<idLevel>",
  "<idTypeLevel>"]
}
```

Output JSON data

```
{
  "id": "<relational_hierarchy>"
}
```

Creating physical tables and joins

Three physical tables, PRODUCT_LINE, PRODUCT_TYPE, and PRODUCT are created. In addition, joins are established between the PRODUCT_TYPE and PRODUCT_LINE tables, and between the PRODUCT_TYPE and PRODUCT tables.

Creating a physical table - PRODUCT_LINE

The physical_table command creates a physical table called PRODUCT_LINE. The identifier contained in the response is used in subsequent commands to refer to the physical table.

Input JSON data

```
{
  "parent": "<idDimension>",
  "datasource": "<idDataSource>",
  "name": "PRODUCT_LINE"
}
```

Output JSON data

```
{
  "id": "<idTableLine>"
}
```

Creating a physical table - PRODUCT_TYPE

The physical_table command creates a physical table called PRODUCT_TYPE. The identifier contained in the response is used in subsequent commands to refer to the physical table.

Input JSON data

```
{
  "parent": "<idDimension>",
  "datasource": "<idDataSource>",
  "name": "PRODUCT_TYPE"
}
```

Output JSON data

```
{
  "id": "<idTableType>"
}
```

Creating a physical table - PRODUCT

The physical_table command creates a physical table called PRODUCT. The identifier contained in the response is used in subsequent commands to refer to the physical table.

Input JSON data

```
{
  "parent": "<idDimension>",
  "datasource": "<idDataSource>",
  "name": "PRODUCT"
```

```
}
```

Output JSON data

```
{
  "id": "<idTableProduct>"
}
```

Creating a physical join - PRODUCT_TYPE to PRODUCT_LINE

The physical_join command creates a physical join between the PRODUCT_TYPE and PRODUCT_LINE tables. The identifier contained in the response is used in subsequent commands to refer to the physical join.

Input JSON data

```
{
  "parent": "<idDimension>",
  "leftMaxCardinality": "one",
  "rightTable": "<idTableType>",
  "leftMinCardinality": "one",
  "rightMaxCardinality": "many",
  "leftTable": "<idTableLine>",
  "rightMinCardinality": "one",
  "name": "FK_PRODUCT_TYPE_PRODUCT_LINE"
}
```

Output JSON data

```
{
  "id": "<idJoin>"
}
```

The physical_association command creates a physical association.

Input JSON data

```
{
  "parent": "<idJoin>",
  "rightColumn": "PRODUCT_LINE_CODE",
  "operator": "equals",
  "leftColumn": "PRODUCT_LINE_CODE"
}
```

Output JSON data

```
{
  "id": "<physical_association>"
}
```

Creating a physical join - PRODUCT_TYPE to PRODUCT

The physical_join command creates a physical join between the PRODUCT_TYPE and PRODUCT tables. The identifier contained in the response is used in subsequent commands to refer to the physical join.

Input JSON data

```
{
  "parent": "<idDimension>",
  "leftMaxCardinality": "one",
  "rightTable": "<idTableProduct>",
  "leftMinCardinality": "one",
  "rightMaxCardinality": "many",
  "leftTable": "<idTableType>",
  "rightMinCardinality": "one",
  "name": "FK_PRODUCT_PRODUCT_TYPE"
}
```

Output JSON data

```
{
  "id": "<idJoinProduct>"
}
```

The `physical_association` command creates a physical association.

Input JSON data

```
{
  "parent": "<idJoinProduct>",
  "rightColumn": "PRODUCT_TYPE_CODE",
  "operator": "equals",
  "leftColumn": "PRODUCT_TYPE_CODE"
}
```

Output JSON data

```
{
  "id": "<physical_association_2>"
}
```

Creating mappings for the level attributes

Mappings are created for the four level attributes: `Product Line Code`, `Product Line En`, `Product Type Code`, and `Product Type En`.

Creating a query item mapping - PRODUCT_LINE_CODE

The `query_item_mapping` command creates a query item mapping.

Input JSON data

```
{
  "parent": "<idDimension>",
  "columnName": "PRODUCT_LINE_CODE",
  "queryItem": "<idProductLineCode>",
  "table": "<idTableLine>"
}
```

Output JSON data

```
{
  "id": "<query_item_mapping_1>"
}
```

Creating a query item mapping - PRODUCT_LINE_EN

The query_item_mapping command creates a query item mapping.

Input JSON data

```
{
  "parent": "<idDimension>",
  "columnName": "PRODUCT_LINE_EN",
  "queryItem": "<idProductLine>",
  "table": "<idTableLine>"
}
```

Output JSON data

```
{
  "id": "<query_item_mapping_2>"
}
```

Creating a query item mapping - PRODUCT_TYPE_CODE

The query_item_mapping command creates a query item mapping.

Input JSON data

```
{
  "parent": "<idDimension>",
  "columnName": "PRODUCT_TYPE_CODE",
  "queryItem": "<idProductTypeCode>",
  "table": "<idTableType>"
}
```

Output JSON data

```
{
  "id": "<query_item_mapping_3>"
}
```

Creating a query item mapping - PRODUCT_TYPE_EN

The query_item_mapping command creates a query item mapping.

Input JSON data

```
{
  "parent": "<idDimension>",
  "columnName": "PRODUCT_TYPE_EN",
  "queryItem": "<idProductType>",
  "table": "<idTableType>"
}
```

Output JSON data

```
{
  "id": "<query_item_mapping_4>"
}
```

Creating a cube for the model

A cube is created for the model.

Creating a cube - `ProductSales`

The `cube` command creates a cube called `ProductSales`. The identifier contained in the response is used in subsequent commands to refer to the cube.

Input JSON data

```
{
  "parent": "<modelId>",
  "name": [
    {
      "locale": "en",
      "text": "ProductSales"
    }
  ]
}
```

Output JSON data

```
{
  "id": "<idCube>"
}
```

Creating a measure dimension for the cube

A measure dimension is created for the cube.

Creating a measure dimension - `Measures`

The `measure_dimension` command creates a measure dimension called `Measures` in the cube. The identifier contained in the response is used in subsequent commands to refer to the measure dimension.

Input JSON data

```
{
  "parent": "<idCube>",
  "name": [
    {
      "locale": "en",
      "text": "Measures"
    }
  ]
}
```

Output JSON data

```
{
  "id": "<idMeasureDimension>"
}
```

Creating a measure

A measure, `Quantity`, is created and made the default measure for the measure dimension. In addition, a mapping, `QUANTITY`, is created for the measure.

Creating a measure - `Quantity`

The `measure` command creates a measure called `Quantity`. The identifier contained in the response is used in subsequent commands to refer to the measure.

Input JSON data

```
{
  "parent": "<idMeasureDimension>",
  "usage": "fact",
  "regularAggregate": "sum",
  "datatype": "int32",
  "scale": "0",
  "name": [
    {
      "locale": "en",
      "text": "Quantity"
    }
  ],
  "precision": "10"
}
```

Output JSON data

```
{
  "id": "<idMeasure>"
}
```

Setting the `Quantity` measure as the default measure

The `measure_dimension` command is used to set the `Quantity` measure as the default measure for the `Measures` measure dimension.

The HTTP request header `X-HTTP-Method-Override: PUT` is used for this action and the HTTP path for the request is

```
http://<server>:<dispatcher_port>/p2pd/servlet/dispatch/FmCommand
/measure_dimension/<idMeasureDimension>
```

Input JSON data

```
{
  "defaultMeasure": "<idMeasure>"
}
```

Output JSON data

```
{

}
```

Creating a physical table - ORDER_DETAILS

The `physical_table` command creates a physical table called `ORDER_DETAILS`. The identifier contained in the response is used in subsequent commands to refer to the physical table.

Input JSON data

```
{
  "parent": "<idMeasureDimension>",
  "datasource": "<idDataSource>",
  "name": "ORDER_DETAILS"
}
```

Output JSON data

```
{
  "id": "<idTableOrderDetails>"
}
```

Creating a query item mapping - QUANTITY

The `query_item_mapping` command creates a query item mapping.

Input JSON data

```
{
  "parent": "<idMeasureDimension>",
  "columnName": "QUANTITY",
  "queryItem": "<idMeasure>",
  "table": "<idTableOrderDetails>"
}
```

Output JSON data

```
{
  "id": "<query_item_mapping_5>"
}
```

Creating a relationship between the relational dimension and the measure dimension

A relationship is created between the relational dimension and the measure dimension.

Creating a relationship - Products to Measures

The `relationship` command creates a relationship between the `Products` relational dimension and the `Measures` measure dimension.

Input JSON data

```
{
  "parent": "<idCube>",
  "leftMaxCardinality": "one",
  "leftMinCardinality": "one",
  "rightMaxCardinality": "many",
  "expression": [
    {
      "columnName": "PRODUCT_NUMBER",
      "objectRef": "<idDimension>",
      "tableName": "PRODUCT",
      "dataSourceRef": "<idDataSource>"
    },
    "=",
    {
      "columnName": "PRODUCT_NUMBER",
      "objectRef": "<idMeasureDimension>",
      "tableName": "ORDER_DETAILS",
      "dataSourceRef": "<idDataSource>"
    }
  ],
  "rightMinCardinality": "one",
  "rightObjectRef": "<idMeasureDimension>",
  "name": "Products-Measures",
  "leftObjectRef": "<idDimension>"
}
```

Output JSON data

```
{
  "id": "<relationship>"
}
```

Saving the model to the local file system

The `model_save_stream` command saves the `Project` model to the local file system. This step is optional and when the model is saved it can be opened in IBM Cognos Cube Designer.

To save the model, use the following request headers:

- `Content-Type: text/plain`
- `Cache-Control: no-cache`
- `Connection: Keep-Alive`

The HTTP path for the request is

```
http://<server>:<dispatcher_port>/p2pd/servlet/dispatch/FmCommand
/model_save_stream/<modelId>
```

Publishing, registering, and starting the cube

The cube is made available to the IBM Cognos Analytics server and is started.

Publishing the `ProductSales` cube

The `cube_deploy` command publishes the `ProductSales` cube to the content store.

Input JSON data

```
{
  "cube": "<idCube>",
  "packageName": "FMDSDKdemo",
  "refreshDataSources": "true",
  "contentManagerModelPath": "~\/folder[@name='My Folders']"
}
```

Output JSON data

```
{

}
```

Registering the ProductSales cube

The cube_register command registers the ProductSales cube with the IBM Cognos Analytics server.

Input JSON data

```
{
  "cube": "<idCube>"
}
```

Output JSON data

```
{

}
```

Starting the ProductSales cube

The cube_start command starts the ProductSales cube.

Input JSON data

```
{
  "cube": "<idCube>"
}
```

Output JSON data

```
{

}
```

Chapter 4. Virtual cube modeling using the Cognos Dynamic Cubes API

You can model virtual cubes using the IBM Cognos Dynamic Cubes API. You can create virtual cubes, and add virtual objects to a virtual cube.

The following topics illustrate the creation of a virtual cube in an existing model. Each step in the virtual cube modeling process is briefly explained, and the input and output JSON objects are displayed. The identifiers that are created by the Cognos Dynamic Cubes API are shown as variables, such as *<modelId>*. Unless otherwise specified, the HTTP POST method is used for the Cognos Dynamic Cubes API calls described in this sample program.

Creating a virtual cube

You create a virtual cube in a model that has been opened in the IBM Cognos Dynamic Cubes API. After you create a virtual cube, you then associate source cubes to the virtual cube.

Creating a virtual cube - `VirtualCube`

The `virtual_cube` command creates a virtual cube called `VirtualCube`. The identifier contained in the response is used in subsequent commands to refer to the virtual cube.

Input JSON data

```
{
  "parent":"<modelId>",
  "mergeOperator":"sum"
  "name":"VirtualCube",
}
```

Output JSON data

```
{
  "id":"<cubeIdVirtual>"
}
```

Adding source cubes

The parent model contains source cubes named `Sales` and `Forecast`, with `id`s of *<cubeIdSales>* and *<cubeIdForecast>*, respectively.

The `virtual_source` command associates the source cubes to the virtual cube. The identifier contained in the response is used in subsequent commands to refer to the source cube.

Input JSON data

```
{
  "parent":"<cubeIdVirtual>"
  "sourceObject":"<cubeIdSales>",
}
```

Output JSON data

```
{
  "id":"<vSourceSales>"
}
```

Input JSON data

```
{
  "parent":"<cubeIdVirtual>"
  "sourceObject":"<cubeIdForecast>",
}
```

Output JSON data

```
{
  "id":"<vSourceForecast>"
}
```

If one or more of the source cubes is deployed as a data source in the content store, use the `sourceName` and `sourcePath` properties to identify the cube, instead of the `sourceObject` property.

Creating a virtual measure dimension

You create a virtual measure dimension in an existing virtual cube. After creating the virtual measure dimension, you associate source measure dimensions with the virtual measure dimension.

Creating a virtual measure dimension - `Measures`

The `virtual_measure_dimension` command creates a virtual measure dimension called `Measures`. The identifier contained in the response is used in subsequent commands to refer to the virtual measure dimension.

Input JSON data

```
{
  "parent":"<cubeIdVirtual>",
  "name":"Measures"
}
```

Output JSON data

```
{
  "id":"<vMeasureDimensionId>"
}
```

Adding source measure dimensions

The parent model contains source measure dimensions with `ids` of *<measureDimensionIdSales>* (in the `Sales` cube) and *<measureDimensionIdForecast>* (in the `Forecast` cube).

The `virtual_source` command associates the source measure dimensions to the virtual cube. The parent cubes of the source measure dimensions are identified by their virtual source `ids`. The identifier contained in the response is used in subsequent commands to refer to the source measure dimension.

Input JSON data

```
{
  "parent":"<vMeasureDimensionId>",
  "name":"SalesMDSource"
  "sourceObject":"<measureDimensionIdSales>",
  "sourceParent":"<vSourceSales>",
}
```

Output JSON data

```
{
  "id":"<vDimensionMeasureSourceSales>"
}
```

Input JSON data

```
{
  "parent":"<vMeasureDimensionId>",
  "name":"Product Forecast Fact MDSource"
  "sourceObject":"<measureDimensionIdForecast>",
  "sourceParent":"<vSourceForecast>",
}
```

Output JSON data

```
{
  "id":"<vDimensionMeasureSourceForecast>"
}
```

Creating a virtual measure

You create a virtual measure in an existing virtual measure dimension. After creating the virtual measure, you associate source measures with the virtual measure.

Creating a virtual measure - `Quantity`

The `virtual_measure` command creates a virtual measure called `Quantity`. The identifier contained in the response is used in subsequent commands to refer to the virtual measure dimension.

Input JSON data

```
{
  "parent":"<vMeasureDimensionId>",
  "dataFormat":"<formatGroup><numberFormat formatType=\"numberFormat\"
     groupDelimiter=\",\" useGrouping=\"true\"/></formatGroup>",
  "name":"Quantity",
  "mergeOperator":"sum"
  "visible":"true",
}
```

Output JSON data

```
{
  "id":"<vMeasureIdQuantity>"
```

```
}
```

Adding a source dimension

The parent model contains a source measure with ids of *<measureIdSalesQuantity>* in the `SalesMDSource` source measure dimension.

The `virtual_source` command associates the source measure to the virtual cube. The parent measure dimension of the source measure is identified by its virtual source `id`. The identifier contained in the response is used in subsequent commands to refer to the source measure dimension.

Input JSON data

```
{
  "parent":"<vMeasureIdQuantity>",
  "name":"Quantity"
  "sourceObject":"<measureIdSalesQuantity>",
  "sourceParent":"<vDimensionMeasureSourceSales>",
}
```

Output JSON data

```
{
  "id":"<vMeasureSourceQuantity>"
}
```

Setting the `Quantity` virtual measure as the default virtual measure

The `virtual_measure_dimension` command is used to set the `Quantity` virtual measure as the default virtual measure for the `Measures` virtual measure dimension.

The HTTP request header `X-HTTP-Method-Override: PUT` is used for this action and the HTTP path for the request is

```
http://<server>:<dispatcher_port>/p2pd/servlet/dispatch/FmCommand
```

```
/virtual_measure_dimension/<vMeasureDimensionId>
```

Input JSON data

```
{
  "defaultVirtualMeasure": "<vMeasureIdQuantity>"
}
```

Output JSON data

```
{

}
```

Creating a virtual dimension

You create a virtual dimension in an existing virtual cube. After creating the virtual dimension, you associate source dimensions with the virtual dimension.

Creating a virtual dimension - Time

The `virtual_dimension` command creates a virtual dimension called `Time`. The identifier contained in the response is used in subsequent commands to refer to the virtual dimension.

Input JSON data

```
{
  "parent":"<cubeIdVirtual>",
  "dimensionStyle":"time",
  "name":"Time"
}
```

Output JSON data

```
{
  "id":"<vDimensionIdTime>"
}
```

Adding source dimensions

The parent model contains source dimensions with `ids` of *<dimensionIdTime>* (in the `Sales` cube) and *<dimensionIdTimeToMonth>* (in the `Forecast` cube).

The `virtual_source` command associates the source dimensions to the virtual cube. The parent cubes of the source dimensions are identified by their virtual source `ids`. The identifier contained in the response is used in subsequent commands to refer to the source dimension.

Input JSON data

```
{
  "parent":"<vDimensionIdTime>",
  "name":"Time"
  "sourceObject":"<dimensionIdTime>",
  "sourceParent":"<vSourceSales>",
}
```

Output JSON data

```
{
  "id":"<vSourceDimensionSalesTime>"
}
```

Input JSON data

```
{
  "parent":"<vDimensionIdTime>",
  "name":"Time_ to month"
  "sourceObject":"<dimensionIdTimeToMonth>",
  "sourceParent":"<vSourceForecast>",
}
```

Output JSON data

```
{
  "id":"<vSourceDimensionTimeToMonth>"
}
```

Creating a virtual hierarchy

You create a virtual hierarchy in an existing virtual dimension. After creating the virtual dimension, you associate source hierarchies with the virtual hierarchy.

Creating a virtual hierarchy - Time

The `virtual_hierarchy` command creates a virtual hierarchy called `Time`. The identifier contained in the response is used in subsequent commands to refer to the virtual hierarchy.

Input JSON data

```
{
  "parent":"<vDimensionIdTime>",
  "name":"Time"
}
```

Output JSON data

```
{
  "id":"<vHiearchyIdTime>"
}
```

Adding source hierarchies

The parent model contains source hierarchies with `id`s of *<hieararchyIdTime>* (in the `Time` source dimension) and *<hieararchyIdTimeToMonth>* (in the `Time_ to month` source dimension).

The `virtual_source` command associates the source hierarchies to the virtual hierarchy. The parent dimensions of the source hierarchies are identified by their virtual source `id`s. The identifier contained in the response is used in subsequent commands to refer to the source hierarchy.

Input JSON data

```
{
  "parent":"<vHiearchyIdTime>",
  "name":"Time"
  "sourceObject":"<hiearrarchyIdTime>",
  "sourceParent":"<vSourceDimensionSalesTime>",
}
```

Output JSON data

```
{
  "id":"<vHiearachySourceTime>"
}
```

Input JSON data

```
{
  "parent":"<vHiearchyIdTime>",
  "name":"Time1"
  "sourceObject":"<hiearchyIdTimeToMonth>",
  "sourceParent":"<vSourceDimensionTimeToMonth>",
}
```

Output JSON data

```
{
  "id":"<vHiearachySourceTimeToMonth>"
}
```

Creating a virtual level

You create a virtual level in an existing virtual hierarchy. After creating the virtual level, you associate source levels with the virtual level.

Creating a virtual level - (All)

The `virtual_level` command creates a virtual level called `(All)`. The identifier contained in the response is used in subsequent commands to refer to the virtual level.

Input JSON data

```
{
  "parent":"<vHiearchyIdTime>",
  "name":"(All)"
}
```

Output JSON data

```
{
  "id":"<vLevelIdTimeAll>"
}
```

Adding source levels

The parent model contains source levels with `id`s of *<levelIdTimeAll>* (in the `Time` hierarchy) and *<levelIdTimeToMonthAll>* (in the `Time1` hierarchy).

The `virtual_source` command associates the source levels to the virtual cube. The parent hierarchies of the source levels are identified by their virtual source `id`s. The identifier contained in the response is used in subsequent commands to refer to the source level.

Input JSON data

```
{
  "parent":"<vLevelIdTimeAll>",
  "name":"(All)"
  "sourceObject":"<levelIdTimeAll>",
  "sourceParent":"<vHiearachySourceTime>",
}
```

Output JSON data

```
{
  "id":"<vLevelSourceAll>"
}
```

Input JSON data

```
{
  "parent":"<vLevelIdTimeAll>",
  "name":"(All)1"
  "sourceObject":"<levelIdTimeToMonthAll>",
  "sourceParent":"<vHiearachySourceTimeToMonth>",
}
```

Output JSON data

```
{
  "id":"<vLevelSourceAll1>"
}
```

Chapter 5. Aggregate modeling using the Cognos Dynamic Cubes API

You can model aggregates using the IBM Cognos Dynamic Cubes API. You can create aggregates, and add aggregate objects to an aggregate.

The following topics illustrate the creation of an aggregate in an existing model. Each step in the aggregate modeling process is briefly explained, and the input and output JSON objects are displayed. The identifiers that are created by the Cognos Dynamic Cubes API are shown as variables, such as *<modelId>*. Unless otherwise specified, the HTTP POST method is used for the Cognos Dynamic Cubes API calls described here.

Creating an aggregate

You create an aggregate in a model that has been opened in the IBM Cognos Dynamic Cubes API. After you create an aggregate, you can add aggregate measures, dimensions, and other objects to it.

Creating an aggregate - `newAggregate`

The `aggregate` command creates an aggregate called `newAggregate` from the source cube with an `id` of *<cubeId>* in the model. This command is equivalent to the **New User Defined In-Memory Aggregate** command in IBM Cognos Cube Designer. The identifier contained in the response is used in subsequent commands to refer to the aggregate.

Input JSON data

```
{
  "style":"inMemory",
  "parent":"<cubeId>",
  "name":"newAggregate"
}
```

Output JSON data

```
{
  "id":"<aggregateId>"
}
```

Creating an aggregate measure

You create an aggregate measure in an existing aggregate.

Creating an aggregate measure

The `aggregate_measure` command creates an aggregate measure from the relational measure with the `id` of *<measureId>* in the source cube. The identifier contained in the response is used in subsequent commands to refer to the aggregate measure.

Input JSON data

```
{
  "parent":"<aggregateId>",
  "measure":"<measureId>"
```

```
}
```

Output JSON data

```
{
  "id":"<aggrMeasureId>"
}
```

Creating an aggregate dimension

You create an aggregate dimension in an existing aggregate.

Creating a virtual dimension - Time

The `aggregate_dimension` command creates an aggregate dimension from the relational dimension with the `id` of *<dimensionId>* in the source cube. The identifier contained in the response is used in subsequent commands to refer to the aggregate dimension.

Input JSON data

```
{
  "dimension":"<dimensionId>",
  "parent":"<aggregateId>"
}
```

Output JSON data

```
{
  "id":"<aggrDimensionId>"
}
```

Creating an aggregate hierarchy

You create an aggregate hierarchy in an existing aggregate dimension.

Creating an aggregate hierarchy

The `aggregate_hierarchy` command creates an aggregate hierarchy from the aggregate dimension previously created and the relational hierarchy with the `id` of *<hierarchyId>* in the source cube. The identifier contained in the response is used in subsequent commands to refer to the aggregate hierarchy.

Input JSON data

```
{
  "hierarchy":"<hierarchyId>",
  "parent":"<aggrDimensionId>"
}
```

Output JSON data

```
{
  "id":"<aggrHierarchyId>"
}
```

Creating an aggregate level

You create an aggregate level in an existing aggregate hierarchy.

Creating an aggregate level

The `aggregate_level` command creates a aggregate level from the aggregate hierarchy previously created and the relational level with the `id` of *<yearLevelId>* in the source cube. The identifier contained in the response is used in subsequent commands to refer to the aggregate level.

Input JSON data

```
{
  "level":"<yearLevelId>",
  "parent":"<aggrHierarchyId>"
}
```

Output JSON data

```
{
  "id":"<aggrYearLevelId>"
}
```

Chapter 6. Performing additional tasks using the Cognos Dynamic Cubes API

You can use the IBM Cognos Dynamic Cubes API to perform a number of tasks related to modeling dynamic cubes. You can perform the following tasks:

- Filter data using an aggregate slicer.
- Create calculated members and measures
- Using relative time functionality
- Create named sets
- Create parameter maps

The following topics provide instructions on how to perform these tasks using the Cognos Dynamic Cubes API.

Filter data using an aggregate slicer

You create an aggregate slicer in an in-database aggregate by using the `slices` property of the `cube` command.

To create an aggregate slicer, you create an aggregate and use the `slices` property to specify an array of slices. Each slice is expressed as a member unique name that contains a caption, the path to the slice in the **Members** folder in IBM Cognos Cube Designer, and the `id` of the corresponding level.

For example, with a cube that has an `id` of *<cubeId>* that contains a Time level named **Year** with an `id` of *<yearLevelId>*, you can create an aggregate slicer using the years 2010 and 2011 by creating an in-database aggregate, using the `cube` command, with the following JSON data.

Input JSON data

```
{
  "parent":"<cubeId>",
  "style":"aggregate",
  "aggregateOrdinal":"1",
  "name":"aggregateSlice",
  "slices":[
    {
      "caption":"2010",
      "path":"[All].[2010]",
      "member":"<yearLevelId>"
    },
    {
      "caption":"2011",
      "path":"[All].[2011]",
      "member":"<yearLevelId>"
    }
  ]
}
```

Create calculated members and measures

You use the `calculated_member` command to create calculated members and measures.

Creating a calculated member

You create a calculated member with an expression that is based on the contents of the **Members** folder beneath a model object. For instance, if you have a hierarchy that has an `id` of *<allResgionsId>* that contains a level with the `id` of *<regionId>*, you can use the `calculated_member` command with the following input to create a calculated member that concatenates the **Americas** and **Asia Pacific** regions

Input JSON data

```
{
  "parent":"<allResgionsId>",
  "name":"testExpression",
  "expression":[
    {
      "path":"[All].[710]",
      "caption":"Americas",
      "member":"<regionId>"
    },
     "||",
    {
      "path":"[All].[740]",
      "caption":"Asia Pacific",
      "member":"<regionId>"
    }
  ]
}
```

Creating a calculated measure

You create a calculated measure with an expression that is based on an existing measure. For instance, if you have a measure dimension with an `id` of *<measureDimensionId>* that contains a measure with the `id` of *<salesTargetId>*, you can use the `calculated_member` command with the following input to create a calculated measure that is equal to 1.5 times the values of the input measure.

Input JSON data

```
{
  "parent":"<measureDimensionId>",
  "expression":[
    {
      "id":"<salesTargetId>"
     },
      "* 1.5"
  ]
}
```

Use relative time functionality

You create predefined and custom relative time members in hierarchies for time-based dimensions with the `relational_hierarchy` and `relative_time_member` commands.

Auto-generating relative time members

You control the auto-generation of prior period and next period relative time members by setting the `generatePriorPeriodsMembers` and `generateNextPeriodsMembers` properties of the

relational_hierarchy command. For more information, see the topics on relative time members in the *IBM Cognos Dynamic Cubes Developer Guide*.

Creating custom relative time members

You use the relative_time_member command to create custom time members. The following examples of JSON input are based on a time-based hierarchy with an id of *<hierarchyId>*, and Month and Quarter levels with ids of *<targetId>* and *<contextId>*, respectively. For more information on the use of the properties shown, see the topics on relative time members in the *IBM Cognos Dynamic Cubes Developer Guide*.

Custom single-period definition

```
{
  "parent":"<hierarchyId>",
  "name":"testSinglePeriod",
  "contextPeriod":"<contextId>",
  "contextPeriodOffset":"-2",
  "style":"simple",
  "targetPeriod":"<targetId>",
  "targetPeriodOffset":"1"
}
```

Custom period-to-date definition

```
{
  "parent":"<hierarchyId>",
  "name":"testPeriodToDate",
  "contextPeriod":"<contextId>",
  "contextPeriodOffset":"-2",
  "style":"periodToDate",
  "targetPeriod":"<targetId>",
  "targetPeriodOffset":"1",
  "toDatePeriod":"<contextId>"
}
```

Custom life-to-date definition

```
{
  "parent":"<hierarchyId>",
  "name":"testLifeToDate",
  "isLifeToDatePeriod":"true",
  "style":"periodToDate",
  "targetPeriod":"<targetId>",
  "targetPeriodOffset":"1",
  "toDatePeriod":"<contextId>"
}
```

Custom n-period running total definition

```
{
  "parent":"<hierarchyId>",
  "name":"testRollingTotal",
  "numberOfPeriods":"10",
  "style":"rollingTotal",
  "targetPeriod":"<targetId>"
}
```

Create named sets

You use the named_set and named_set_foldercommands to create and update named sets and folders.

Creating a named set

You create a named set with an expression that is based on the contents of the **Members** folder beneath a model object. For instance, a cube with an id of *<cubeId>* contains a level with the id of *<yearLevelId>*. The named_set command is used with the following input to create a named set that contains data for the year 2012.

Input JSON data

```
{
  "parent":"<cubeId>",
  "name":"ns1",
  "expression":[
    {
       "path":"[All].[2012]",
       "caption":"2012",
       "member":"<yearLevelId>"
    }
  ]
}
```

Output JSON data

```
{
  "id":"<namedSetId1>"
}
```

Creating a named set folder

You create a named set folder under a cube. For instance, a cube has an id of *<cubeId>*. The named_set_folder command is used with the following input to create a named set folder in the cube.

Input JSON data

```
{
  "parent":"<cubeId>",
  "name":"nsFolder1"
}
```

Output JSON data

```
{
  "id":"<folderId>"
}
```

Creating another named set

You use the named_set command to create a second named set by adding 1.5 to the named set with an id of *<namedSetId1>* and contained in the named set folder with an id of *<folderId>*.

Input JSON data

```
{
  "parent":"<folderId>",
  "name":"ns2",
  "expression":[
```

```
    {
      "id":"<namedSetId1>"
    },
      " + 2"]
}
```

Output JSON data

```
{
  "id":"<namedSetId2>"
}
```

Create parameter maps

You use the `parameter_map` and `relational_parameter_map` commands to create and update parameter maps and relational parameter maps.

Creating a parameter map

You create a parameter map by specifying a model `id` along with an array of key-value pairs and a default value. In this example you create a parameter map in a model with an `id` of *<modelId>* with 2 key-value pairs and a default value.

Input JSON data

```
{
  "parent":"<modelId>",
  "name":"TestParameterMap",
  "entries":[
    {
      "key":"a",
      "value":"1"
    },
    {
      "key":"b",
      "value":"2"
    }
  ],
  "defaultValue":"ABC"
}
```

Creating a relational parameter map

You create a relational parameter map by specifying a model `id` along with a default value. In this example you create a parameter map in a model with an `id` of *<modelId>* with a default value.

Input JSON data

```
{
  "parent":"<modelId>",
  "name":"testRelationalParameterMap",
  "defaultValue":"ABCD"
}
```

After creating a relational parameter map, you can associate query items and filters with it. In this example, you create a query item (using the `query_item` command) associated with the relational parameter map with an `id` of *<relationalParameterMapId>*.

Input JSON data

```
{
  "parent":"<relationalParameterMapId>",
  "name":"testQueryItem"
}
```

Chapter 7. Cognos Dynamic Cubes command reference

There are two types of commands used in the IBM Cognos Dynamic Cube API to create and manipulate dynamic cubes, control commands and model commands.

Control commands
These commands control model and cube lifecycle functions and miscellaneous tasks. These commands use the HTTP POST method.

Model commands
These commands are used on specific Cognos Dynamic Cubes objects, such as models, cubes, dimensions, and measures. These commands can be used to do the following actions:

- Create an object with the HTTP POST method.
- Retrieve the properties of an object with the HTTP GET method.
- Update the properties of an object with the HTTP PUT method.
- Delete an object with the HTTP DELETE method.

Control commands

Control commands control model and cube lifecycle functions and also run miscellaneous tasks.

All control commands use the HTTP POST method. Some control commands require an *<object_id>* on the HTTP path. The documentation for each control command describes the function of the command, and also describes the *<object_id>* (if required) and the input and output JSON objects.

authenticate

Sets or clears CAM passport information for the model.

<object_id>
`id` of the model.

Input JSON object

`passport`
Specifies the CAM passport that is to be set for the session.

The CAM passport is the `id` property of the `CAMPassport` that is part of the `biBusHeader` after a user authenticates using the IBM Cognos Software Development Kit `logon` method.

See the *IBM Cognos Software Development Kit Developer Guide* for more information on authenticating with the IBM Cognos Analytics server.

If an input JSON object is not specified, the CAM passport is cleared for the session.

Output JSON object
Empty.

cube_deploy

Publishes a cube to Content Manager.

<object_id>
Not applicable.

Input JSON object

`cube`
Specifies the `id` of the cube to be published.

contentManagerModelPath
Specifies the path to publish to in Content Manager.

packageName
Specifies the package name to be created or updated.

refreshDataSources
Specifies whether data sources are to be refreshed. Can be either `true` or `false`.

Output JSON object
Empty.

cube_register

Registers a cube with the IBM Cognos Analytics server.

<object_id>
Not applicable.

Input JSON object

cube
Specifies the `id` of the cube to register.

Output JSON object
Empty.

cube_start

Starts a cube.

<object_id>
Not applicable.

Input JSON object

cube
Specifies the `id` of the cube to start.

Output JSON object
Empty.

model_close

Closes a model.

<object_id>
Not applicable.

Input JSON object

model
Specifies the `id` of the model that is to be closed.

Output JSON object
Empty.

model_new

Creates a model either in memory or in a specified location.

<object_id>
Not applicable.

Input JSON object

filePath
If specified, store the model files in this location. The location is relative to the IBM Cognos Analytics server. Otherwise, the model is stored in memory.

name
Specifies the name of the project.

namespace
Specifies the namespace of the project.

locale
Specifies the default locale of the project.

Output JSON object

id
An identifier for the model. This identifier can be used to refer to the model in subsequent commands.

model_open

Opens a model from a file location or from a Content Manager path.

<object_id>
Not applicable.

Input JSON object

filePath
If specified, open the model files from this location. The location is relative to the IBM Cognos Analytics server.

contentManagerModelPath
If specified, open the model files from this Content Manager location.

Either `filePath` or `contentManagerModelPath` can be specified, but not both.

Output JSON object

id
An identifier for the model. This identifier can be used to refer to the model in subsequent commands.

model_open_stream

Opens a model from a stream that is passed in the HTTP request.

<object_id>
Not applicable.

Input stream
Specifies the input stream that consists of the model to open.

Output JSON object
Empty.

model_save

Saves an open model.

<object_id>
Not applicable.

Input JSON object

model
Specifies the `id` of the model to be saved.

Output JSON object
Empty.

model_save_as

Saves a model in a new location.

<object_id>
Not applicable.

Input JSON object

model
Specifies the id of the model that is to be saved.

filePath
Specifies the location to store the model files. The location is relative to the IBM Cognos Analytics server.

Output JSON object
Empty.

model_save_stream

Saves a model in the output stream of the HTTP response.

The HTTP request must include a `Content-Type: text/plain` request header.

<object_id>
id of the model.

Input
Not applicable.

Output
The model that is being saved.

search

Retrieves the id of an object that is based on a search path in a model.

<object_id>
id of the model to be searched.

Input JSON object

path
Specifies the path to an object, such as `[Model].[Products]`. These paths can be found in the model file (.fmd).

Output JSON object

id
The id property of the object found.

type
The type of the object found. For example, `relational_dimension`.

Model commands

Model commands are used on specific IBM Cognos Dynamic Cubes objects, such as models, cubes, dimensions, and measures. These commands can be used to create and delete objects, as well as to retrieve and modify object properties.

Detailed documentation for objects and object properties is available in the *IBM Cognos Dynamic Cubes User Guide* and in the model schema reference in the *IBM Cognos Software Development Kit Framework Manager Developer Guide* and is not repeated here.

The HTTP request and response objects vary depending on which of the four HTTP methods are being used.

HTTP POST method

Model commands use the HTTP POST method to create objects.

<object_id>
Not applicable.

Input JSON object

parent
Specifies the id of the parent object of the object that is being created.

other parameters
Extra parameters are specified depending on the type of object that is being created. These parameters are described in the documentation for each model command.

Output JSON object

id
An identifier for the object. This identifier can be used to refer to the object in subsequent commands.

HTTP GET method

Model commands use the HTTP GET method to retrieve the properties of an object.

<object_id>
Specifies the id of the object whose properties are being retrieved.

Input JSON object
Not applicable.

Output JSON object
Contains property names and values for the object. These properties vary depending on the type of object and are described in the documentation for each model command.

HTTP PUT method

Model commands use the HTTP PUT method to update one or more of the properties of an object.

<object_id>
Specifies the id of the object whose properties are being updated.

Input JSON object
Contains property names and values that are being updated for the object. The properties that can be updated vary depending on the type of object and are described in the documentation for each model command.

Output JSON object
Empty.

HTTP DELETE method

Model commands use the HTTP DELETE method to delete an object.

<object_id>
Specifies the id of the object to delete.

Input JSON object
Not applicable.

Output JSON object
Empty.

The documentation for each model command describes which properties are applicable for the POST, GET, and PUT methods.

aggregate

The `aggregate` command creates and deletes aggregates, and also updates and retrieves the properties of an aggregate.

The following table lists the properties of an aggregate, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 1: Aggregate properties for input and output methods

Name	Description	HTTP methods
`parent`	`id` of the parent cube.	`POST`
`dimensions`	Array of `id` values of child aggregate dimensions.	`GET`
`measures`	Array of `id` values of child aggregate measures.	`GET`

`name`	Name.	`POST`
		`GET`
		`PUT`

`style`	Aggregate style. Can be `inDatabase` or `inMemory`. **Important:** Only `inMemory` is currently supported. To create an in-database aggregate, create a cube with the `aggregate` style.	`POST` `GET`

aggregate_dimension

The `aggregate_dimension` command creates and deletes aggregate dimensions, and also updates and retrieves the properties of an aggregate dimension.

The following table lists the properties of an aggregate dimension, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 2: Aggregate dimension properties for input and output methods

Name	Description	HTTP methods
`parent`	`id` of the parent aggregate.	`POST`
`aggregateHierarchies`	Array of `id` values of child aggregate hierarchies.	`GET`
`dimension`	`id` of the relational dimension used to create this aggregate dimension.	`POST` `GET`

aggregate_hierarchy

The `aggregate_hierarchy` command creates and deletes aggregate hierarchies, and also updates and retrieves the properties of an aggregate hierarchy.

The following table lists the properties of an aggregate hierarchy, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 3: Aggregate hierarchy properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent aggregate dimension.	POST
`aggregateLevels`	Array of `id` values of child aggregate levels.	GET
`hierarchy`	`id` of the relational hierarchy used to create this aggregate hierarchy.	POST GET

aggregate_level

The `aggregate_level` command creates and deletes aggregate levels, and also updates and retrieves the properties of an aggregate level.

The following table lists the properties of an aggregate level, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 4: Aggregate level properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent aggregate hierarchy.	POST
`hierarchy`	`guid` of the relational hierarchy used to create this aggregate level.	GET
`level`	`id` of the relational level used to create this aggregate level.	POST GET

aggregate_measure

The `aggregate_measure` command creates and deletes aggregate measures, and also updates and retrieves the properties of a aggregate measure.

The following table lists the properties of an aggregate measure, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 5: Aggregate measure properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent aggregate.	POST
`measure`	`id` of the measure used to create this aggregate measure.	POST GET

calculated_member

The `calculated_member` command creates and deletes calculated members and measures , and also updates and retrieves the properties of a calculated member or measure.

The following table lists the properties of a calculated member describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 6: Calculated member properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent relational hierarchy.	`POST`

`comment`	Comment.	`POST`
		`GET`
		`PUT`

`description`	Description. Localized text.	`POST`
		`GET`
		`PUT`

`name`	Name. Localized text.	`POST`
		`GET`
		`PUT`

`screenTip`	Screen tip. Localized text.	`POST`
		`GET`
		`PUT`

`expression`	Expression that defines the calculated member or measure.	`POST` `GET`

cube

The `cube` command creates and deletes cubes, and also updates and retrieves the properties of a cube.

The following table lists the properties of a cube, describes each of them, and specifies whether each property is applicable in an HTTP `POST` or `PUT` method (as an input), or in an HTTP `GET` method (as an output).

Table 7: Cube properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent model or namespace.	`POST`
`accessRules`	Array of `id` values of child security filters.	`GET`

`aggregateOrdinal`	Ordinal, only applicable for an aggregate.	`POST`
		`GET`
		`PUT`

`comment`	Comment.	`POST`
		`GET`
		`PUT`

`description`	Description. Localized text.	`POST`
		`GET`
		`PUT`

Table 7: Cube properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**
`dimensions`	Array of `id` values of child relational dimensions.	`GET`
`inDatabaseAggregates`	Array of `id` values of child in-database aggregates.	`GET`
`inMemoryAggregates`	Array of `id` values of child in-memory aggregates.	`GET`
`measureDimension`	`id` of the child measure dimension.	`GET`

`name`	Name. Localized text.	`POST`
		`GET`
		`PUT`

`namedSets`	Array of `id` values of child named sets.	`GET`
`namedSetFolders`	Array of `id` values of child named set folders.	`GET`
`relationships`	Array of `id` values of child relationships.	`GET`

`removeNonExistentTuples`	Remove non-existent tuples. Can be either `true` (default) or `false`.	`POST`
		`GET`
		`PUT`

`screenTip`	Screen tip. Localized text.	`POST`
		`GET`
		`PUT`

slices	Array of member unique names of child slices. For more information, see Filter data using an aggregate slicer.	`POST`
		`GET`
		`PUT`

`style`	Cube style. Can be `regular`, `aggregate`, or `virtual`. Using the `aggregate` style creates an in-database aggregate. To create an in-memory aggregate, create an aggregate.	`POST` `GET`

datasource

The `datasource` command creates and deletes data sources, and also updates and retrieves the properties of a data source.

The following table lists the properties of a data source, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 8: Data source properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent model.	`POST`

Table 8: Data source properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**

`catalog`	Catalog name. Required when you are importing metadata from a Content Manager data source.	`POST`
		`GET`
		`PUT`

`cmDataSource`	Content Manager data source name. Required when you are importing metadata from a Content Manager data source.	`POST`
		`GET`
		`PUT`

`cube`	Cube name. Required when you are importing metadata from an IBM InfoSphere® Warehouse Cubing Services cube	`POST`
		`GET`
		`PUT`

`functionSetId`	Set of functions available in this data source.	`POST`
		`GET`
		`PUT`

`interface`	Interface.	`POST`
		`GET`
		`PUT`

`queryProcessing`	Query processing, such as `limitedLocal` or `databaseOnly`.	`POST`
		`GET`
		`PUT`

`queryType`	Query type, such as `relational` or `multidimensional`.	`POST`
		`GET`
		`PUT`

`name`	Name.	`POST`
		`GET`
		`PUT`

`rollupProcessing`	Rollup processing, such as `unspecified`, `local`, `database`, or `extended`.	`POST`
		`GET`
		`PUT`

`schema`	Schema name. Required when you are importing metadata from a Content Manager data source.	`POST`
		`GET`
		`PUT`

folder

The `folder` command creates and deletes folders, and also updates and retrieves the properties of a folder.

The following table lists the properties of a folder, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 9: Folder properties for input and output methods

Name	Description	HTTP methods
`parent`	`id` of the parent model or folder.	POST
`comment`	Comment.	POST GET PUT
`cubes`	Array of `id` values of child cubes.	GET
`description`	Description. Localized text.	POST GET PUT
`dimensions`	Array of `id` values of child relational dimensions.	GET
`folders`	Array of `id` values of child relational folders.	GET
`name`	Name. Localized text.	POST GET PUT
`namespaces`	Array of `id` values of child namespaces.	GET
`querySubjects`	Array of `id` values of child query subjects.	POST GET PUT
`screenTip`	Screen tip. Localized text.	POST GET PUT
`virtualCubes`	Array of `id` values of child virtual cubes.	GET

level

The `level` command creates and deletes levels, and also updates and retrieves the properties of a level.

The following table lists the properties of a level, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 10: Level properties for input and output methods

Name	Description	HTTP methods
`parent`	`id` of the parent relational dimension or relational hierarchy.	POST

Table 10: Level properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**

`comment`	Comment.	`POST`
		`GET`
		`PUT`

`currentPeriod`	Current period.	`POST`
		`GET`
		`PUT`

`description`	Description. Localized text.	`POST`
		`GET`
		`PUT`

`isUnique`	Indicates that level members can be uniquely identified by business keys. Can be `true` or `false`.	`POST`
		`GET`
		`PUT`

`levelKeys`	Array of `id` values of child query items that are level keys.	`GET`

`levelType`	Level type.	`POST`
		`GET`
		`PUT`

`name`	Name. Localized text.	`POST`
		`GET`
		`PUT`

`screenTip`	Screen tip. Localized text.	`POST`
		`GET`
		`PUT`

measure

The `measure` command creates and deletes measures, and also updates and retrieves the properties of a measure.

The following table lists the properties of a measure, describes each of them, and specifies whether each property is applicable in an HTTP `POST` or `PUT` method (as an input), or in an HTTP `GET` method (as an output).

Table 11: Measure properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent measure dimension or measure folder.	`POST`

`comment`	Comment.	`POST`
		`GET`
		`PUT`

Table 11: Measure properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**

`currency`	The ISO currency code.	`POST`
		`GET`
		`PUT`

`datatype`	Data type. See the model schema reference in the *IBM Cognos Software Development Kit Framework Manager Developer Guide* for allowable values.	`POST`
		`GET`
		`PUT`

`description`	Description. Localized text.	`POST`
		`GET`
		`PUT`

`format`	Data format.	`POST`
		`GET`
		`PUT`

`isHidden`	Inverse of visible. Can be `true` or `false`.	`POST`
		`GET`
		`PUT`

`isUnsortable`	Indicates that data values for this object can be sorted or compared. Can be `true` or `false`.	`POST`
		`GET`
		`PUT`

`name`	Name. Localized text.	`POST`
		`GET`
		`PUT`

`precision`	Precision.	`POST`
		`GET`
		`PUT`

`promptType`	Prompt type. See the model schema reference in the *IBM Cognos Software Development Kit Framework Manager Developer Guide* for allowable values.	`POST`
		`GET`
		`PUT`

`regularAggregate`	Regular aggregate. See the model schema reference in the *IBM Cognos Software Development Kit Framework Manager Developer Guide* for allowable values.	`POST`
		`GET`
		`PUT`

`roles`	Array of `id` values of child query item roles.	`GET`

`scale`	Scale.	`POST`
		`GET`
		`PUT`

Table 11: Measure properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**

`screenTip`	Screen tip. Localized text.	`POST`
		`GET`
		`PUT`

`size`	Maximum size of a value in bytes.	`POST`
		`GET`
		`PUT`

`usage`	Usage type. See the model schema reference in the *IBM Cognos Software Development Kit Framework Manager Developer Guide* for allowable values.	`POST`
		`GET`
		`PUT`

measure_dimension

The `measure_dimension` command creates and deletes measure dimensions, and also updates and retrieves the properties of a measure dimension.

The following table lists the properties of a measure dimension, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 12: Measure dimension properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent model.	`POST`
`calculatedMeasures`	Array of `id` values of child calculated measures.	`GET`

`comment`	Comment.	`POST`
		`GET`
		`PUT`

`defaultMeasure`	`id` of the default measure or calculated measure. The measure or calculated measure must exist.	`GET` `PUT`

`description`	Description. Localized text.	`POST`
		`GET`
		`PUT`

`filters`	Array of `id` values of child relational filters.	`GET`
`folders`	Array of `id` values of child measure folders.	`GET`
`measures`	Array of `id` values of child measures.	`GET`

`name`	Name. Localized text.	`POST`
		`GET`
		`PUT`

Table 12: Measure dimension properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**
screenTip	Screen tip. Localized text.	POST GET PUT

measure_folder

The measure_folder command creates and deletes measure folders, and also updates and retrieves the properties of a measure folder.

The following table lists the properties of a measure folder, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 13: Measure folder properties for input and output methods		
Name	**Description**	**HTTP methods**
parent	id of the parent measure dimension or measure folder.	POST
calculatedMeasures	Array of id values of child calculated measures.	GET
comment	Comment.	POST GET PUT
description	Description. Localized text.	POST GET PUT
folders	Array of id values of child measure folders.	GET
measures	Array of id values of child measures.	GET
name	Name. Localized text.	POST GET PUT
screenTip	Screen tip. Localized text.	POST GET PUT

model

The model command deletes models, and also updates and retrieves the properties of a model.

To create a new model, use the model_new command.

The following table lists the properties of a model, describes each of them, and specifies whether each property is applicable in an HTTP PUT method (as an input), or in an HTTP GET method (as an output).

Table 14: Model properties for input and output methods		
Name	**Description**	**HTTP methods**
comment	Comment.	GET PUT
cubes	Array of id values of child cubes.	GET
datasources	Array of id values of child data sources.	GET
defaultLocale	Default locale.	GET
description	Description. Localized text.	GET PUT
dimensions	Array of id values of child relational dimensions.	GET
folders	Array of id values of child folders.	GET
locales	Array of locales.	GET
name	Name. Localized text.	GET PUT
namespaces	Array of id values of child namespaces.	GET
parameterMaps	Array of id values of child parameter maps.	GET
projectName	Name of the project.	GET
querySubjects	Array of id values of child relational query subjects.	GET
screenTip	Screen tip. Localized text.	POST GET PUT
virtualCubes	Array of id values of child virtual cubes.	GET

named_set

The named_set command creates and deletes named sets, and also updates and retrieves the properties of a named set.

The following table lists the properties of a named set, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 15: Named set properties for input and output methods		
Name	**Description**	**HTTP methods**
parent	id of the parent cube or named set folder.	POST
comment	Comment.	POST GET PUT

Table 15: Named set properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**

`description`	Description. Localized text.	`POST`
		`GET`
		`PUT`

`expression`	Expression that defines the named set.	`POST` `GET`

`name`	Name. Localized text.	`POST`
		`GET`
		`PUT`

`screenTip`	Screen tip. Localized text.	`POST`
		`GET`
		`PUT`

named_set_folder

The `named_set_folder` command creates and deletes named set folders, and also updates and retrieves the properties of a named set folder.

The following table lists the properties of a named set folder, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 16: Named set folder properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent cube or named set folder.	`POST`
`namedSets`	Array of `id` values of child named sets.	`GET`

`namedSetFolders`	Array of `id` values of child named set folders.	`POST`
		`GET`
		`PUT`

`comment`	Comment.	`POST`
		`GET`
		`PUT`

`description`	Description. Localized text.	`POST`
		`GET`
		`PUT`

`name`	Name. Localized text.	`POST`
		`GET`
		`PUT`

Table 16: Named set folder properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**
screenTip	Screen tip. Localized text.	POST GET PUT

namespace

The namespace command creates and deletes namespaces, and also updates and retrieves the properties of a namespace.

The following table lists the properties of a namespace, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 17: Namespace properties for input and output methods		
Name	**Description**	**HTTP methods**
parent	id of the parent model, namespace, or folder.	POST
comment	Comment.	POST GET PUT
cubes	Array of id values of child cubes.	GET
description	Description. Localized text.	POST GET PUT
dimensions	Array of id values of child relational dimensions.	GET
folders	Array of id values of child folders.	GET
name	Name. Localized text.	POST GET PUT
namespaces	Array of id values of child namespaces.	GET
querySubjects	Array of id values of child relational query subjects.	GET
screenTip	Screen tip. Localized text.	POST GET PUT
virtualCubes	Array of id values of child virtual cubes.	GET

parameter_map

The `parameter_map` command creates and deletes parameter maps, and also updates and retrieves the properties of a parameter map.

The following table lists the properties of a parameter map, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 18: Parameter map properties for input and output methods

Name	Description	HTTP methods
`parent`	`id` of the parent model.	POST
`entries`	Array of key-value pairs.	POST GET PUT
`defaultValue`	Default value.	POST GET PUT
`name`	Name.	POST GET PUT

physical_association

The `physical_association` command creates and deletes physical associations, and also to update and retrieve the properties of a physical association.

The following table lists the properties of a physical association, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 19: Physical association properties for input and output methods

Name	Description	HTTP methods
`parent`	`id` of the parent physical join.	POST
`leftColumn`	Name of the left column.	POST GET
`operator`	Operator that is used to join left and right columns. Can be one of `none`, `equals`, `notEquals`, `lessThan`, `greaterThan`, `lessThanOrEquals`, or `greaterThanOrEquals`.	POST GET PUT
`rightColumn`	Name of the right column.	POST GET

physical_join

The physical_join command creates and deletes physical joins, and also updates and retrieves the properties of a physical join.

The following table lists the properties of a physical join, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 20: Physical join properties for input and output methods

Name	Description	HTTP methods
parent	id of the parent relational dimension or relational query subject.	POST
associations	Array of id values of child physical associations.	GET
leftMaxCardinality	Maximum cardinality of the left-side physical table. Can be one or many.	POST GET PUT
leftMinCardinality	Minimum cardinality of the left-side physical table. Can be one or many.	POST GET PUT
leftTable	id of the left-side physical table.	POST GET
name	Name.	POST GET PUT
rightMaxCardinality	Maximum cardinality of the right-side physical table. Can be one or many.	POST GET PUT
rightMinCardinality	Minimum cardinality of the right-side physical table. Can be one or many.	POST GET PUT
rightTable	id of the right-side physical table.	POST GET

physical_table

The physical_table method creates and deletes physical tables, and also updates and retrieves the properties of a physical table.

The following table lists the properties of a physical table, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 21: Physical table properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent relational dimension or relational query subject.	POST
`datasource`	`id` of the data source that contains the physical table.	POST GET PUT
`name`	Name.	POST GET PUT

query_item

The `query_item` command creates and deletes query items, and also updates and retrieves the properties of a query item.

The following table lists the properties of a query item, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 22: Query item properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent level, relational query subject, or relational parameter map.	POST
`currency`	The ISO currency code.	POST GET PUT
`datatype`	Data type. See the model schema reference in the *IBM Cognos Software Development Kit Framework Manager Developer Guide* for allowable values.	POST GET PUT
`format`	Data format.	POST GET PUT
`isHidden`	Inverse of visible. Can be `true` or `false`	POST GET PUT
`isLevelKey`	Indicates that this query item is a level key. Can be `true` or `false`	POST GET PUT
`isUnsortable`	Indicates that data values for this object can be sorted or compared. Can be `true` or `false`.	POST GET PUT

Table 22: Query item properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**

`precision`	Precision.	`POST`
		`GET`
		`PUT`

`promptType`	Prompt type. See the model schema reference in the *IBM Cognos Software Development Kit Framework Manager Developer Guide* for allowable values.	`POST`
		`GET`
		`PUT`

`regularAggregate`	Regular aggregate. See the model schema reference in the *IBM Cognos Software Development Kit Framework Manager Developer Guide* for allowable values.	`POST`
		`GET`
		`PUT`

`roles`	Array of `id` values of child query item roles.	`GET`

`scale`	Scale.	`POST`
		`GET`
		`PUT`

`size`	Maximum size of a value in bytes.	`POST`
		`GET`
		`PUT`

`usage`	Usage type. See the model schema reference in the *IBM Cognos Software Development Kit Framework Manager Developer Guide* for allowable values.	`POST`
		`GET`
		`PUT`

query_item_mapping

The `query_item_mapping` command creates and deletes query item mappings, and also updates and retrieves the properties of a query item mapping.

The following table lists the properties of a query item mapping, describes each of them, and specifies whether each property is applicable in an HTTP `POST` or `PUT` method (as an input), or in an HTTP `GET` method (as an output).

Table 23: Query item mapping properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent relational dimension or relational query subject.	`POST`
`columnName`	Column name.	`POST` `GET`
`queryItem`	`id` of the query item.	`POST` `GET`

Table 23: Query item mapping properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**
`sqlObject`	`id` of the SQL object.	`POST` `GET`
`table`	`id` of the physical table.	`POST` `GET`

query_item_role

The `query_item_role` command creates and deletes query item roles, and also updates and retrieves the properties of a query item role.

The following table lists the properties of a query item role, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 24: Query item role properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent query item or measure.	POST
`isIntristic`	If set to `true`, indicates that the attribute with this role is not displayed in the studios but is available by using the roleValue function. If more than one role is specified, the intrinsic attribute is ANDed for all roles. The default value is `false`.	GET

`name`	Name. Localized text.	`POST`
		`GET`
		`PUT`

relational_dimension

The `relational_dimension` command creates and deletes relational dimensions, and also to update and retrieve the properties of a relational dimension.

The following table lists the properties of a relational dimension, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 25: Relational dimension properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent model, cube, or folder.	POST

`comment`	Comment.	`POST`
		`GET`
		`PUT`

`createRelationship`	Specifies whether a simple relationship with no expression is created between the dimension and the measure dimension of the parent cube, if any. Can be `true` or `false`.	`POST` `GET`

Table 25: Relational dimension properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**
`defaultHierarchy`	`id` of the default hierarchy.	`GET`

`description`	Description. Localized text.	`POST`
		`GET`
		`PUT`

`dimensionStyle`	Dimension type. Can be `regular`, `measure`, or `time`.	`GET`
`filters`	Array of `id` values of child relational filters.	`GET`
`levels`	Array of `id` values of child levels.	`GET`

`multilingualSupport`	Multilingual support. Can be `disabled`, `byRow`, or `byColumn`.	`POST`
		`GET`
		`PUT`

`name`	Name. Localized text.	`POST`
		`GET`
		`PUT`

`relationalHierarchies`	Array of `id` values of child relational hierarchies.	`GET`

`screenTip`	Screen tip. Localized text.	`POST`
		`GET`
		`PUT`

relational_filter

The `relational_filter` command creates and deletes relational filters, and also updates and retrieves the properties of a relational filter.

The following table lists the properties of a relational filter, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 26: Relational filter properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent relational dimension, relational measure, or relational parameter map.	`POST`

`expression`	Expression that defines the filter value.	`POST`
		`GET`
		`PUT`

`generateKeyFilter`	Generate a measure dimension filter. Can be `true` or `false`.	`POST`
		`GET`
		`PUT`

Table 26: Relational filter properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**
name	Name.	POST GET PUT

relational_hierarchy

The relational_hierarchy command creates and deletes relational hierarchies, and also to update and retrieve the properties of a relational hierarchy.

The following table lists the properties of a relational hierarchy, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 27: Relational hierarchy properties for input and output methods		
Name	**Description**	**HTTP methods**
parent	id of the parent relational dimension.	POST
accessRules	Array of id values of child security filters.	GET
balanced	Balanced hierarchy. Can be true or false.	POST GET PUT
calculatedMembers	Array of id values of child calculated members.	GET
captionForMembers	Caption of padding members. Can be empty or parent.	POST GET PUT
comment	Comment.	POST GET PUT
defaultHierarchy	Specifies that this hierarchy is the default hierarchy. Can be true or false.	POST GET PUT
description	Description. Localized text.	POST GET PUT
generateNextPeriodsMembers	Specifies whether next-period time members should be auto-generated. Can be true or false.	POST GET PUT

Table 27: Relational hierarchy properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**

`generatePriorPeriodsMembers`	Specifies whether prior-period time members should be auto-generated. Can be `true` or `false`.	`POST`
		`GET`
		`PUT`

`hasRelativeTimeMembers`	Add relative time members. Can be `true` or `false`.	`POST`
		`GET`
		`PUT`

`includeRelativeTimeSubtree`	Specifies whether the relative time members sub-tree should be displayed in IBM Cognos Cube Designer. Can be `true` or `false`.	`POST`
		`GET`
		`PUT`

`isParentChild`	Parent-Child. Can be `true` or `false`.	`POST` `GET`

`levels`	Array of `id` values of child levels.	`POST`
		`GET`
		`PUT`

`multiRoot`	Multiple root members. Can be `true` or `false`.	`POST`
		`GET`
		`PUT`

`name`	Name. Localized text.	`POST`
		`GET`
		`PUT`

`ragged`	Ragged hierarchy. Can be `true` or `false`.	`POST`
		`GET`
		`PUT`

`relativeTimeMembers`	Array of `id` values of child relative time members.	`POST`
		`GET`
		`PUT`

`rootCaption`	Root caption. Localized text	`POST`
		`GET`
		`PUT`

`rootMember`	Specifies the external name of the root member for a hierarchy as captured from the data source. This property is only applicable to OLAP sources.	`POST`
		`GET`
		`PUT`

Table 27: Relational hierarchy properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**
screenTip	Screen tip. Localized text.	POST GET PUT

relational_parameter_map

The relational_parameter_map command creates and deletes relational parameter maps, and also updates and retrieves the properties of a relational parameter map.

The following table lists the properties of a relational parameter map, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 28: Relational parameter map properties for input and output methods		
Name	**Description**	**HTTP methods**
parent	id of the parent model.	POST
name	Name.	POST GET PUT
filters	Array of id values of child relational filters.	GET
queryItems	Array of id values of child query items.	GET
defaultValue	Default value.	POST GET PUT

relational_query_subject

The relational_query_subject command is used to create and delete relational query subjects, and also to update and retrieve the properties of a relational query subject.

For more information about relational query subjects, see the topics on query subjects in the *IBM Cognos Framework Manager User Guide*.

The following table lists the properties of a relational query subject, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 29: Relational query subject properties for input and output methods		
Name	**Description**	**HTTP methods**
parent	id of the parent model, namespace, or folder.	POST
comment	Comment.	POST GET PUT

Table 29: Relational query subject properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**
description	Description. Localized text.	POST GET PUT
joins	Array of id values of child physical joins.	GET
name	Name. Localized text.	POST GET PUT
queryItemMappings	Array of id values of child query item mappings.	GET
queryItems	Array of id values of child query items.	GET
screenTip	Screen tip. Localized text.	POST GET PUT
tables	Array of id values of child physical tables.	GET

relationship

The relationship command creates and deletes relationships, and also updates and retrieves the properties of a relationship.

The following table lists the properties of a relationship, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 30: Relationship properties for input and output methods		
Name	**Description**	**HTTP methods**
parent	id of the parent cube.	POST
expression	Expression joining left-side and right-side objects.	POST GET PUT
leftObjectRef	id of the left-side object.	POST GET PUT
leftMaxCardinality	Maximum cardinality of the left-side object. Can be one or many.	POST GET PUT
leftMinCardinality	Minimum cardinality of the left-side object. Can be one or many.	POST GET PUT

Table 30: Relationship properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**

`name`	Name.	`POST`
		`GET`
		`PUT`

`rightObjectRef`	`id` of the right-side object.	`POST`
		`GET`
		`PUT`

`rightMaxCardinality`	Maximum cardinality of the right-side object. Can be `one` or `many`.	`POST`
		`GET`
		`PUT`

`rightMinCardinality`	Minimum cardinality of the right-side object. Can be `one` or `many`.	`POST`
		`GET`
		`PUT`

relative_time_member

The `relative_time_member` command creates and deletes cubes, and also updates and retrieves the properties of a relative time member.

The following table lists the properties of a relative time member, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 31: Relative time member properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent hierarchy.	`POST`

`comment`	Comment.	`POST`
		`GET`
		`PUT`

`description`	Description. Localized text.	`POST`
		`GET`
		`PUT`

`name`	Name. Localized text.	`POST`
		`GET`
		`PUT`

`contextPeriod`	`id` of the level containing the context period.	`POST`
		`GET`
		`PUT`

Table 31: Relative time member properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**

`contextPeriodOffset`	Offset from the context period.	`POST`
		`GET`
		`PUT`

`isLifeToDatePeriod`	Specifies a life-to-date definition. Can be either `true` or `false` (default).	`POST`
		`GET`
		`PUT`

`numberOfPeriods`	Specifies the number of time periods	`POST`
		`GET`
		`PUT`

`targetPeriod`	`id` of the level containing the target period.	`POST`
		`GET`
		`PUT`

`targetPeriodOffset`	Offset from the target period.	`POST`
		`GET`
		`PUT`

`toDatePeriod`	`id` of the level containing the to-date period.	`POST`
		`GET`
		`PUT`

`screenTip`	Screen tip. Localized text.	`POST`
		`GET`
		`PUT`

`style`	Relative time member style. Can be `simple`, `periodToDate`, or `rollingTotal`.	`POST` `GET`

security_filter

The `security_filter` command creates and deletes security filters, and also updates and retrieves the properties of a security filter.

The following table lists the properties of a security filter, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 32: Security filter properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent relational hierarchy, cube, or security view.	POST

Table 32: Security filter properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**

`expression`	Expression of the access rule.	`POST`
		`GET`
		`PUT`

`isAllAccess`	All access. Can be `true` or `false`.	`POST` `GET`

`memberAccess`	Scope. Can be `grantAll`, `grantMembers`, `grantMembersAndDescendants`, `grantMembersAndAncestors`, `grantMembersDescendantsAndAncestors`, or `denyMembersAndDescendants`	`POST`
		`GET`
		`PUT`

`name`	Name.	`POST`
		`GET`
		`PUT`

`objectReference`	The `id` of the object that is being filtered.	`POST` `GET`
`permission`	Permission type. Can be `grant` or `deny`.	`POST` `GET`

security_view

The `security_view` command creates and deletes security views, and also updates and retrieves the properties of a security view.

The following table lists the properties of a security view, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 33: Security view properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent cube.	POST

`filters`	Array of `id`s of child security filters.	`POST`
		`GET`
		`PUT`

`name`	Name.	`POST`
		`GET`
		`PUT`

sql_object

The `sql_object` command creates and deletes SQL objects, and also to update and retrieve the properties of a SQL object.

For more information about SQL objects, see the topic on data source query subjects in the *IBM Cognos Framework Manager User Guide*.

The following table lists the properties of a SQL object, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 34: SQL object properties for input and output methods

Name	Description	HTTP methods
`parent`	`id` of the parent relational query subject.	POST
`datasource`	`id` of the data source against which the SQL object is run.	POST GET
`name`	Name.	POST GET
`sqlStatement`	The SQL statement as a string.	POST GET

virtual_cube

The `virtual_cube` command creates and deletes virtual cubes, and also updates and retrieves the properties of a virtual cube.

The following table lists the properties of a virtual cube, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 35: Virtual cube properties for input and output methods

Name	Description	HTTP methods
`parent`	`id` of the parent model or namespace.	POST
`comment`	Comment.	POST GET PUT
`description`	Description. Localized text.	POST GET PUT
`name`	Name. Localized text.	POST GET PUT
`namedSets`	Array of `id` values of child named sets.	GET
`namedSetFolders`	Array of `id` values of child named set folders.	GET

Table 35: Virtual cube properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**

`screenTip`	Screen tip. Localized text.	`POST`
		`GET`
		`PUT`

`sourceObjects`	Array of `id` values of virtual source objects.	`GET`
`virtualMeasureDimension`	`id` of the child virtual measure dimension.	`GET`
`virtualDimensions`	Array of `id` values of child virtual dimensions.	`GET`

virtual_dimension

The `virtual_dimension` command creates and deletes virtual dimensions, and also updates and retrieves the properties of a virtual dimension.

The following table lists the properties of a virtual dimension, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 36: Virtual dimension properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent virtual cube.	`POST`

`comment`	Comment.	`POST`
		`GET`
		`PUT`

`defaultVirtualHierarchy`	`id` of the default virtual hierarchy.	`POST` `GET`

`description`	Description. Localized text.	`POST`
		`GET`
		`PUT`

`dimensionStyle`	Dimension type. Can be `regular` or `time`.	`POST` `GET`

`name`	Name. Localized text.	`POST`
		`GET`
		`PUT`

`screenTip`	Screen tip. Localized text.	`POST`
		`GET`
		`PUT`

`sourceObjects`	Array of `id` values of virtual source objects.	`GET`

Table 36: Virtual dimension properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**
`virtualHierarchies`	Array of `id` values of child virtual hierarchies.	`GET`

virtual_hierarchy

The `virtual_hierarchy` command creates and deletes virtual hierarchies, and also updates and retrieves the properties of a virtual hierarchy.

The following table lists the properties of a virtual hierarchy, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 37: Virtual hierarchy properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent virtual dimension.	`POST`

`addRelativeTimeMembers`	Add relative time members. Can be `true` or `false`.	`POST`
		`GET`
		`PUT`

`calculatedMembers`	Array of `id` values of child calculated members.	`GET`

`comment`	Comment.	`POST`
		`GET`
		`PUT`

`description`	Description. Localized text.	`POST`
		`GET`
		`PUT`

`name`	Name. Localized text.	`POST`
		`GET`
		`PUT`

`parentChild`	Parent-Child. Can be `true` or `false`.	`POST` `GET`

`screenTip`	Screen tip. Localized text.	`POST`
		`GET`
		`PUT`

`sourceObjects`	Array of `id` values of virtual source objects.	`GET`
`virtualLevels`	Array of `id` values of child virtual levels.	`GET`

virtual_level

The `virtual_level` command creates and deletes virtual levels, and also updates and retrieves the properties of a virtual level.

The following table lists the properties of a virtual level, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 38: Virtual level properties for input and output methods

Name	Description	HTTP methods
`parent`	`id` of the parent virtual dimension or virtual hierarchy.	POST
`comment`	Comment.	POST GET PUT
`description`	Description. Localized text.	POST GET PUT
`name`	Name. Localized text.	POST GET PUT
`screenTip`	Screen tip. Localized text.	POST GET PUT
`sourceObjects`	Array of `id` values of virtual source objects.	GET

virtual_measure

The `virtual_measure` command creates and deletes virtual measures, and also updates and retrieves the properties of a virtual measure.

The following table lists the properties of a virtual measure, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 39: Virtual measure properties for input and output methods

Name	Description	HTTP methods
`parent`	`id` of the parent virtual measure dimension or virtual measure folder.	POST
`comment`	Comment.	POST GET PUT

Table 39: Virtual measure properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**

`dataFormat`	Data format.	`POST`
		`GET`
		`PUT`

`description`	Description. Localized text.	`POST`
		`GET`
		`PUT`

`mergeOperator`	Merge operator.	`POST`
		`GET`
		`PUT`

`name`	Name. Localized text.	`POST`
		`GET`
		`PUT`

`precedence`	Precedence.	`POST`
		`GET`
		`PUT`

`screenTip`	Screen tip. Localized text.	`POST`
		`GET`
		`PUT`

`sourceObjects`	Array of `id` values of virtual source objects.	`GET`

`visible`	Visible.	`POST`
		`GET`
		`PUT`

virtual_measure_dimension

The `virtual_measure_dimension` command creates and deletes virtual measure dimensions, and also updates and retrieves the properties of a virtual measure dimension.

The following table lists the properties of a virtual measure dimension, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 40: Virtual measure dimension properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent virtual cube.	POST
`calculatedMeasures`	Array of `id` values of child calculated measures.	GET

Table 40: Virtual measure dimension properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**

`comment`	Comment.	`POST`
		`GET`
		`PUT`

`defaultVirtualMeasure`	`id` of the default virtual measure or calculated measure. The virtual measure or calculated measure must exist.	`GET` `PUT`

`description`	Description. Localized text.	`POST`
		`GET`
		`PUT`

`folders`	Array of `id` values of child virtual measure folders.	`POST`
		`GET`
		`PUT`

`name`	Name. Localized text.	`POST`
		`GET`
		`PUT`

`screenTip`	Screen tip. Localized text.	`POST`
		`GET`
		`PUT`

`sourceObjects`	Array of `id` values of virtual source objects.	`GET`
`virtualMeasures`	Array of `id` values of child virtual measures.	`GET`

virtual_measure_folder

The `virtual_measure_folder` command creates and deletes virtual measure folders, and also updates and retrieves the properties of a virtual measure folder.

The following table lists the properties of a virtual measure folder, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output).

Table 41: Virtual measure folder properties for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the parent virtual measure dimension or virtual measure folder.	`POST`

`comment`	Comment.	`POST`
		`GET`
		`PUT`

Table 41: Virtual measure folder properties for input and output methods (continued)		
Name	**Description**	**HTTP methods**

`description`	Description. Localized text.	`POST`
		`GET`
		`PUT`

`folders`	Array of `id` values of child virtual measure folders.	`POST`
		`GET`
		`PUT`

`measures`	Array of `id` values of child virtual measures.	`GET`

`name`	Name. Localized text.	`POST`
		`GET`
		`PUT`

`screenTip`	Screen tip. Localized text.	`POST`
		`GET`
		`PUT`

virtual_source

The `virtual_source` command creates and deletes virtual sources, and also updates and retrieves the properties of a virtual source. Virtual sources are used to associate source objects to virtual objects.

The following tables lists the properties of a virtual source, describes each of them, and specifies whether each property is applicable in an HTTP POST or PUT method (as an input), or in an HTTP GET method (as an output). The properties differ depending on whether the virtual object is a virtual cube or a different virtual object.

Table 42: Virtual source properties (for a virtual cube) for input and output methods		
Name	**Description**	**HTTP methods**
`parent`	`id` of the virtual cube that uses this virtual source.	`POST`

`name`	Name. This is an optional property. If it is omitted, `sourceName` is used. Localized text.	`POST`
		`GET`
		`PUT`

`sourceName`	Name of the source cube. Required for objects retrieved from the content store.	`POST` `GET`
`sourceObject`	`id` of the source cube. Required for objects retrieved from within the model.	`POST` `GET`
`sourcePath`	Path of the source cube. Required for objects retrieved from the content store.	`POST` `GET`

Table 43: Virtual source properties (for a virtual object other than a virtual cube) for input and output methods

Name	Description	HTTP methods
`parent`	`id` of the virtual object that uses this virtual source.	POST
`name`	Name. This is an optional property. If it is omitted, `sourceName` is used. Localized text.	POST GET PUT
`sourceName`	Name of the source object.	GET
`sourceObject`	`id` of the source object.	POST GET
`sourceParent`	`id` of the parent object of `sourceObject`.	POST GET

Localized text

Some model attributes, such as names, tooltips, and descriptions, support localized text. You can enter the same data in multiple languages and the version that is displayed by the IBM Cognos Analytics server is determined by the locale of the user.

Attributes that support localized text are expressed as an array of localized text elements as shown in the following example.

```
"name": [
  {
    "text": "Sample",
    "locale": "en"
  },
  {
    "text": "Exemple",
    "locale": "fr"
  }  ],
```

Each text string is paired with a locale code. You can omit the array if you are adding data for a single locale only.

```
"name":
  {
    "text": "Sample",
    "locale": "en"
  },
```

If you are using the default locale, you can add the text.

```
"name": "Sample",
```

When you retrieve the attributes of an object, localized data is always output as an array that contains `text` and `locale` elements.

Expressions

Expressions are used by some model commands to define a filter or a calculation.

An expressions is an array of one or more expression parts. Each expression part can be one of the following objects:

- A column reference.
- A member unique name.
- A string, such as `"="` or `"||"`.
- An `id` of a model object, such as `{"id":"133fd4f9bf904747ba9d3f89b50e4d4b"}`.
- The XML representation of an `expression` object, as documented in the model schema reference in the *IBM Cognos Software Development Kit Framework Manager Developer Guide*. An example is `"<expression><refobj>[Model].[Time].[levels].[Year].[Current Year]<\/refobj> > 2000<\/expression>"`

Column reference

A column reference consists of the following 4 name-value pairs.

columnName
Name of the column in the data source.

dataSourceRef
`id` of the data source.

objectRef
`id` of the referred object.

tableName
Name of the table in the data source.

For an example of this type of expression, see "Creating a relationship between the relational dimension and the measure dimension" on page 21

Member unique name

A member unique name consists of the following 3 name-value pairs.

caption
A caption for the member unique name.

path
The path to the object in the **Members** folder in IBM Cognos Cube Designer.

member
The `id` of the model object referred to.

For an example of this type of expression, see "Create calculated members and measures" on page 38

Notices

This information was developed for products and services offered worldwide.

This material may be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service. This document may describe products, services, or features that are not included in the Program or license entitlement that you have purchased.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Software Group
Attention: Licensing

3755 Riverside Dr.
Ottawa, ON
K1V 1B7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Depending upon the configurations deployed, this Software Offering may use session and persistent cookies that collect each user's

- name
- user name
- password

for purposes of

- session management
- authentication
- enhanced user usability
- single sign-on configuration
- usage tracking or functional purposes other than session management, authentication, enhanced user usability and single sign-on configuration

These cookies cannot be disabled.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at https://www.ibm.com/privacy/us/en/.

Trademarks

IBM, the IBM logo and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at " Copyright and trademark information " at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies:

- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Index

Q

R

S

V