Defining DeepSee Models

Version 2017.2
2020-06-25

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)

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

Table of Contents

About This Book

This book describes, for developers, how to define basic DeepSee models: cubes and subject areas. It also describes how to define listing groups, which define listings outside of cube definitions. It includes the following sections:

- Introduction

- Basic Concepts

- Summary of Model Options

- Principles and Recommendations

- Defining Cubes

- Compiling and Building Cubes

- Defining Dimensions, Hierarchies, and Levels

- Details of Defining Levels

- Defining Properties

- Defining Measures

- Defining Listings

- Defining Listing Fields

- Defining Calculated Members

- Defining Named Sets

- Defining Subject Areas

- Defining Listing Groups

- Reference Information for Cube Classes

- Reference Information for Subject Area Classes

- Details for the Fact and Dimension Tables

For a detailed outline, see the table of contents.

The other developer books for DeepSee are as follows:

- *Getting Started with DeepSee* briefly introduces DeepSee and the tools that it provides.

- *DeepSee Developer Tutorial* guides developers through the process of creating a sample that consists of a cube, subject areas, pivot tables, and dashboards.

- *DeepSee Implementation Guide* describes how to implement DeepSee, apart from creating the model.

- *Advanced DeepSee Modeling Guide* describes how to use the more advanced and less common DeepSee modeling features: computed dimensions, unstructured data in cubes, compound cubes, cube relationships, term lists, quality measures, KPIs, plugins, and other special options.

- *Using MDX with DeepSee* introduces MDX and describes how to write MDX queries manually for use with DeepSee cubes.

- *DeepSee MDX Reference* provides reference information on MDX as supported by DeepSee.

- *Tools for Creating DeepSee Web Clients* provides information on the DeepSee JavaScript and REST APIs, which you can use to create web clients for your DeepSee applications.

The following books are for both developers and users:

- *DeepSee End User Guide* describes how to use the DeepSee User Portal and dashboards.

- *Creating DeepSee Dashboards* describes how to create and modify dashboards in DeepSee.

- *Using the DeepSee Analyzer* describes how to create and modify pivot tables, as well as use the Analyzer in general.

Also see the article *Using PMML Models in Caché*.

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

1

Introduction

This chapter introduces DeepSee models.

Be sure to consult the online *InterSystems Supported Platforms* document for this release for information on system requirements for DeepSee.

1.1 Purpose of DeepSee

InterSystems DeepSee enables you to embed business intelligence (BI) into your applications so that your users can ask and answer sophisticated questions of their data. Your application can include *dashboards* like the following example:
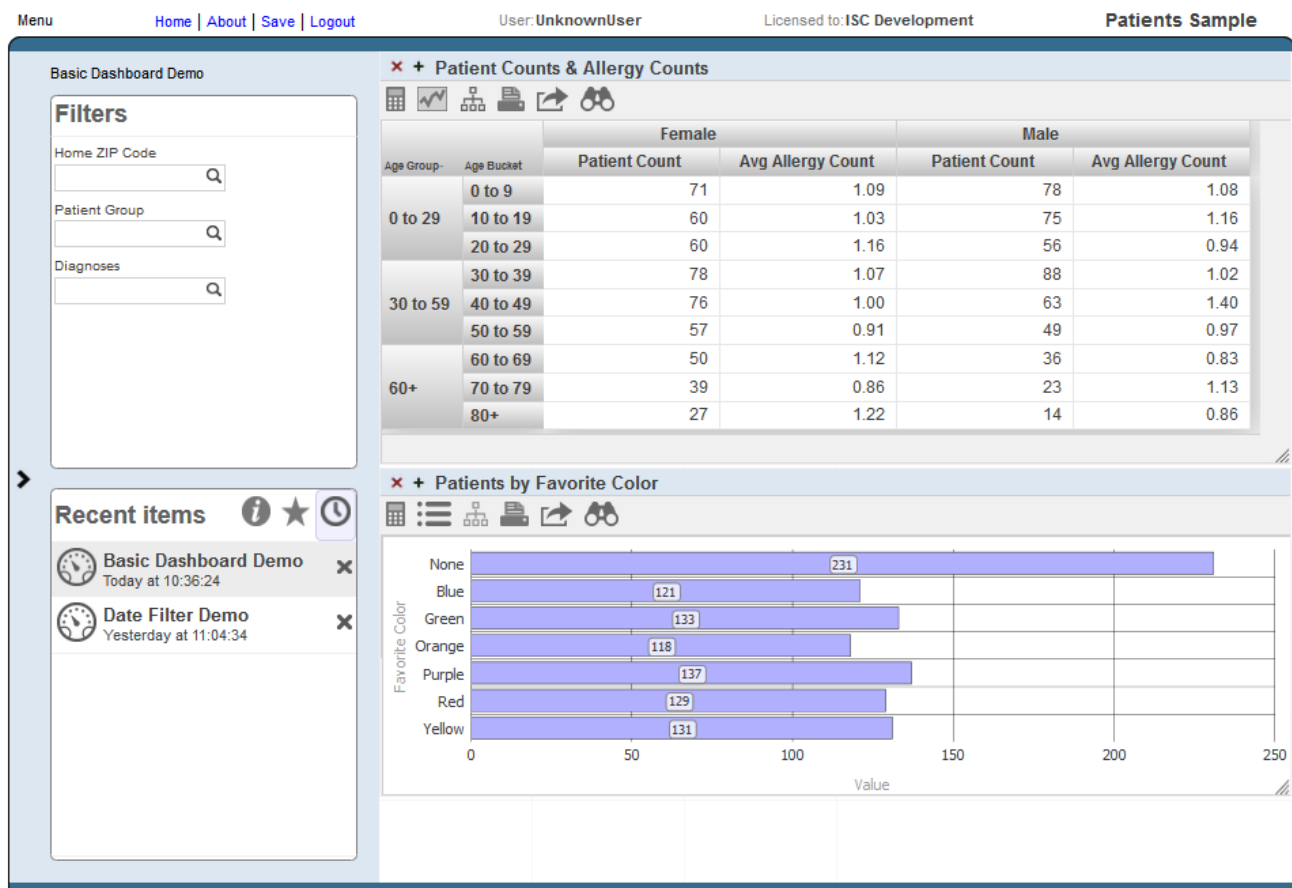
The *widgets* on a dashboard are driven by *pivot tables* and *KPIs* (key performance indicators). For a pivot table, a user can display a *listing*, which displays source values.

Pivot tables, KPIs, and listings are queries and are executed at runtime:

- A pivot table can respond to runtime input such as filter selections made by the user. Internally it uses an *MDX* (MultiDimensional eXpressions) query that communicates with a DeepSee cube.

    A *cube* consists of a *fact table* and its indices. A fact table consists of a set of *facts* (rows), and each fact corresponds to a base record. For example, the facts could represent patients or departments.

    Depending on your configuration and implementation, DeepSee detects changes in your transactional tables and propagates them to the fact tables as appropriate.

    DeepSee generates an MDX query automatically when a user creates the pivot table in the Analyzer.

- A KPI can also respond to runtime user input. Internally, it uses either an MDX query (with a DeepSee cube) or an SQL query (with any table or tables).

    In either case, you create the query manually or copy it from elsewhere.

- A listing displays selected values from the source records used for the rows of the pivot table that the user has selected. Internally, a listing is an SQL query.

    You can specify the fields to use and let DeepSee generate the actual query. Or you can specify the entire query.

1.2 Introduction to DeepSee Models

A model includes some or all of the following elements:

- At least one cube definition. A cube describes ways that you can query a set of specific base elements (such as patients or transactions). A cube includes *levels*, which enable you to group records from the base set, and *measures*, which show aggregate values of those records. It also defines listings and other items.

    When you create pivot tables, you use levels and measures. Consider the following pivot table:

Patient Group	Avg Test Score
Group A	75.08
Group B	74.22
None	

    In this pivot table, the rows correspond to the members of the `Patient Group` level; each member is shown as one row. The data column displays the aggregate value of the `Avg Test Score` measure for each of these members; for this measure, the system computes the average value. Notice that the `Avg Test Score` is null for the `None` patient group.

- Any number of subject areas. A *subject area* is a subcube that enables users to focus on smaller sets of data without the need for multiple cubes. A subject area also enables you to customize captions and defaults of the cube.

- Any number of KPIs.

    In DeepSee, a KPI is an interactive dataset that can be displayed on a dashboard. The KPI can define *actions*, which a user can launch and which execute your custom code. (Note that actions are discussed in "Defining Custom Actions" in the *DeepSee Implementation Guide*.)

You use these elements to create dashboards as follows:

- Within the DeepSee Analyzer, you create pivot tables.

  Each pivot table is a query that you create by drag and drop actions. The query runs against a cube or subject area.

- Within the Dashboard Designer, you add pivot tables and KPIs to dashboards, along with any filter controls or other needed controls.

The chapter "Basic Concepts" provides an overview of cubes and subject areas. A model can contain many additional elements. See the chapter "Summary of Model Options" for a complete comparison of the options.

1.3 Introduction to the Model Development Process

The model development process typically is as follows:

1. Create, compile, and build a basic cube with only a few items.

2. Use the Analyzer or the DeepSee shell to examine the results and to identify changes to make.

   For information on the Analyzer, see *Using the DeepSee Analyzer*.

   For information on the shell, see *Getting Started with DeepSee*.

3. Repeat the preceding steps as necessary.

4. When the cube is finalized or nearly finalized, define subject areas based on the cube.

Note that you might need multiple cubes.

1.4 Introduction to the DeepSee Architect

You use the Architect to create cubes and subject areas.

To access the Architect, do the following:

1. Click the InterSystems Launcher and then click **Management Portal**.

   Depending on your security, you may be prompted to log in with a Caché username and password.

2. Switch to the appropriate namespace as follows:

   a. Click **Switch**.

   b. Click the namespace.

   c. Click **OK**.

3. Click **DeepSee** and then click **Architect**.

   When you first display the Architect, you see the following:
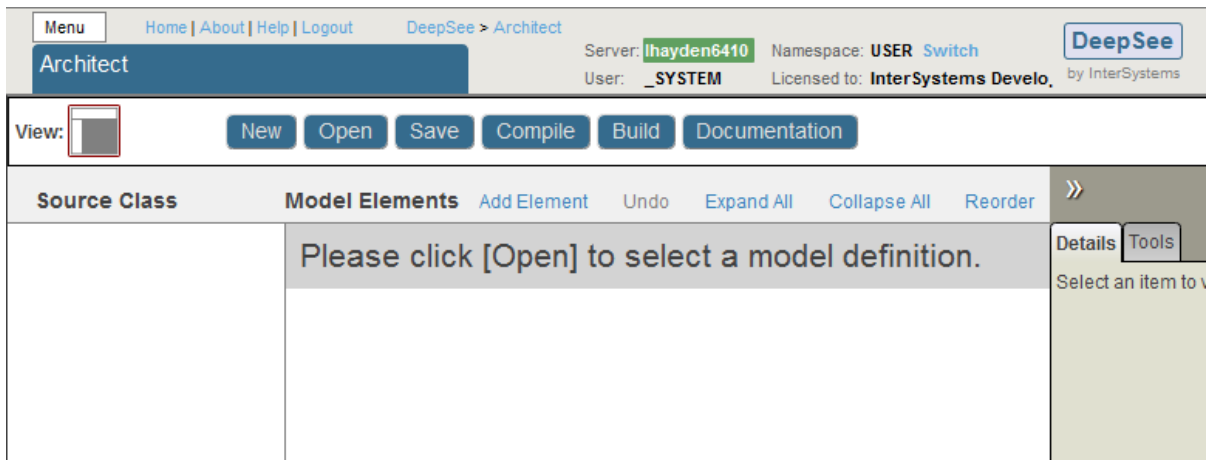
4. Click **Open**, click **Cubes**, click the name of a cube, and then click **OK**.

   Now the system displays something like the following:
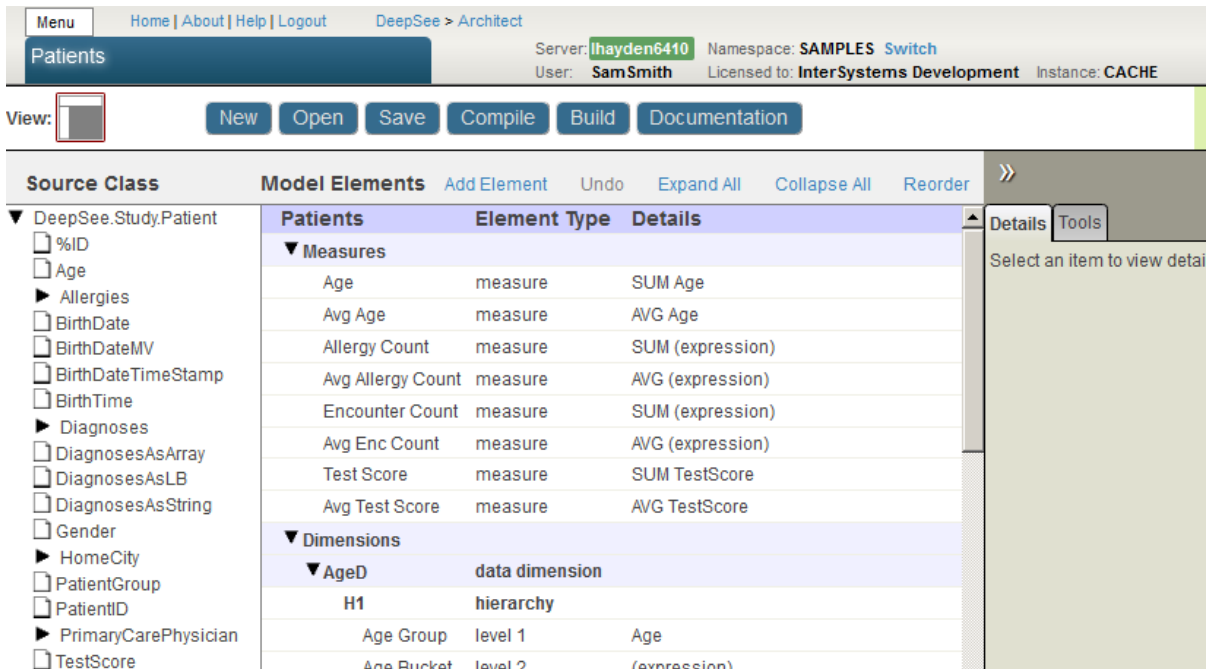
The top area contains navigation links and buttons to perform different tasks.

Below that, the page consists of the following areas, when the Architect is displaying a cube.

1.4.1 Class Viewer

The left area is the *Class Viewer*, and it shows the properties in the base class used by the cube; this area is not shown for a subject area. For example:
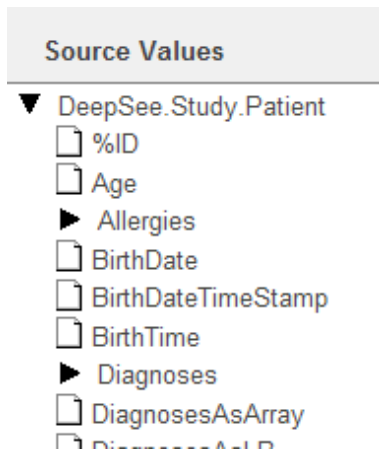
You can resize this area. To do so, drag the vertical divider on the right edge of this area.

The following rules control the display of a class in the Architect:

- All properties are shown except for relationship properties, private properties, and transient properties.

- This display is recursive; that is, properties of properties are shown.

- If a property is a collection (a list or an array) or a relationship, it is shown as a folder that displays the properties of the class used in the collection or relationship.

- If a property is of type %List (which is the object equivalent of **$LISTBUILD**), it is not shown as a folder.

  For example, see the DiagnosesAsLB property, which is included in the Patients sample to illustrate this point.

- If a class is not accessible from the base class via cascading dot syntax, it is not shown.

  For example, the Architect does not display DeepSee.Study.PatientDetails or DeepSee.Study.PatientEncounter.

- The Architect displays properties inherited from superclasses. (The sample does not demonstrate this.)

All core cube elements are based either on a source property or on a source expression (which is an ObjectScript expression) that can use properties of any class.

Important: The Architect provides a useful view of the class properties, which makes it very easy to create DeepSee elements based on those properties. It is important, however, to know that although this view provides a convenient way to access some properties, you can also use a source expression to access any data. These source expressions are evaluated when the cube is built and thus do not affect your runtime performance.

1.4.2 Model Viewer

The center area is the *Model Viewer*, and it shows the current contents of the cube. For example:

You can resize this area. To do so, drag the vertical divider on the left edge of this area.
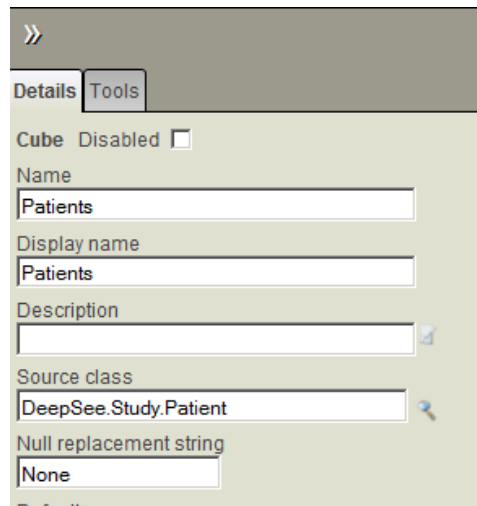
The links at the top include **Add Element**, which you can use to add measures, dimensions, and other items to the cube. In the area below the links, you can select items for editing, including the cube itself in the first row. You can also delete an item by clicking the X button in the row for that item.

1.4.3 Details Area

The right area is the *Details Area*, and it shows details for the element that is currently selected in Model Viewer (if any), or for the cube (if nothing is selected).

For example:

To hide this area, click the Hide Details button . If you do so, the Model Viewer is widened.

Then, to display this area again, click the Show Details button .

In this area, you primarily work on the **Details** tab. The **Tools** tab provides quick access to other tools, not documented this book. This tab looks like this:
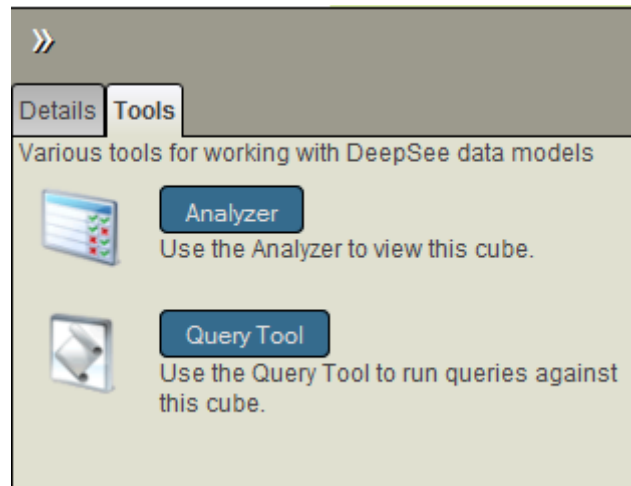
2

Basic Concepts

This chapter explains the most important concepts in DeepSee models: cubes, subject areas, and their contents. It discusses the following topics:

- Cubes
- Dimensions, hierarchies, and levels
- Properties
- Measures
- Listings
- Calculated members
- Subject areas
- Filters
- How DeepSee builds and uses fact tables
- How DeepSee generates listings

A model can contain many additional elements, which are discussed in the *Advanced DeepSee Modeling Guide*. See the next chapter for a complete comparison of the options.

2.1 Introduction to Cubes

A cube is an MDX concept and it defines MDX elements for use in the Analyzer. These elements determine how you can query the data, specifically, a set of specific records (such as patient records or transaction records). The set of records is determined by the *source class* for the cube.

A cube can contain all the following definitions:

- Levels, which enable you to group records
- Hierarchies, which contain levels
- Dimensions, which contain hierarchies
- Level properties, which are values specific to the members of a level
- Measures, which show aggregate values of those records

- Listings, which are queries that enable you to access the source data

- Calculated members, which are members based on other members

- Named sets, which are reusable sets of members or other MDX elements

The following sections discuss most of these items. For information on named sets, see *Using MDX with DeepSee*.

2.1.1 The Source Class of a Cube

In most cases, the source class for a cube is a persistent class.

The source class can also be a *data connector*, which is a class that extends %DeepSee.DataConnector. A data connector maps the results of an arbitrary SQL query into an object that can be used as the source of a cube. Typically, a data connector accesses external non-Caché data, but you can also use it to specify an SQL query against Caché, including an SQL query on a view. See "Defining and Using Data Connectors" in the *DeepSee Implementation Guide*.

The source class can also be a child collection class.

2.2 Dimensions, Hierarchies, and Levels

This section discusses dimensions, hierarchies, and levels.

2.2.1 Levels and Members

A level consists of members, and a *member* is a set of records. For the City level, the Juniper member selects the patients whose home city is Juniper. Conversely, each record in the cube belongs to one or more members.

Most pivot tables simply display data for the members of one or more levels. For example, the Age Group level has the members 0 to 29, 30 to 59, and 60+. The following pivot table shows data for these members:

Age Group	Patient Count
0 to 29	4,255
30 to 59	4,119
60+	1,626

For another example, the following pivot table shows data for the Age Group and Gender levels (shown as the first and second columns respectively).

Age Group		Patient Count
0 to 29	Female	2,073
	Male	2,182
30 to 59	Female	2,030
	Male	2,089
60+	Female	939
	Male	687

You can also drag and drop individual members for use as rows or columns. For example:

Gender	Patient Count
Female	5,115
Male	4,885
1910s	78
Elm Heights	1,138

For more details on the options, see *Using the DeepSee Analyzer*.

2.2.2 Member Names and Keys

Each member has both a name and an internal key. The compiler does not require either of these to be unique, even within the same level. Duplicate member names are legitimate in some cases, but you should check for and avoid duplicate member keys.

For an example of legitimate duplicate member names, consider that different people can have the same name, and you would not want to combine them into a single member. When a user drags and drops a member, DeepSee uses the member key, rather than the name, in its generated query, so that the query always accesses the desired member.

Duplicate member keys, however, make it impossible to refer to each individual member. Given a member key, DeepSee returns only the first member that has that key. You can and should ensure that your model does not result in duplicate member keys.

For information on the scenarios in which you might have duplicate member names and duplicate member keys, see "Defining Member Keys and Names Appropriately," in the next chapter.

2.2.3 Source Values

Each level is based on a *source value*, which is either a class property or an ObjectScript expression. For example, the Gender level is based on the Gender property of the patient. For another example, the Age Group level is based on an expression that converts the patient's Age property to a string (0 to 29, 30 to 59, or 60+), depending on the age.

2.2.4 Hierarchies and Dimensions

In DeepSee, levels belong to hierarchies, which belong to dimensions. Hierarchies and dimensions provide additional features beyond those provided by levels.

Hierarchies are a natural and convenient way to organize data, particularly in space and time. For example, you can group cities into postal codes, and postal codes into countries.

There are three practical reasons to define hierarchies in DeepSee:

- DeepSee has optimizations that make use of them. For example, if you are displaying periods (year plus month) as rows or columns, and you then filter to a specific year, the query runs more quickly if your model defines years as the parent of periods.

- You can use hierarchies within a pivot table as follows: If you double-click a member of a level, DeepSee performs a drilldown to show the child members of that member, if any. For example, if you double-click a year, DeepSee drills down to the periods within that year; for details, see *Using the DeepSee Analyzer*.

- MDX provides functions that enable you to work with hierarchies. For example, you can query for the child postal codes of a given country, or query for the other postal codes in the same country.

  You can use these functions in handwritten queries; the Analyzer does not provide a way to create such queries via drag and drop.

A *dimension* contains one or more parent-child hierarchies that organize the records in a similar manner; for example, a single dimension might contain multiple hierarchies related to allergies. There is no formal relationship between two different hierarchies or between the levels of one hierarchy and the levels of another hierarchy. The practical purpose of a dimension is to define the default behavior of the levels that it contains — specifically the All level, which is discussed in the next subsection.

2.2.5 The All Level and the All Member

Each dimension can define a special, optional level, which appears in all the hierarchies of that dimension: the *All level*. If defined, this level contains one member, the *All member*, which corresponds to all records in the cube.

For example, the `AgeD` dimension includes one hierarchy with levels as follows:
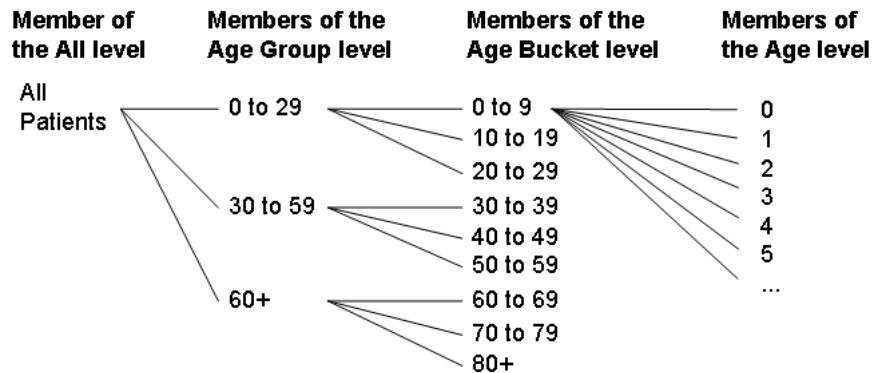
For a given dimension, you specify whether the All member exists, as well as its logical name and its display name. Within this dimension, the All member is named `All Patients`.

2.2.6 List-Based Levels

In DeepSee, unlike many other BI tools, you can base a level upon a list value. For example, a patient can have multiple diagnoses. The `Diagnoses` level groups patients by diagnosis. For example:

Diagnoses	Patient Count
None	8,480
asthma	660
CHD	319
diabetes	508
osteoporosis	217
Total	10,184

For a list-based level, any given source record can belong to multiple members. The pivot table shown here includes some patients multiple times.

2.2.7 See Also

See the chapter "Defining Dimensions, Hierarchies, and Levels."

2.3 Properties

Each level may define any number of *properties*. If a level has a property, then each member of that level has a value for that property; other levels do not have values for the property.

In the Patients sample, the `City` level includes the properties `Population` and `Principal Export`.

Each property is based on a source value, which is either a class property or an ObjectScript expression. For the `City` level, the properties `Population` and `Principal Export` are based directly on class properties.

You can use properties in queries in much the same way that you use measures. For example, in the Analyzer, you can use properties as columns (this example shows two properties):

City	Population	Principal Export
Cedar Falls	90,000	iron
Centerville	49,000	video games
Cypress	3,000	gravel
Elm Heights	33,194	lettuce
Juniper	10,333	wheat
Magnolia	4,503	bundt cake
Pine	15,060	spaghetti
Redwood	29,192	peaches
Spruce	5,900	mud

In contrast to measures, properties cannot be aggregated. A property is null for all levels except for the level to which it belongs.

See the chapter "Defining Properties."

2.4 Measures

A cube also defines measures, which show aggregate values in the data cells of a pivot table.

Each measure is based on a source value, which is either a class property or an ObjectScript expression. For example, the `Avg Test Score` measure is based on the patient's `TestScore` property.

The definition of a measure also includes an aggregation function, which specifies how to aggregate values for this measure. Functions include SUM and AVG.

For example, the following pivot table shows the `Patient Count` measure and the `Avg Test Score` measure. The `Patient Count` measure counts the patients used in any context, and the `Avg Test Score` measure shows the average test score for the patients used in any context. This pivot table shows the value for these measures for the members of the `Age Group` level:

Age Group	Patient Count	Avg Test Score
0 to 29	4,255	59.79
30 to 59	4,119	59.83
60+	1,626	60.09

See the chapter "Defining Measures."

2.5 Listings

A cube can also contain listings. Each listing has a name and specifies the fields to display when the user requests that listing. The following shows an example:

#	PatientID	Age	Gender	Home City	Test Score
1	SUBJ_100524	30	F	Elm Heights	73
2	SUBJ_101001	30	F	Juniper	91
3	SUBJ_101014	30	F	Elm Heights	87
4	SUBJ_101072	30	F	Redwood	70
5	SUBJ_101331	30	F	Cedar Falls	

The records shown depend on the context from which the user requested the listing.

If you do not define any listings in a cube, then if a user requests a listing, the Analyzer displays the following message:

```
Error #5001: %ExecuteListing: this cube does not support drill through
```

See the chapter "Defining Listings."

Also:

- A cube can define *individual* listing fields with which users can create custom listings in the Analyzer. See the chapter "Defining Listing Fields."

- You can define listings outside of cube definitions (and without needing access to the Architect). See the chapter "Defining Listing Groups."

2.6 Calculated Members

A *calculated member* is based on other members. You can define two kinds of calculated members:

- A *calculated measure* is a measure is based on other measures. (In MDX, each measure is a member of the Measures dimension.)

  For example, one measure might be defined as a second measure divided by a third measure.

  The phrase *calculated measure* is not standard in MDX, but this documentation uses it for brevity.

- A non-measure calculated member typically aggregates together other non-measure members. Like other non-measure members, this calculated member is a group of records in the fact table.

Calculated members are evaluated after the members on which they are based.

You can create calculated members of both kinds within the cube definition, and users can create additional calculated members of both kinds within the Analyzer.

2.6.1 Calculated Measures

It is very useful to define new measures based on other measures. For example, in the Patients sample, the `Avg Test Score` measure is defined as the `Test Score` measure divided by the `Count` measure. Consider the following pivot table:

Allergy Severities	Count	Allergy Count	Avg Allergy Count
Minor	1,167	1,909	1.64
Moderate	1,114	1,847	1.66
Life-threatening	1,201	1,939	1.61
Inactive	1,143	1,805	1.58
Unable to determine	1,101	1,780	1.62
No Data Available	4,000		
Nil known allergies	1,482	0	0

When this pivot table is run, DeepSee determines the values for the `Count` and `Test Score` measures for each member of the `Allergy Severities` level; a later section of this chapter describes how DeepSee does this. Then, for the `Avg Test Score` value for each member, DeepSee divides the `Test Score` value by the `Count` value.

2.6.2 Non-Measure Calculated Members

For a non-measure calculated member, you use an MDX aggregation function to combine other non-measure members. The most useful function is %OR.

Remember that each non-measure member refers to a set of records. When you combine multiple members into a new member, you create a member that refers to all the records that its component members use.

For a simple example, consider the `ColorD` dimension, which includes the members `Red`, `Yellow`, and `Blue`. These members access the patients whose favorite color is red, yellow, or blue, respectively. You can use %OR to create a single new member that accesses all three groups of patients.

For example:

Primary Colors	Patient Count
Primary Colors	3,765

2.6.3 See Also

See the chapter "Defining Calculated Members."

2.7 Subject Areas

A subject area is a subcube with optional overrides to names of items. You define a subject area to enable users to focus on smaller sets of data without having to build multiple cubes. In a subject area, you can do the following:

- Specify a filter that restricts the data available in the subject area. For information on filters, see the next section.

You can hardcode this filter, or you specify it programmatically, which means that you can specify it based on the **$roles** of the user, for example.

- Hide elements defined in the cube so that the Analyzer displays a subset of them.

- Define new names, captions, and descriptions for the visible elements.

- Specify the default listing for the subject area.

- Redefine or hide listings defined in the cube.

- Define new listings.

You can then use the subject area in all the same places where you can use a cube. For example, you can use it in the Analyzer, and you can execute MDX queries on it in the shell or via the API.

See the chapter "Defining Subject Areas."

2.8 Filters

In BI applications, it is critical to be able to filter data in pivot tables and in other locations. This section discusses the filter mechanisms in DeepSee and how you can use them in your application.

2.8.1 Filter Mechanisms

DeepSee provides two simple ways to filter data: member-based filters and measure-based filters. You can combine these, and more complex filters are also possible, especially if you write MDX queries directly.

2.8.1.1 Member-Based Filters

A member is a set of records. In the simplest member-based filter, you use a member to filter the pivot table (for example, other contexts are possible, as this section describes later). This means that the pivot table accesses only the records that belong to that member.

For example, consider the following pivot table, as seen in the Analyzer:

Age Group	Patient Count
0 to 29	4,364
30 to 59	4,036
60+	1,600
Total	10,000

Suppose that we apply a filter that uses the `0 to 29` member of the `Age Group` level. The resulting pivot table looks like this:

Age Group	0 to 29

Age Group	Patient Count
0 to 29	4,364
Total	4,364

The Analyzer provides options to display null rows and columns. If we display null rows, the pivot table looks like this:

Age Group	0 to 29 ▾	
Age Group	**Patient Count**	
0 to 29	4,364	
30 to 59		
60+		
Total	4,364	

We can use the same filter in any pivot table. For example, consider the following unfiltered pivot table:

Favorite Color	Female	Male	Total
None	1,187	1,220	2,407
Blue	669	566	1,235
Green	675	642	1,317
Orange	658	598	1,256
Purple	639	616	1,255
Red	665	606	1,271
Yellow	622	637	1,259
Total	5,115	4,885	10,000

This pivot table shows the `Patient Count` measure although the headings do not indicate this. If we filter this pivot table in the same way as the previous one, we see this:

Age Group	0 to 29 ▾		
Favorite Color	**Female**	**Male**	**Total**
None	494	579	1,073
Blue	263	257	520
Green	261	294	555
Orange	260	256	516
Purple	270	273	543
Red	310	280	590
Yellow	268	299	567
Total	2,126	2,238	4,364

Notice that the total record count is the same in both cases; in both cases, we are accessing only patients that belong to the `0 to 29` member.

You can also use multiple members together in a filter, and you can combine filters that refer to members of different levels. Also, rather than choosing the members to include, you can choose the members to exclude.

Tip: Member-based filters are so easy to create and so powerful that it is worthwhile to create levels whose sole purpose is for use in filters.

2.8.1.2 Measure-Based Filters

DeepSee supports *searchable measures*. With such a measure, you can apply a filter that considers the values at the level of the source record itself.

For the Patients sample, you can have a filter that accesses only the patients who have an encounter count of 10 or higher. If we use this filter in a pivot table, we might see this:

Age Group	Patient Count	Total
0 to 29	3,246	3,246
30 to 59	3,305	3,305
60+	1,330	1,330
Total	7,881	7,881

If we use the same filter in a different pivot table, we might see this:

Favorite Color	Female	Male	Total
None	926	973	1,899
Blue	537	451	988
Green	530	523	1,053
Orange	515	454	969
Purple	495	486	981
Red	533	485	1,018
Yellow	469	504	973
Total	4,005	3,876	7,881

In both cases, the total patient count is the same, because in both cases, the pivot table uses only the patients who have at least 10 encounters.

A searchable measure can also contain text values. With such measures, you can use the operators = <> and LIKE to filter the records.

2.8.1.3 More Complex Filters

It is possible to create more complex filters that combine member- and measure-based filters. The following shows an example of such a filter, as created in the Analyzer:
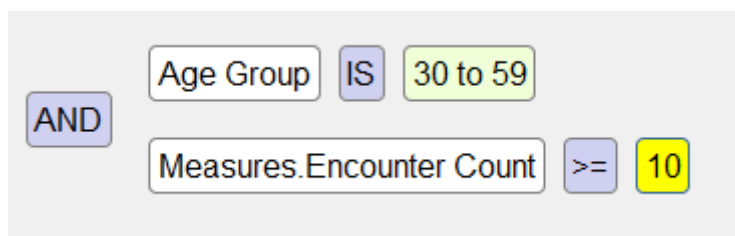
Internally, the query does not use AND and OR, but instead uses MDX syntax. All DeepSee filters use MDX syntax.

You can also create filters that use MDX functions. For example:

- The FILTER function uses the aggregate values of a measure, rather than the lowest-level values which a measure-based filter uses. For example, you can use this to filter out patients who belong to cities that have fewer than 1000 patients.

    In the Analyzer, the Levels option for a row or column uses this function internally.

- The EXCEPT function can be used to remove specific members. DeepSee uses this function when you create a member-based filter that excludes your selected members.

  DeepSee MDX provides many other functions that perform set operations.

- The TOPCOUNT and other functions access members based on their ranking.

For an introduction to MDX and a survey of your options, see *Using MDX with DeepSee*. Also see *DeepSee MDX Reference*.

2.8.2 Using Filters

When you define a pivot table, you can specify how it is filtered. In practice, however, it is undesirable to create multiple similar pivot tables with different filters, because the pivot tables can become difficult to maintain. Instead, you can use any or all of the following tools:

- In the Analyzer, you can define named filters, which you can then use in multiple pivot tables. A named filter is available in the Analyzer along with the contents of the cube or subject area (see the next item).

- In the Architect, you can define subject areas that are filtered views of a base cube. Then when you create pivot tables, you start with a subject area rather than with the cube itself. These pivot tables are always filtered by the subject area filter, in addition to any filters that are specific to the pivot tables themselves.

  In a subject area, you can specify a hardcoded filter, or you can customize a callback method to specify the filter at runtime (to base it on a value such as **$roles**, for example).

- In the User Portal, when you create dashboards, you can include filter controls in them (this applies only to simple, member-based filters). Then the user can select the member or members to include or exclude.

Filters are always cumulative.

2.9 How DeepSee Builds and Uses Fact Tables

When you compile a cube, DeepSee generates a fact table and related tables. When you build a cube, DeepSee populates these tables and generates their indices. At runtime, DeepSee uses these tables. This section describes this process. It includes the following topics:

- Structure of a fact table

- How DeepSee populates a fact table

- How DeepSee uses a fact table

DeepSee does not generate tables for subject areas. A subject area uses the tables that have been generated for the cube on which the subject area is based.

2.9.1 Structure of a Fact Table

A fact table typically has one record for each record of the base table; this row is a fact. The fact contains one field for each level and one field for each measure. The following shows a sketch:

Age	Allergies	Home City	Home ZIP	...	Test Score	meas B	...
42	dairy products,eggs	Juniper	32006	nnnnn	78	nnnnn	nnnnn
7		Cypress	34577	nnnnn	53	nnnnn	nnnnn
52		Elm Heights	38928	nnnnn	89	nnnnn	nnnnn
13	ant bites,bee stings,soy	Pine	34577	nnnnn	61	nnnnn	nnnnn
8		Juniper	32006	nnnnn	90	nnnnn	nnnnn
12	dairy	Elm Heights	38928	nnnnn	67	nnnnn	nnnnn
34	ant bites,dairy products	Centerville	36711	nnnnn	74	nnnnn	nnnnn
26	ant bites,bee stings,eggs	Magnolia	34577	nnnnn	89	nnnnn	nnnnn
89	ant bites	Spruce	32006	nnnnn	84	nnnnn	nnnnn
22	soy	Spruce	32006	nnnnn	59	nnnnn	nnnnn
...							

The field for a given level might contain no values, a single value, or multiple values. Each distinct value corresponds to a member of this level. The field for any given measure contains either null or a single value.

When DeepSee builds this fact table, it also generates indices for it.

The fact table does not contain information about hierarchies and dimensions. In the fact table, regardless of relationships among levels, each level is treated in the same way: the fact table contains one column for each level, and that column contains the value or values that apply to each source record.

Tip: By default, the fact table has the same number of rows as the base table. When you edit a cube class in Studio, you can override its **OnProcessFact()** callback, which enables you to ignore selected rows of the base table. If you do so, the fact table has fewer rows than the base table.

2.9.2 Populating the Fact Table

When you build a cube, the system iterates through the records of the base table. For each record, the system does the following:

- Examines the definition of each level and obtains either no value, a single value, or multiple values.

  In this step, the system determines how to categorize the record.

- Examines the definition of each measure and obtains either no value or a single value.

The system then writes this data to the corresponding row in the fact table and updates the indices appropriately.

2.9.2.1 Determining the Values for a Level

Each level is specified as either a source property or a source expression. Most source expressions return a single value for given record, but if the level is of type list, its value is a Caché list of multiple values.

For a given record in the base table, the system evaluates that property or expression at build time, stores the corresponding value or values in the fact table, and updates the indices appropriately.

For example, the `Age Bucket` level is defined as an expression that returns one of the following strings: `0-9`, `10-19`, `20-29`, and so on. The value returned depends upon the patient's age. The system writes the returned value to the fact table, within the field that corresponds to the `Age Bucket` level.

For another example, the `Allergy` level is a list of multiple allergies of the patient.

2.9.2.2 Determining the Value for a Measure

When DeepSee builds the fact table, it also determines and stores values for measures. Each measure is specified as either a source property or an ObjectScript source expression.

For a given row in the base table, the system looks at the measure definition, evaluates it, and stores that value (if any) in the appropriate measure field.

For example, the `Test Score` measure is based on the `TestScore` property of the patients.

2.9.2.3 Determining the Value for a Property

When DeepSee builds the fact table, it also determines values for properties, but it does not store these values in the fact table. In addition to the fact table, the system generates a table for each level (with some exceptions; see the appendix "Details for the Fact and Dimension Tables"). When the system builds the fact table, it stores values for properties in the appropriate dimension tables.

2.9.3 Using a Fact Table

Consider the following pivot table:

Age Bucket	Count	Test Score	Avg Test Score
0 to 9	1,410	82,860	58.77
10 to 19	1,471	88,229	59.98
20 to 29	1,374	83,297	60.62
30 to 39	1,529	90,769	59.36
40 to 49	1,487	89,755	60.36
50 to 59	1,103	65,909	59.75
60 to 69	745	44,056	59.14
70 to 79	569	34,698	60.98
80+	312	18,960	60.77

The first column displays the names of the members of the `Age Bucket` level. The first data column shows the `Patient Count` measure, the second data column shows the `Test Score` measure, and the last column shows the `Avg Test Score` measure. The `Avg Test Score` measure is a calculated member.

The system determines these values as follows:

1. The first row refers to the `0-9` member of the `Age Bucket` level. The system uses the indices to find all the relevant patients (shown here with red highlighting) in the fact table:

Age Bucket	Allergies	level C	...	Test Score	meas B	meas C	...
40 to 49	eggs	nnnnn	nnnnn	84	nnnnn	nnnnn	nnnnn
0 to 9		nnnnn	nnnnn	53	nnnnn	nnnnn	nnnnn
50 to 59		nnnnn	nnnnn	89	nnnnn	nnnnn	nnnnn
10 to 19	ant bites,bee stings,soy	nnnnn	nnnnn	61	nnnnn	nnnnn	nnnnn
0 to 9		nnnnn	nnnnn	90	nnnnn	nnnnn	nnnnn
10 to 19	dairy	nnnnn	nnnnn	67	nnnnn	nnnnn	nnnnn
30 to 39	ant bites,dairy products	nnnnn	nnnnn	74	nnnnn	nnnnn	nnnnn
20 to 29	ant bites,bee stings,eggs	nnnnn	nnnnn	89	nnnnn	nnnnn	nnnnn
80 +	ant bites	nnnnn	nnnnn	84	nnnnn	nnnnn	nnnnn
20 to 29	soy	nnnnn	nnnnn	59	nnnnn	nnnnn	nnnnn
...							

2. In the pivot table, the `Patient Count` column shows the count of patients used in a given context.

   For the first cell in this column, DeepSee counts the number of records in the fact table that it has found for the `0-9` member.

3. In the pivot table, the `Test Score` column shows the cumulative test score for the patients in a given context.

For the first cell in this column, DeepSee first finds the values for the `Test Score` in the fact table that it has found for the `0-9` member:

Age Bucket	Allergies	level C	...	Test Score	meas B	meas C	...
40 to 49	eggs	nnnnn	nnnnn	84	nnnnn	nnnnn	nnnnn
0 to 9		nnnnn	nnnnn	53	nnnnn	nnnnn	nnnnn
50 to 59		nnnnn	nnnnn	89	nnnnn	nnnnn	nnnnn
10 to 19	ant bites,bee stings,soy	nnnnn	nnnnn	61	nnnnn	nnnnn	nnnnn
0 to 9		nnnnn	nnnnn	90	nnnnn	nnnnn	nnnnn
10 to 19	dairy	nnnnn	nnnnn	67	nnnnn	nnnnn	nnnnn
30 to 39	ant bites,dairy products	nnnnn	nnnnn	74	nnnnn	nnnnn	nnnnn
20 to 29	ant bites,bee stings,eggs	nnnnn	nnnnn	89	nnnnn	nnnnn	nnnnn
80 +	ant bites	nnnnn	nnnnn	84	nnnnn	nnnnn	nnnnn
20 to 29	soy	nnnnn	nnnnn	59	nnnnn	nnnnn	nnnnn
...							

Then it aggregates those numbers together, in this case by adding them.

4. In the pivot table, the `Avg Test Score` column is meant to display the average test score for the patients in a given context.

   The `Avg Test Score` measure is a calculated member, computed by dividing `Test Score` with `Patient Count`.

The system repeats these steps for all cells in the result set.

2.10 How DeepSee Generates Listings

This section describes how the system uses the listings defined in the cube.

Within a pivot table, a user selects one or more cells.

	Female		Male	
Age Group	Patient Count	Avg Test Score	Patient Count	Avg Test Score
0 to 29	2,064	60.06	2,115	60.62
30 to 59	2,071	60.52	2,090	60.33
60+	961	61.50	699	61.03

The user then clicks the Listing button ⚏, and the system displays a listing, which shows the values for the lowest-level records associated with the selected cells (also considering all filters that affect this cell):

#	PatientID	Age	Gender	Home City	Test Score
1	SUBJ_100524	30	F	Elm Heights	73
2	SUBJ_101001	30	F	Juniper	91
3	SUBJ_101014	30	F	Elm Heights	87
4	SUBJ_101072	30	F	Redwood	70
5	SUBJ_101331	30	F	Cedar Falls	

To generate this display, the system:

1. Creates an temporary listing table that contains the set of source ID values that correspond to the facts used in the selected cells.

2. Generates an SQL query that uses this listing table along with the definition of your listing.

3. Executes this SQL query and displays the results.

Your cube can contain multiple listings (to show different fields for different purposes). When you create a pivot table in the Analyzer, you can specify which listing to use for that pivot table.

The listing query is executed at runtime and uses the source data rather than the fact table. Because of this, if the fact table is not completely current, it is possible for the listing to show a different set of records than you see in the fact table.

3

Summary of Model Options

A model can contain many elements in addition to cubes and subject areas. The additional elements are discussed in the *Advanced DeepSee Modeling Guide*. For reference and planning, this chapter summarizes *all* the elements, from both this book and the *Advanced DeepSee Modeling Guide*. This chapter discusses the following topics:

- Items you can use directly in pivot tables
- Items you can use in calculated members and measures
- Comparison of possible widget data sources
- High-level summary

3.1 Items That You Can Use in Pivot Tables

This section compares items that you can use *directly* in the Analyzer to define a pivot table. These items are categorized as follows:

- Levels and other items that group or filter records
- Items that behave measures
- Properties
- Items that are not directly accessible

3.1.1 Items That Group or Filter Records

The following items can be used to group or filter records. Except for some searchable measures, none of these items can be used as measures.

Item	Use in Rows and Columns options	Use in Filters option
dimension or level	Yes	Yes
dimension or level of a related cube	Yes	Yes
shared dimension or level in the same compound cube	Yes	Yes
calculated member (non-measure)	Yes	Yes

Item	Use in Rows and Columns options	Use in Filters option
calculated member (non-measure) in a related cube	*Yes*	*Yes*
calculated member (non-measure) in another cube in the same compound cube	No	No
computed dimension	*Yes*	*Yes*
computed dimension in a related cube	*Yes*	*Yes*
computed dimension in another cube in the same compound cube	No	No
named set	*Yes*	*Yes*
named set in another cube	No	No
named filter	No	*Yes*
named filter in another cube	No	No
searchable measures	*Yes*, except for text, string, and iKnow measures	*Yes*

Related cubes, compound cubes, computed dimensions, and iKnow measures are discussed in the *Advanced DeepSee Modeling Guide*.

Named filters are discussed in *Using the DeepSee Analyzer*.

3.1.2 Items That Behave Like Measures

The following items behave like measures and are shown in the body of a pivot table. With one exception (noted), these items cannot be used for filtering.

Item	Use in Rows and Columns options	Use in Measures option	Use in Filters option
measure	*Yes*, except for text, string, and iKnow measures	*Yes*, except for text, string, and iKnow measures	*Yes, if searchable*
measure of a related cube	No[*]	No[*]	No
measure of another cube in the same compound cube	*Yes*	*Yes*	No
calculated measure	*Yes*	*Yes*	No
calculated measure from another cube	No	No[**]	No
quality measure	*Yes*	*Yes*	No
quality measure from another cube	Not applicable[**]	Not applicable[**]	Not applicable[**]
property of pivot-type plugin	*Yes*	*Yes*	No
pivot-type plugin from another cube	Not applicable[**]	Not applicable[**]	Not applicable[**]

Related cubes, compound cubes, iknow measures, quality measures, and plugins are discussed in the *Advanced DeepSee Modeling Guide*.

[*]These measures would not be aggregated correctly if used this way.

[**]Quality measures and plugins are designed to be *directly* associated with any cubes where they are to be used.

3.1.3 Properties

The following table summarizes how properties can be used:

Item	Use in Rows and Columns options	Use in Measures option	Use in Filters option
property	*Yes*	No	No
property in another cube (related or compound)	*Yes*	No	No

Related cubes and compound cubes are discussed in the *Advanced DeepSee Modeling Guide*.

3.1.4 Items That You Cannot Access Directly in the Analyzer

For reference, note that you cannot directly access the following items in the Analyzer:

• KPIs

• Term lists

• Aggregate-type plugins

• Ensemble business metrics

Except for business metrics, however, you can define calculated members that use these items; see the next section. Then you can use those calculated members in the Analyzer.

These items are all discussed in the *Advanced DeepSee Modeling Guide*.

3.2 Items That You Can Use in Calculated Members and Measures

Via calculated members and measures, you can greatly extend your model without rebuilding any cubes. The following table summarizes all the kinds of model element that you can use within the definition of a calculated member or measure:

Item	To Access This Item from a Calculated Member or Measure
MDX standard cube items (dimension, hierarchy, level, measure, property, calculated member, named set)	Many options; see *Using MDX with DeepSee*.
computed dimension[*]	Use the dimension and member name to create a member expression that refers to the desired member, in the same way that you do for any standard member.

Item	To Access This Item from a Calculated Member or Measure
quality measure[*]	Use the `%QualityMeasure` dimension to create a quality measure expression that refers to the measure.
KPI or plugin[*]	Use the %KPI function to refer to the value of a property in the KPI or plugin.
term list[*]	• Use the %LOOKUP function to return the value of a term list item. • Use the LOOKUP function to return a given field (the value field or another field), for a term list item. • Use the %TERMLIST function to create a set of members, given a term list pattern.

[*]These items are discussed in the *Advanced DeepSee Modeling Guide*.

DeepSee does not provide a way to access a Ensemble business metric from within a calculated member.

3.3 Comparison of Possible Widget Data Sources

The preceding chapter introduced pivot tables, which are the most common kind of data source for a widget on a dashboard. DeepSee provides many other kinds of data sources. You can *directly* use any of the following items as data sources:

• Pivot tables

• KPIs (see the *Advanced DeepSee Modeling Guide*)

• Pivot-type plugins (see the *Advanced DeepSee Modeling Guide*)

• Ensemble business metrics (see *Developing Ensemble Productions*)

The following table compares these items:

Feature	Pivot Tables	KPIs	Plugins	Ensemble Business Metrics
How definition is stored	Folder item	Class definition	Class definition	Class definition
Where defined	Analyzer	Studio	Studio	Studio
Data source options	MDX query	• MDX query • SQL query • Values returned by custom code	MDX query	Values returned by custom code
Supports custom computation using lowest-level records?	No	No	Yes	No
Supports filtering?	Yes	Yes, depending on your implementation	Yes, depending on your implementation	No
Supports custom filtering (for example, provide user with only a subset of the members of a level)?	No	Yes, depending on your implementation	Yes, depending on your implementation	No
Supports listings?	Yes	Yes, depending on your implementation	Yes	No
Supports the Mini Analyzer?	Yes	No	No	No

3.4 High-Level Summary of Options

For reference, the following table summaries the possible contents of a DeepSee model, including information on which tool you use to create each element:

Item	Purpose	Location of definition	Primary tool to create	Where described
dimension, hierarchy, and level	Define groups of records	cube class	Architect	"Defining Dimensions, Hierarchies, and Levels"

Item	Purpose	Location of definition	Primary tool to create	Where described
measure (numeric, integer, date, age, or boolean)	Aggregate values across multiple records	cube class	Architect	"Defining Measures"
measure (text or string)	Store string data in the fact table. These measures are usually also searchable.			
searchable measure	Aggregate values across multiple records; *also* support filtering of lowest-level records			
iKnow measure	Used by iKnow dimensions; not visible in the Analyzer	cube class	Architect	"Using Unstructured Data in Cubes (iKnow)" in the *Advanced DeepSee Modeling Guide*
property	Contain data specific to a level	cube class	Architect	"Defining Properties"
listing	Provide access to lowest-level details	cube class	Architect	"Defining Listings," "Defining List Fields", and "Defining Listing Groups"
calculated measure	Define a measure based on other model elements	cube class or pivot table	Architect or Analyzer	"Defining Calculated Members"
calculated member (non-measure)	Define a member based on other model elements			
named set	Define a reusable set of members	cube class	Architect	"Defining Named Sets"
subject area	Filter a cube or otherwise refine its definition	subject area class	Architect	"Defining Subject Areas"
computed dimension	Define groups of records to retrieve at runtime (usually via SQL or MDX)	cube class	Architect	"Defining Computed Dimensions" in the *Advanced DeepSee Modeling Guide*
quality measure	Define measure via MDX outside of a cube	quality measure class	Quality Measure Manager	"Defining Quality Measures" in the *Advanced DeepSee Modeling Guide*
related cubes	Use levels of a different cube	cube class	Architect	"Defining Cube-Cube Relationships" in the *Advanced DeepSee Modeling Guide*

Item	Purpose	Location of definition	Primary tool to create	Where described
compound cube	Combine measures from different cubes or see these measures side by side	subject area class	Architect	"Defining Shared Dimensions and Compound Cubes" in the *Advanced DeepSee Modeling Guide*
KPI	Query the data in a more flexible way, for use as an alternative data source	KPI class	Studio	"Defining Basic KPIs" in the *Advanced DeepSee Modeling Guide*
pivot-type plugin	Calculate a value based on the lowest-level data; use via drag and drop	plugin class	Studio	"Defining Plugins" in the *Advanced DeepSee Modeling Guide*
aggregate-type plugin	Calculate a value based on the lowest-level data; use in calculated members			
term list	Define values outside of a cube, for various purposes	DeepSee folder item	Term List manager	Defining Term Lists in the *Advanced DeepSee Modeling Guide*
business metric	Compute data specific to an Ensemble production	business metric class	Studio	*Developing Ensemble Productions*
named filter	Define a filter for use with a specific cube	global	Analyzer	*Using the DeepSee Analyzer*

4

Principles and Recommendations

This chapter discusses core principles and other recommendations for your DeepSee models:

- Choosing a base table

- Choosing the form of the base table

- Avoiding large numbers of levels and measures

- Defining measures appropriately

- Understanding time levels

- Defining hierarchies appropriately

- Defining member keys and names appropriately

- Avoiding very granular levels

- Using list-based levels carefully

- Handling null values appropriately

- Usability considerations

- Considerations when you define multiple cubes

- Other recommendations

4.1 Choosing a Base Table

When defining a cube, the first step is to choose the class to use as the base class for that cube. The key point to remember is this: Within the cube, any record count refers to a count of records in this class (as opposed to some other class referenced within the cube). Similarly, the selection of the base class determines the meaning of all measures in the cube.

For example:

- If the base class is `Transactions`, transactions are counted. In this cube, you can have measures like the following:

  - Average broker fee per transaction (or average for any group of transactions)

  - Average transaction value per transaction (or average for any group of transactions)

- If the base class is `Customers`, customers are counted. In this cube, you can have measures like the following:

  - Average broker fee per customer (or average for any group of customer)

  –   Average transaction value per customer (or average for any group of customers)

You can have multiple cubes, each using a different base class, and you can use them together in dashboards.

4.2 Choosing the Form of the Base Table

It is also important to consider the *form* of the base table. The following table summarizes the key considerations:

Base Table	Support for DSTIME?	Support for Listings?
Local persistent class (other than a linked table)	Yes	Yes
Linked table	No	Yes
Data connector that uses local tables	No	Yes
Data connector that uses linked tables	No	No

The DSTIME feature is the easiest way to keep the corresponding cube current. When this feature is not available, other techniques are possible. For details, see "Keeping the Cubes Current" in the *DeepSee Implementation Guide*.

4.3 Avoiding Large Numbers of Levels and Measures

Caché imposes a maximum limit on the number of levels and measures on a cube, because there is a limit on the number of indices in a class. For information on this limit (which may increase over time), see "General System Limits" in the *Caché Programming Orientation Guide*. For information on the number of indices that DeepSee creates for levels and measures, see the appendix "Details for the Fact and Dimension Tables."

It is also best to keep the number of levels and measures much smaller than required by this limit, because an overly complex model can be hard to understand and use.

4.4 Defining Measures Appropriately

The value on which a measure is based must have a one-to-one relationship with the records in the base table. Otherwise, DeepSee would not aggregate that measure suitably. This section demonstrates this principle.

4.4.1 Measures from Parent Tables

Do not base a measure on a field in a parent table. For example, consider the following two tables:

*   `Order` — Each row represents an order submitted by a customer. The field `SaleTotal` represents the total monetary value of the order.

*   `OrderItem` — Each row represents an item in that order. In this table, the field `OrderItemSubtotal` represents the monetary value of this part of the order.

Suppose that we use `OrderItem` as the base table. Also suppose that we define the measure `Sale Total`, based on the parent's `SaleTotal` field. The goal for this measure is to display the total sale amount for all the sales of the selected order items.

Let us consider the contents of the fact table. The following shows an example:

dim A	dim B	...	Sale Total	meas B	meas C	...
nnnnn	nnnnn	nnnnn	279.07	nnnnn	nnnnn	nnnnn
nnnnn	nnnnn	nnnnn	279.07	nnnnn	nnnnn	nnnnn
nnnnn	nnnnn	nnnnn	279.07	nnnnn	nnnnn	nnnnn
nnnnn	nnnnn	nnnnn	279.07	nnnnn	nnnnn	nnnnn
nnnnn	nnnnn	nnnnn	52.14	nnnnn	nnnnn	nnnnn
nnnnn	nnnnn	nnnnn	52.14	nnnnn	nnnnn	nnnnn

The first four rows represent the items in the same order. The next two rows represent the items of another order, and so on.

Suppose that this model has a dimension called `Item Type`. Let us examine what happens when DeepSee retrieves records for all items of type R:

Item Type	dim B	...	Sale Total	meas B	meas C	...
type Q	nnnnn	nnnnn	279.07	nnnnn	nnnnn	nnnnn
type R	nnnnn	nnnnn	279.07	nnnnn	nnnnn	nnnnn
type R	nnnnn	nnnnn	279.07	nnnnn	nnnnn	nnnnn
type S	nnnnn	nnnnn	279.07	nnnnn	nnnnn	nnnnn
type R	nnnnn	nnnnn	52.14	nnnnn	nnnnn	nnnnn
type T	nnnnn	nnnnn	52.14	nnnnn	nnnnn	nnnnn

To compute the value of the `Sale Total` measure for type R, DeepSee adds together the three values shown here: 279.07, 279.07, and 52.14. But this action double-counts one of the orders.

Depending on the case, the `Sale Total` measure might aggregate correctly; that is, it might show the correct total sales figure for the selected order items. But you cannot ensure that the measure does this, because you cannot prevent double-counting as shown as in this example.

4.4.2 Measures from Child Tables

You can use a value in a child table as the basis of a measure, but to do so, you must aggregate that value across the relevant rows of the child table.

Consider the following two tables:

- `Customer` — Each row represents a customer.

- `Order` — Each row represents a customer order. The field `SaleTotal` represents the total monetary value of the order.

Suppose that we use `Customer` as the base table, and that we want to create a measure based on the `SaleTotal` field.

Because a customer potentially has multiple orders, there are multiple values for `SaleTotal` for a given customer. To use this field as a measure, we must aggregate those values together. The most likely options are to add the values or to average them, depending on the purpose of this measure.

4.5 Understanding Time Levels

A time level groups records by time; that is, any given member consists of the records associated with a specific date and time. For example, a level called Transaction Date would group transactions by the date on which they occurred. There are two general kinds of time levels, and it is important to understand their differences:

- *Timeline-based levels.* This kind of time level divides the timeline into adjacent blocks of time. Any given member of this level consists of a single block of time. Or, more accurately, the member consists of the records associated with that block of time. For a level called `Transaction Quarter Year`, the member `Q1-2011` would group all the transactions that occurred in any of the dates that belong to the first quarter of 2011.

  This kind of level can have any number of members, depending on the source data.

- *Date-part based levels.* This kind of time level considers only *part* of the date value and ignores the timeline. Any given member consists of multiple blocks of time from different parts of the timeline, as shown in the following figure. Or, more accurately, the member consists of the records associated with those blocks of time. For a level called `Transaction Quarter`, the member `Q1` would group all the transactions that occurred in any of the dates that belong to the first quarter of any year.

  This kind of level has a fixed number of members.

The following figure compares these kinds of time levels:

Transaction Quarter Year level
Timeline-based level

| ... | Q1-2011 | Q2-2011 | Q3-2011 | Q4-2011 | Q1-2012 | ... |

time

| Q1 | Q2 | Q3 | Q4 |

Transaction Quarter level
Date-part-based level
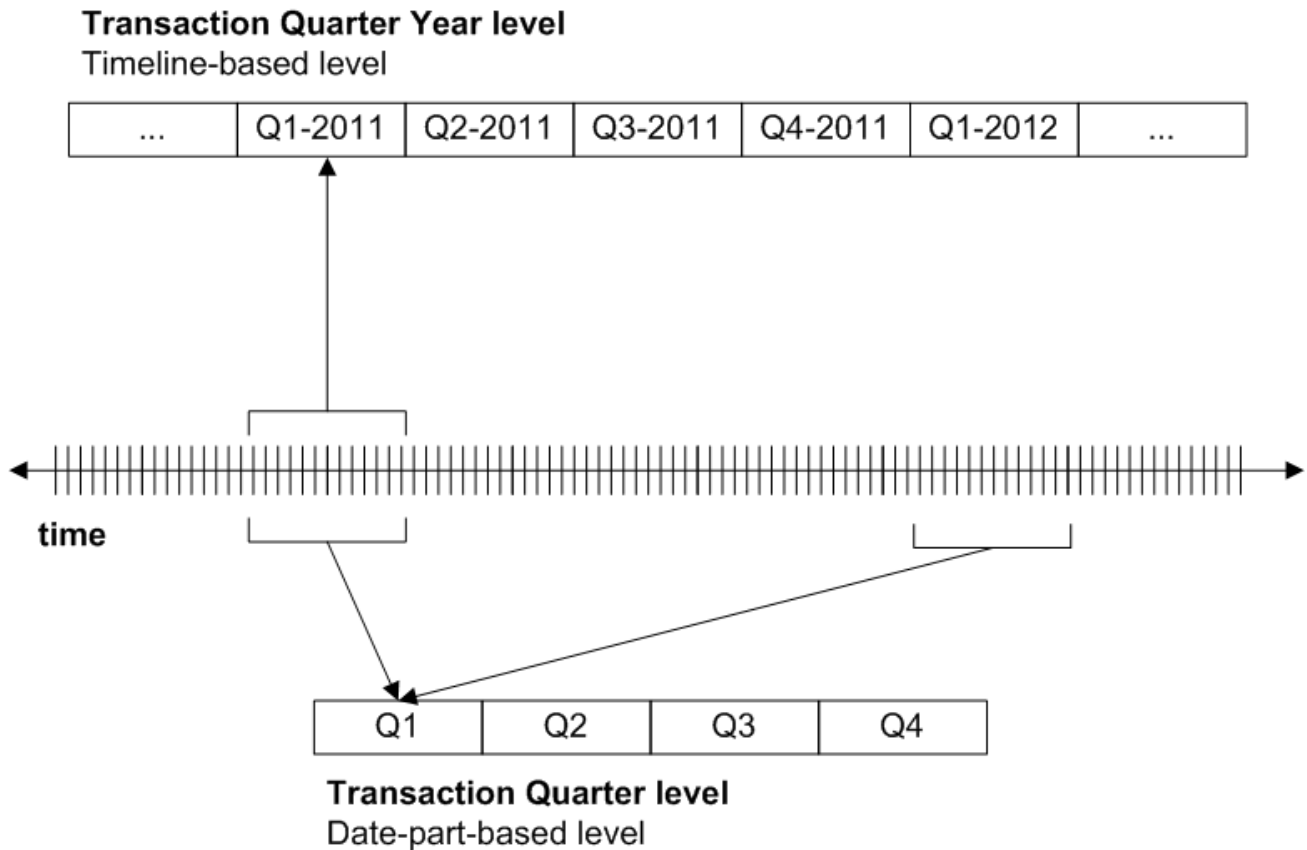
You can use these kinds of levels together without concern; the engine will always return the correct set of records for any combination of members. However, there are two typical sources of confusion:

- If you define hierarchies of time levels (that is, hierarchies that contain more than one level), you must think carefully about what you include. As noted in the next section, MDX hierarchies are parent-child hierarchies. Any member in the parent level must contain all the records of its child members.

  This means, for example, that you can use the `Transaction Year` level as the parent of the `Transaction Quarter Year` level, but not of the `Transaction Quarter` level. To see this, refer to the preceding figure. The `Q1` member refers to the first quarters of all years, so no single year can be the parent of `Q1`.

  "Time Levels and Hierarchies," later in this book, gives guidelines on appropriate time hierarchies.

- Some MDX functions are useful for levels that represent the timeline, but are not useful for levels that represent only a date part. These functions include PREVMEMBER, NEXTMEMBER, and so on. For example, if you use PRE-VMEMBER with `Q1-2011`, the engine returns `Q4-2010` (if your data contains the applicable dates). If you use PREVMEMBER with `Q1`, there is nothing useful to return.

You might want to try defining the following flexible approach: Define a Year level so that you have one level that represents the timeline. Define all other time levels as date-part levels. Then you can use these levels together as in the following example:

Year-	Quarter	Patient Count
2000	Q1	2
	Q2	2
	Q3	1
	Q4	3
2001	Q1	5
	Q2	4
	Q3	2
	Q4	5
2002	Q1	1
	Q2	6
	Q3	5

This pivot table uses a combination of Year (a timeline level) and Quarter (a date-part level) for rows. The Quarter level is also useful on its own, because it can provide information on quarter-by-quarter patterns. The sample data in SAMPLES does not demonstrate this, but there are many kinds of seasonal activity that you could see this way. The following shows an example:
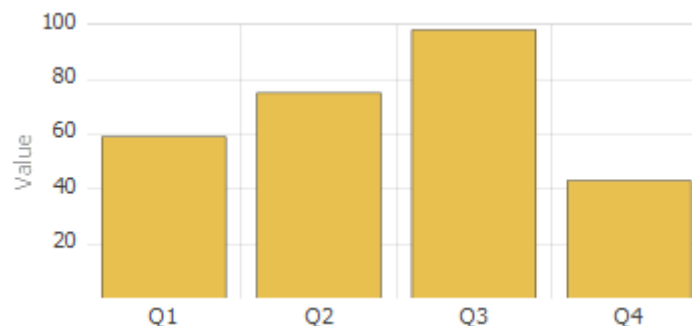
4.6 Defining Hierarchies Appropriately

In any hierarchy, pay attention to the order of the levels as shown in the Architect. A level is automatically a child of the previously listed level in that hierarchy. For example, consider the `HomeD` dimension from the Patients sample:
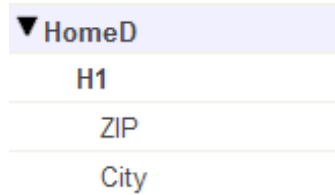
The `ZIP` level is the parent of the `City` level. More precisely, each member of the `ZIP` level is the parent of one or more members of the `City` level. (In reality, there is a many-to-many relationship between ZIP codes and cities, but the Patients sample is simplistic, and in this sample, ZIP codes represent larger areas than cities.)

In your hierarchies, the first level should be the least granular, and the last one should be the most granular.

MDX hierarchies are parent-child hierarchies. To enforce this rule, DeepSee uses the following logic when it builds the cube:

- For the first level defined in a hierarchy, DeepSee creates a separate member for each unique source value.

- For the levels after that, DeepSee considers the source value for the level in combination with the parent member.

  For example, suppose that the first level is `State`, and the second level is `City`. When it creates members of the `City` level, DeepSee considers both the city name and the state to which that city belongs.

The internal logic is slightly different for time hierarchies, but the intent is the same.

4.6.1 What Happens if a Hierarchy Is Inverted

In contrast, suppose that we move the `City` level so that it is before the `ZIP` level in the cube, recompile, and rebuild. If we use the `ZIP` level for rows in a pivot table, we see something like the following:

ZIP	Patient Count
32006	1,126
32006	1,104
32007	1,095
34577	1,136
34577	1,159
34577	1,108
36711	1,076
38928	1,113
38928	1,083

In this case, DeepSee has created more than one member with the same name, because it assumes that (for example) there are two ZIP 32006 codes, which belong to different cities.

In this case, it is obviously incorrect to have multiple members with the same name. In other scenarios, however, it is legitimate to have multiple members with the same name. For example, different countries can have cities with the same name. That is, if you have duplicate member names, that *can* indicate an inverted hierarchy, but not in all cases.

4.6.2 Time Hierarchies

MDX hierarchies are parent-child hierarchies. All the records for a given child member must be contained in the parent member as well. It is important to carefully consider how to place time levels into hierarchies. See the previous section. Also "Time Levels and Hierarchies," later in this book, gives guidelines on appropriate time hierarchies.

4.7 Defining Member Keys and Names Appropriately

Each member has both a name and an internal key. By default, these are the same, except for time levels. Both of these identifiers are strings. When you define your model, you should consider the following items:

- It can be correct to have duplicate member names. Different people, for example, can have the same name. When a user drags and drops a member, DeepSee uses the member key, rather than the name, in its generated query.

- InterSystems recommends that you ensure that member keys are unique in each level. Duplicate member keys make it difficult to refer to all the individual members.

  In particular, if you have duplicate member keys, users cannot reliably drill down by double-clicking.

- To help users distinguish the members, if a level has multiple members with the same name, InterSystems also recommends that you add a property to the level whose value is the same as the key. To do so, simply base the property on the same source property or source expression that the level uses.

  Then the users can display the property to help distinguish the members.

  Or add a tooltip to the level, and have that tooltip contain the member key.

The following sections describe the scenarios in which you might have duplicate member keys and names.

4.7.1 Ways to Generate Duplicate Member Keys

Duplicate member keys are undesirable. If you have duplicate member keys, users cannot reliably drill down by double-clicking.

Except for time levels, each unique source value for a level becomes a member key. (For time levels, the system uses slightly different logic to generate unique keys for all members.)

If there is a higher level in the same hierarchy, it is possible for DeepSee to generate multiple members with the same key. See "Ensuring Uniqueness of Member Keys," later in this book.

4.7.2 Ways to Generate Duplicate Member Names

Duplicate member names may or may not be undesirable, depending on your business needs.

Except for time levels, each unique source value for a level becomes a member name, by default. (For time levels, the system generates unique names for all members.)

There are only two ways in which DeepSee can generate multiple members with the same name:

- There is a higher level in the same hierarchy. This higher level might or might not be suitable. See "Defining Hierarchies Appropriately," earlier in this chapter.

- You have defined a level as follows:

  – The member names are defined separately from the level definition itself.

– The member names are based on something that is not unique.

See "Using Property Values as Member Names," later in this book.

Note that Caché does not automatically trim leading spaces from string values as it builds the fact table. If the source data contains leading spaces, you should use a source expression that removes those. For example:

```
$ZSTRIP(%source.myproperty,"<W")
```

Otherwise, you will create multiple members with names that *appear* to be the same (because some of the names have extra spaces at the start).

4.8 Avoiding Very Granular Levels

For users new to DeepSee, it is common to define a level that has a one-to-one relationship with the base table. Such a level is valid but is not particularly useful, unless it is also combined with filters to restrict the number of members that are seen.

A very granular level has a huge number of members (possibly hundreds of thousands or millions), and DeepSee is not designed for this scenario.

For example, suppose that we modified the Patient sample to have a Patient level. Then we could have a pivot table like this:

PatientID	Patient Count	Age	Allergy Count
SUBJ_100301	1	2	
SUBJ_100302	1	79	
SUBJ_100303	1	17	2
SUBJ_100304	1	72	
SUBJ_100305	1	66	1
SUBJ_100306	1	44	
SUBJ_100307	1	66	1

This is valid but an SQL query would produce the same results more efficiently. Consider the processing that is described in "How DeepSee Builds and Uses Fact Tables," in the previous chapter. (That description gives the conceptual flow rather than the actual processing, but the overall idea is the same.) That processing is intended to aggregate values together as quickly as possible. The pivot table shown above has no aggregation.

If you need a pivot table like the one shown here, create it as an SQL-based KPI. See the *Advanced DeepSee Modeling Guide*.

4.9 Using List-Based Levels Carefully

In DeepSee, unlike many other BI tools, you can base a level upon a list value. Such levels are useful, but it is important to understand their behavior.

For example, a patient can have multiple allergies. Each allergy has an allergen and a severity. Suppose that the base table is `Patients` and the model includes the `Allergies` and `Allergy Severity` levels. We could create a pivot table that looks like this:

Allergies	Nil known allergies	Minor	Moderate	
additive/coloring agent		137	121	
animal dander		132	136	
ant bites		126	131	
bee stings		124	141	
dairy products		128	114	
dust mites		126	103	
eggs		132	125	
fish		137	135	
mold		116	136	
nil known allergies	1,578			
peanuts		129	114	
pollen		131	135	

Upon first seeing this pivot table, the user might think that this pivot table shows correlations between different sets of patient allergies. It does not.

This pivot table, as with all other pivot tables in this cube, shows sets of patients. For example, the `ant bites` row represents patients who have an allergy to ant bites. The `Minor` column represents patients who have at least one allergy that is marked as minor. There are 126 patients who have an allergy to ant bites and who have at least one allergy that is marked as minor. This does *not* mean that there are 126 patients with minor allergies to ant bites.

It is possible to create a pivot table that does show correlations between different sets of patient allergies. To do so, however, you would have to define a model based on the patient allergy, rather than the patient.

When you use a list-based level in a filter, the results require careful thought. Consider the following pivot table:

Allergies	Patient Count
No Data Available	3,943
additive/coloring agent	413
animal dander	418
ant bites	436
bee stings	446
dairy products	445
dust mites	416
eggs	389
fish	419
mold	397
nil known allergies	1,427
peanuts	444
pollen	426
shellfish	442
soy	449
tree nuts	453
wheat	435

This pivot table shows patients, grouped by allergy. Now suppose that we apply a filter to this pivot table, and the filter selects the fish member:

Allergies fish	
Allergies	**Patient Count**
additive/coloring agent	13
animal dander	19
ant bites	16
bee stings	23
dairy products	24
dust mites	22
eggs	12
fish	419
mold	13
peanuts	22
pollen	22
shellfish	23
soy	20
tree nuts	24
wheat	19

Now we are viewing only patients who have an allergy to fish. Note the following:

- The pivot table shows the fish member. For this member, the patient count is the same as in the previous pivot table.

- It also shows some other members of the Allergies level. For these members, the patient count is lower than in the previous pivot table.

- The pivot table does not include the No Data Available member of the Diagnoses level. Nor does it include the Nil Known Allergies member.

To understand these results, remember that we are viewing only patients who are allergic to fish, and this is not the same as "viewing only the fish allergy." The patients who are allergic to fish are also allergic to other things. For example, 13 of the patients who are allergic to fish are also allergic to additives/coloring agents.

If we change the filter to select only the No Data Available member, we see this:

Allergies No Data Available	
Allergies	**Patient Count**
No Data Available	3,943

In this case, we are viewing only the patients who do not have any recorded allergy. By definition (because of how this level is defined), there is no overlap of these patients with the patients who have specific allergies.

Note: List-based levels cannot have child or parent levels.

4.10 Handling Null Values Correctly

For any measure or level, the source value could potentially be null in some cases. It is important to understand how DeepSee handles nulls and to adjust your model according to your business needs and usability requirements.

4.10.1 Null Values in a Measure

For a measure, if the source value is missing for a given record, DeepSee does not write any value into the measure column of the fact table. Also, DeepSee ignores that record when aggregating measure values. In most scenarios, this is appropriate behavior. If it is not, you should use a source expression that detects null values and replaces them with a suitable value such as 0.

4.10.2 Null Values in a Level

For a level, if the source value is missing for a given record in the base class, DeepSee automatically creates a member to contain the null values (with one exception). You specify the null replacement string to use as the member name; otherwise, the member is named `Null`.

The exception is computed dimensions, which are discussed in the *Advanced DeepSee Modeling Guide*. The replacement string has no effect in this case.

4.11 Usability Considerations

It is also useful to consider how users see and use the model elements. This section explains how the Analyzer and pivot tables represent model elements and concludes with some suggestions for your models.

4.11.1 Consider How Dimensions, Hierarchies, and Levels Are Presented

The Model Contents area of the Analyzer displays each dimension and the levels in it, but does not display the hierarchies (for reasons of space). For example:
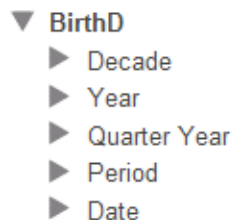
Therefore, a user working in the Analyzer does not necessarily know which levels are related via hierarchies.

If a level is used for rows, then the name of the level appears as the column title. For example:

If a dimension is used for rows, the name of the dimension appears as the column title. Also, the system uses an MDX function that gets the All member for the dimension, as well as all members of the first level defined in that dimension:

AgeD	Patient Count
All Patients	10,000
0 to 29	4,255
30 to 59	4,119
60+	1,626

4.11.2 Consider How to Use All Members

The Model Contents area displays every All member. For example:

▼ **AgeD**
　☀ All Patients
　▶ Age Group
　▶ Age Bucket
　▶ Age
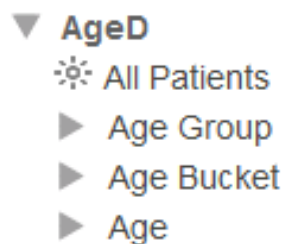
In the Analyzer, users can drag and drop the All member, in the same way they can drag any other member.

The All member for one dimension is equivalent (except for its name) to the All member for any other dimension. If it has a suitably generic name, an All member is useful as the bottom line. For example:

Diagnoses	Patient Count	Avg Age	Avg Enc Count
None	8,425	33.24	31.26
asthma	703	34.79	31.42
CHD	323	67.49	52.85
diabetes	504	57.24	47.38
osteoporosis	200	79.46	60.56
All Patients	10,000	36.01	33.03

4.12 Considerations with Multiple Cubes

If you find that you need to define multiple cubes to analyze one area of your business, consider the following points:

- If you have a large number of cubes, relationships are often quite useful. Rather than defining the same dimension repeatedly in different cubes (with different definitions), you can define it in a single place. This is more convenient for development and lessens the amount of disk space that is needed.

  The slight disadvantage of relationships is that if you rebuild the independent cube, you must also rebuild all the dependent cubes in the appropriate order, as discussed in the *Advanced DeepSee Modeling Guide*.

- If you define relationships or shared dimensions, use the Cube Manager to build the cubes in the correct order; see "Keeping the Cubes Current" in the *DeepSee Implementation Guide*.

Or define a utility method or routine that builds the cubes in the appropriate order. It is easier to maintain such a method as you add cubes than it is to manually rebuild them in the correct order.

Building related cubes in the wrong order can cause problems that are difficult to troubleshoot.

- If you need pivot tables that display measures from multiple cubes, you must define shared dimensions and a compound cube. A compound cube is the only way to use measures together that belong to different cubes.

4.13 Recommendations

The following recommendations may also be useful to you, depending on your business needs:

- Decide whether you will use dimensions directly in pivot tables. If so, assign user-friendly names to them (at least for display names).

  If not, keep their names short and omit spaces, to enable you to write MDX queries and expressions more easily.

  Also, use a different name for the dimension than for any level in the dimension, in order to keep the syntax clear, if you need to view or create MDX queries or expressions.

- Hierarchy names are not visible in the Analyzer or in pivot tables. If you use short names (such as `H1`), your MDX queries and expressions are shorter and easier to read.

- Define only one hierarchy in any dimension; this convention gives the users an easy way to know if the levels in a dimension are associated with each other.

- Define an All member in only one dimension. Give this All member a suitably generic name, such as `All Patients`. Depending on how you intend to use the All member, another suitable name for it might be `Total` or `Aggregate Value`.

- Use user-friendly names for levels, which are visible in pivot tables.

- You can have multiple levels with the same name in different hierarchies. Pivot tables and filters, however, show only the level name, so it is best to use unique level names.

- Specify a value for the **Field name in fact table** option for each applicable level and measure; this option does not apply to time levels, iKnow levels, or iKnow measures. Take care to use unique names.

  It is much easier to troubleshoot when you can identify the field in which a given level or measure is stored, within the fact table.

  For information, see "Details for the Fact and Dimension Tables."

5

Defining Cubes

This chapter describes the basics of defining cubes. It includes the following topics:

- How to define a cube
- Possible source classes for a cube
- How to specify the other cube options
- How to add items to a cube
- Rules for names of items in a cube
- Other common options for items in a cube
- How to compile and build a cube
- How to display a cube in the Analyzer
- How to delete a cube

Important: DeepSee uses SQL to access data while building the cube, and also when executing detail listings. If your model refers to any class properties that are SQL reserved words, you must enable support for delimited identifiers so that DeepSee can escape the property names. For a list of reserved words, see "Reserved Words" in the *Caché SQL Reference*. For information on enabling support for delimited identifiers, see "Identifiers" in *Using Caché SQL*.

5.1 Defining a Cube

To define a cube:

1. In the Architect, click **New**.

   The system displays a dialog box where you can enter details for the new cube.

2. Click **Cube**.

3. Enter the following information at a minimum:

   - **Cube Name** — Logical name of the cube to use in queries.
   - **Class Name for the Cube** — Complete package and class name for the cube class.

- **Source Class** — Complete package and class name of the base class for this cube. See the subsection "Possible Source Classes."

  You can either type the class name or click **Browse** and select the class.

  The other options are discussed later in this chapter.

  Apart from the class name of the cube class, you can edit all cube options after creating the cube.

4. Click **OK**.

5. Optionally save the cube. To do so:

   a. Click **Save**.

   b. Click **OK**.

   The system creates the class.

Another option is to use a utility to generate the cube class, as discussed in the following subsection.

Or manually create the class as described in the appendix "Reference Information for Cube Classes."

5.1.1 Generating the Cube Class

The class %DeepSee.WizardUtils provides a method that you can use to generate a cube class. The method is as follows:

```
classmethod %GenerateCubeDefinition(pSourceClass As %Library.String(MAXLEN="")="",
                                    pCubeName As %Library.String(MAXLEN=""),
                                    pCubeClass As %Library.String(MAXLEN="")="",
                                    pAutoDelete As %Library.Integer = 0)
```

Where:

- *pSourceClass* is the full name of the source class for the cube.

- *pCubeName* is the logical name of the cube.

- *pCubeClass* is the full name of the cube class.

- *pAutoDelete* controls whether the cube class is deleted, if it already exists. If this argument is nonzero, the class is deleted; otherwise it is not.

This method generates a cube definition as follows:

- It has one measure for each numeric property in the source class.

- It has one date dimension for each date property in the source class. This dimension contains one hierarchy with three levels. The levels are year, year and month, and date.

- It has one data dimension for each other property in the source class. This dimension contains one hierarchy with one level.

- It has one listing that uses all properties in the source class.

The method ignores transient and multidimensional properties.

5.1.2 Changing the Base Class for a Cube

On rare occasions, you might need to change the base class for a cube. To do so, you can do either of the following in the Architect:

- Edit the **Source Class** option for the cube in the Details Area.

- Click the **Change** link next to **Source Values** at the top of the **Class Viewer**.

  If you do so, the system displays a dialog box where you can choose a source class; this is the same dialog box as is shown in "Possible Source Classes."

After you do this, be sure to modify the source property or source expression appropriately for all parts of the model.

5.2 Possible Source Classes for a Cube

In the Architect, if you click **Browse** next to **Source Class**, the system displays a dialog box like the following:
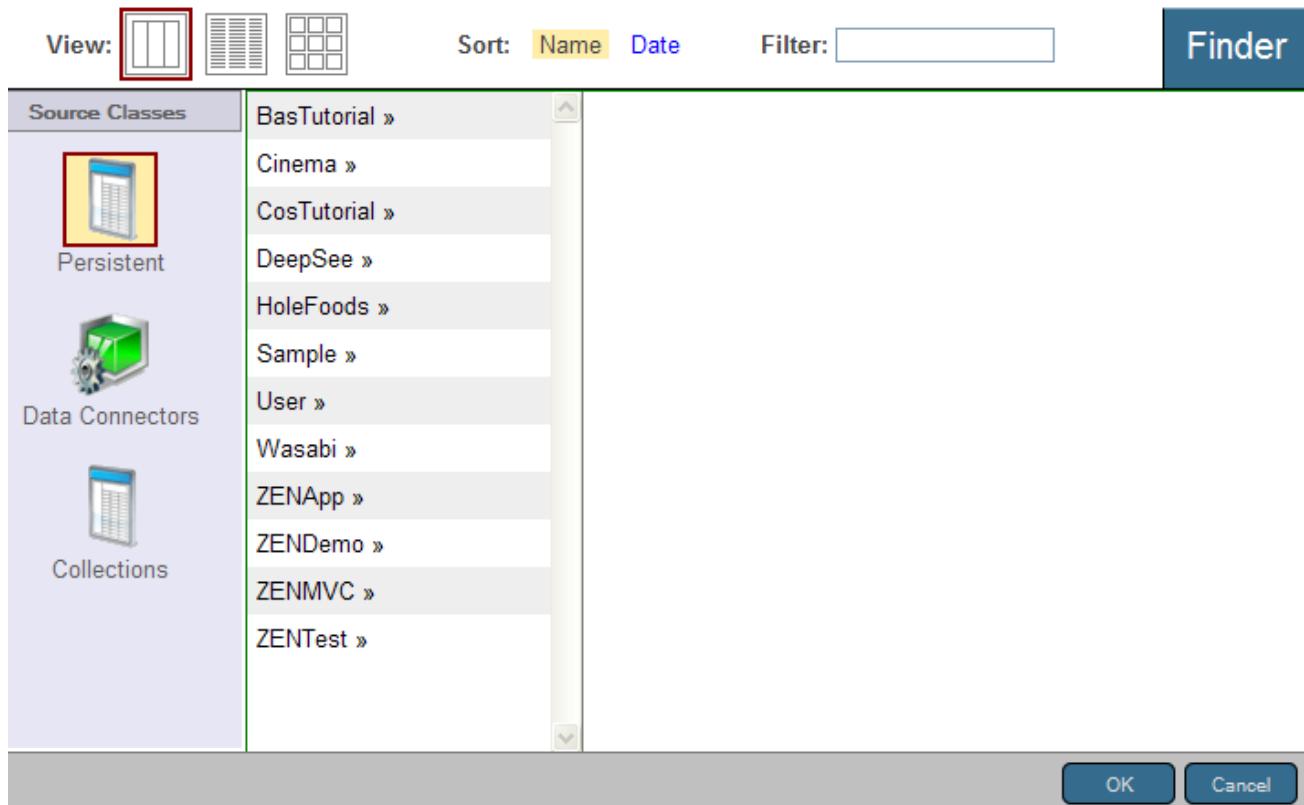
Here you can select any class that can be used as the source of a cube in this namespace. There are three types of classes you can use this way:

- Persistent classes — Classes that extend %Library.Persistent.

- Data connector classes — Classes that extend %DeepSee.DataConnector. A data connector maps the results of an arbitrary SQL query into an object that can be used as the source of a cube. Typically, a data connector accesses external non-Caché data, but you can also use it to specify an SQL query against Caché, including an SQL query on a view.

  For a cube that is based on a data connector, all listings in that cube must also be based on data connectors. Also, all these data connectors must have the same property marked as `idkey="true"`, because the underlying mechanism uses the same ID values in all cases.

  See "Defining and Using Data Connectors" in the *DeepSee Implementation Guide*.

- Collection classes.

5.3 Other Cube Options

In the Architect, you can specify the following options for a cube:

- **Cube name** — Logical name of the cube to use in queries.

- **Display name** — Localizable name of the cube. If you do not specify this, the user interface instead displays the logical name.

- **Description** — (Optional) Comments to add to the cube class definition. Each line is saved as a separate comment line at the start of the class definition.

- **Caption**— (Optional) Specify the caption to display in the Analyzer and other utilities when working with this cube.

- **Domain**— (Optional) Specify the name of the domain to contain the localized strings of this cube. You might find it convenient to use a single domain for all your cubes; in other cases, it might be appropriate to have a separate domain for each cube. See the chapter "Performing Localization" in the *DeepSee Implementation Guide*.

- **Source class** — Complete package and class name of the base class for this cube.

- **Null replacement string** — (Optional) Specifies the string (for example, `None`) to use as the member name if the source data for a level is null.

  This option can overridden for levels, by a level option of the same name.

- **Default listing** — (Optional) Logical name of the default listing for this cube. This listing must be defined in the cube.

- **Resource** — (Optional) Specify the Caché resource that secures the cube.

  For information on how this is used, see "Setting Up Security" in the *DeepSee Implementation Guide*.

- **Owner** — (Optional) Specify the owner of the cube. Specify a Caché username.

- **Count measure name** — (Optional) Specify an alternative name for the Count measure. The default is `%COUNT`. It is useful to rename the Count measure if you create a compound cube; see the *Advanced DeepSee Modeling Guide*.

- **Count measure caption** — (Optional) Specify an alternative caption for the Count measure. The default is `COUNT`.

- **Initial build order** — (Optional) Specifies an optional ORDER BY clause for use when building the entire cube; does not affect cube synchronization or incremental updates. Specify a comma-separated list of fields in the source table. You can use the SQL keywords ASC and DESC. For example: `Age DESC,Gender`

  For the implications of this option, see the section "Controlling the Fact Order" in the *DeepSee Implementation Guide*.

- **Build restriction** — (Optional) Specifies an optional WHERE clause to use when building or updating the cube; this causes the cube to use a subset of the records. Specify an SQL comparison expression that uses fields in the source table. For example: `Gender='F'`

  This option has no effect if the cube is based on a data connector.

  For an alternative option, see "Restricting the Records Used in the Cube" in the chapter "Using Advanced Features of Cubes and Subject Areas" of the *Advanced DeepSee Modeling Guide*.

- **Depends On** — (Optional) Specifies the class or classes that must be runnable before this class can be compiled. This option controls how the Architect sets the DependsOn compiler keyword.

  By default, this option is blank, and DeepSee automatically sets the DependsOn keyword equal to the name of the source class for the cube. In some cases (for example with cube relationships), you might need to specify an additional class.

  If you need to specify this option, specify a comma-separated list of classes and specify the full package and class name for each class in the list. Your list should include the source class for the cube.

For information on relationships between cubes, see the *Advanced DeepSee Modeling Guide*.

5.4 Adding Items to a Cube

In the Architect, there are two general ways to add items to a cube:

- By using the **Add Element** link, as follows:

  1. Click **Add Element** at the top of the Model Viewer.

     The system displays a dialog box where you can choose the type of item to add.

  2. Enter the item name.

  3. Click the item type.

  4. Click **OK**.

- By using a drag-and-drop action, as follows:

  1. Drag a property name from the Class Viewer.

  2. Drop the name onto a heading in the Model Viewer; the results depend upon where you drop the name. For example, if you drag and drop a property name onto the **Measures** heading, the Architect creates a measure.

In both cases, the Architect adds the item and displays it in the Model Viewer. You can then make edits in the Details Area.

The following table indicates the parts of this book that discuss the different types of cube items. These sections also provide specific information on where you can drag and drop property names.

For information on ...	See the chapter ...
Data Dimension	"Defining Dimensions, Hierarchies, and Levels"
Time Dimension	
Age Dimension	
Hierarchy	
Level	
Property	"Defining Properties"
Measure	"Defining Measures"
Listing	"Defining Listings"
Listing Field	"Defining Listing Fields"
Calculated Member	"Defining Calculated Members"
Named Set	"Defining Named Sets"

5.5 Names for Model Elements

When you define a model element, you specify a logical name for it (the **Name** field in the Architect). This name is used in MDX queries and also is the default display name for that element. This section discusses requirements and suggestions for these names.

The logical names must follow these rules:

- The first character must be either a letter (in the Latin-1 character set), a number, or the underscore character (_).

- The other characters must be either letters, numbers, spaces, or underscore characters.

  Note that if you use spaces in a name, you must enclose the name within square brackets when writing MDX queries.

- The name must not be an MDX reserved keyword. Reserved keywords are not case-sensitive in MDX.

The logical names must also follow these additional rules:

- Within a given Caché namespace, each cube name must be unique.

- Within a given cube, each dimension name must be unique.

- Within a given dimension, each hierarchy name must be unique.

- Within a given hierarchy, each level name must be unique.

  Note: Level names do not have to be unique within a cube. If you do have multiple levels with the same name within a cube, however, you must specify the **Field name in fact table** option and ensure that each level has a unique name in the fact table. See "Specifying Field Names in the Fact Table," later in this book.

- Within a given level, each property name must be unique.

- Names of intrinsic properties are case-insensitive reserved keywords and cannot be used as names of your properties, with one exception. The exception is that you can create a property named Name (in any case), if that property also has the option **Use as member names** enabled.

  For a list of intrinsic properties, see "Intrinsic Properties" in the *DeepSee MDX Reference*.

- Within a given cube, each measure name must be unique.

5.6 Other Common Options

In the Architect, when you define a model element, you can also specify the following options for it:

- **Display name** — (Optional) Localized name of this element for use in user interfaces. If you do not specify this, the user interface instead displays the logical name.

- **Description** — (Optional) Description of this element.

- **Disabled** — (Optional) If you select this check box, the element is disabled (not seen by the compiler). When you recompile the cube, this element is ignored.

- **Additional Description** — (Optional) Additional notes about this element, for display only within the Architect and Studio.

5.7 Compiling and Building a Cube

In the Architect, as you develop your cubes, you will probably recompile and rebuild them multiple times. Briefly:

- To compile a cube, click **Compile**.

  The system starts to compile the class and displays a dialog box that shows progress.

  If you have made changes that you have not yet saved, the system saves them.

  Then click **Done**.

- To build a cube, click **Build**. The system displays a dialog box. Click **Build**.

  DeepSee starts to build the cube and displays progress as it does so. Then click **OK**.

  The cube is then available for use in the Analyzer.

For more information, see the chapter "Compiling and Building Cubes," later in this book.

5.8 Opening a Cube in the Analyzer

As you develop your cubes, you should periodically use the Analyzer and examine the results. To open a cube in the Analyzer:

1. Click **DeepSee**, click **Analyzer**, and then click **Go**.

   Tip: If the Analyzer is already open, just click the **Analyzer** link at the top of the page.

2. If the left area is not currently displaying the cube you are validating, click **Open** and choose the cube.

For specific tips on validating levels, see "Validating Your Levels," later in this book. For general information on using the Analyzer, see *Using the DeepSee Analyzer*.

5.9 Deleting a Cube

To delete a cube, do the following:

1. In the Terminal, execute the following command:

   ```
   do ##class(%DeepSee.Utils).%KillCube(cubeName)
   ```

   Where *cubeName* is the logical name of the cube to remove. This command removes the cube cache and indices.

2. Also in the Terminal, delete the cube metadata, as follows:

   ```
   kill ^DeepSee.Cubes("cubes",cubeName)
   ```

   Where *cubeName* is the logical name of the cube to remove.

3. Delete the cube class (and its generated classes and their data) in either of the following ways:

   - In the Terminal, execute the following command:

     ```
     do $system.OBJ.Delete(classname)
     ```

Where *classname* is the full package and class name of the cube class. For example:

```
do $system.OBJ.Delete("Mypackage.Myclass")
```

- In Studio, right-click the cube class and select **Delete**.

Or, if you decide not to delete the cube, recompile and rebuild it.

6

Compiling and Building Cubes

This chapter describes how to compile and build cubes. It includes the following topics:

- When to recompile and rebuild

- How to compile a cube

- How to build a cube in the Architect

- How to build a cube programmatically

- How to minimize cube size during development

- How to use parallel processing during a cube build

- Build errors

- DeepSee task log

Note: During the build process, users cannot execute queries. (However, if a query is currently running, you can build the cube.)

See the *DeepSee Implementation Guide* for the following related topics:

- Cube synchronization, which updates cubes incrementally (and permits queries to be executed at the same time). See the chapter "Keeping the Cubes Current" in the above listed book.

- The DeepSee Cube Manager utility, which you typically use for production systems. The Cube Manager creates automated tasks that build or synchronize cubes according to your specifications. See the chapter "Keeping the Cubes Current" in the above listed book.

- The cube version feature, which enables you to modify a cube definition, build it, and provide it to users, with only a short disruption of running queries. See the appendix "Using Cube Versions" in the above listed book.

6.1 When to Recompile and Rebuild

If you make any change to a cube class or a subject area class, you must recompile that class before those changes take effect. For many changes to a cube, you must also rebuild the cube before those changes take effect.

The following table lists the required actions after changes:

Element Type	Type of Change	Required Actions
Cube (root element)	Edits to **Name** or **Source class**	Recompile and rebuild
	Other changes that apply to the cube but not to specific elements in the cube	Recompile
Measure	Edits to the following options of an existing measure (many other elements have some or all of these common options). • **Disabled** • **Hidden** • **Display name** • **Description** • **Format string**	Recompile
	All other changes, including adding and deleting measures	Recompile and rebuild
Dimension (not a computed dimension)	Edits to the following options of an existing dimension: • Common options as listed in "Measure" • **Enable the All level for this dimension** • **Caption for All member** • **Display name for All member**	Recompile
	All other changes, including adding and deleting dimensions	Recompile and rebuild
Computed dimension	All changes	Recompile
iKnow dimension	All changes	Recompile
Hierarchy	Edits to the common options of an existing hierarchy (as listed in "Measure")	Recompile
	All other changes, including adding and deleting hierarchies	Recompile and rebuild
Level	Edits to the following options of an existing level: • Common options as listed in "Measure" • **Null replacement string** • **Sort option**	Recompile
	All other changes, including adding and deleting levels[*]	Recompile and rebuild
Property	Edits to the following options of an existing property: • Common options as listed in "Measure" • **Sort members by property value**	Recompile
	All other changes, including adding and deleting properties	Recompile and rebuild
Listing	All changes	Recompile

Element Type	Type of Change	Required Actions
Calculated member	All changes	Recompile
Named set	All changes	Recompile
Subject area	All changes	Recompile
Compound cube (a kind of subject area)	All changes	Recompile (*after* recompiling all cubes used in the compound cube)
Quality measure	All changes	Recompile the quality measure class
KPI or plugin	All changes	Recompile the KPI or plugin class

*The current server locale determines the names of members of a time dimension. (See "Using the Locale to Control the Names of Time Members," later in this book.) If you change the locale, it is necessary to recompile and rebuild the cube.

6.2 Compiling a Cube

To compile a cube class in the Architect:

1. Click **Compile**.

   The system starts to compile the class and displays a dialog box that shows progress.

   If you have made changes that you have not yet saved, the system saves them before compiling the cube.

2. Click **OK**.

Or open the cube class in Studio and compile it in the same way that you compile other classes.

When you compile a cube class, the system automatically generates the fact table and all related classes if needed. If the fact table already exists, the system regenerates it only if it is necessary to make a structural change.

If there are any cached results for this cube, the system purges them.

6.3 Building a Cube

The phrase *building a cube* refers to two tasks: adding data to the fact table and other tables and building the indices used to access this data.

To build a cube in the Architect:

1. Click **Build**.

   The system displays a dialog box.

2. Optionally specify a value for **Maximum Number of Records to Build**.

   By default, DeepSee iterates through all records in the source table and builds the same number of records to the fact table. You can override this behavior when you build the cube. If you specify the **Maximum Number of Records to Build**

option, DeepSee iterates through only that number of records. The result is a smaller fact table that DeepSee builds more quickly.

If the **Maximum Number of Records to Build** field is initialized with a number, that means that the cube class overrides the default behavior. (For details, see the `maxFacts` attribute for `<cube>` in the appendix "Reference Information for Cube Classes.") In this case, you can either use the value provided by the cube class or enter a smaller value.

3. Click **Build**.

   DeepSee starts to build the cube and displays progress as it does so.

4. Click **Done**.

The cube is then available for use as described in *Using the DeepSee Analyzer*.

6.4 Building the Cube Programmatically

To build the cube programmatically, execute the **%BuildCube()** class method of the %DeepSee.Utils class. This method has the following signature:

```
classmethod %BuildCube(pCubeName As %String, pAsync As %Boolean = 1, pVerbose As
%Boolean = 1, pIndexOnly As %Boolean = 1, pMaxFacts As %Boolean = 0, pTracking As
%Boolean = 0, ByRef pBuildStatistics As %String = "") as %Status
```

Where:

- *pCubeName* is the logical name of the cube as given in its XData block; this is not case-sensitive.

- *pAsync* controls whether DeepSee performs the build in multiple background processes. If this argument is true, the system uses multiple processes and does not return until they are all complete. If this argument is false, the system uses a single process and does not return until it is complete.

  Note: If you have specified the cube option **Initial build order**, DeepSee ignores the value of *pAsync* and uses a single process to build the cube. These options are described in "Specifying Cube Options," earlier in this book.

- *pVerbose* controls whether the method writes status information. If this argument is 1, the system writes status updates to the command line. (This argument does not affect whether the method writes build errors or other logging information.)

- *pIndexOnly* controls whether the method only updates the indices. If this argument is 1, the system only updates the indices of the fact table.

- *pMaxFacts* specifies the maximum number of rows in the fact table. This determines the number of rows of the base table that the system uses when building the cube.

  If *pMaxFacts* is 0, the default, all rows of the base table are processed.

- *pTracking* is for internal use.

- *pBuildStatistics* returns an array of information about the cube build. This array has the following nodes:

Node	Value
pBuildStatistics("elapsedTime")	Elapsed build time, in seconds.
pBuildStatistics("errors")	Number of build errors.
pBuildStatistics("factCount")	Number of facts that were built and indexed.
pBuildStatistics("missingReferences")	Number of missing references.
pBuildStatistics("expressionTime")	Time spent processing sourceExpressions to build the cube elements.
pBuildStatistics("iKnowTime")	Time spent building iKnow indices.

This method returns a status. If errors occur during the cube build, the status code indicates the number of build errors.

For example:

```
 set status = ##class(%DeepSee.Utils).%BuildCube("patients")
```

This method writes output that indicates the following information:

- Number of processors used.

- Total elapsed time taken by the build process

- Total amount of time spent evaluating source expressions, summed across all processors.

For example:

```
Building cube [patients]
Existing cube deleted.
Fact table built:       1,000 fact(s) (2 core(s) used)
Fact indices built:     1,000 fact(s) (2 core(s) used)
Complete
Elapsed time:                   1.791514s
Source expression time:         0.798949s
```

If `Source expression time` seems too high, you should re-examine your source expressions to be sure that they are as efficient as possible; in particular, if the expressions use SQL queries, double-check that you have the appropriate indices on the tables that the queries use.

6.5 Minimizing Cube Size During Development

While you are developing a cube, you typically recompile and rebuild it frequently. If you are using a large data set, you might want to limit the number of facts in the fact table, in order to force the cube to be rebuilt more quickly. To do this, do one of the following:

- If you build the cube in the Architect, specify a value for **Maximum Number of Records to Build**.

- Edit the cube class in Studio and add the `maxFacts` attribute to the `<cube>` element. See the appendix "Reference Information for Cube Classes."

  If you do so, be sure to remove this attribute before deployment.

- Build the cube in the Terminal and specify the *pMaxFacts* argument. See "Building a Cube in the Terminal" earlier in this chapter.

6.6 Using Parallel Processing During a Cube Build

If all the following items are true, DeepSee uses multiple cores to perform the build:

- You specify *pAsync* as 1 when you build the cube (see "Building the Cube Programmatically").

- The source for a cube is a persistent class (rather than a data connector). Data connectors are described in the *DeepSee Implementation Guide*.

- The persistent class is bitmap-friendly.

- The **Initial build order** option of the cube has not been set. These options are described in "Specifying Cube Options," earlier in this book.

6.6.1 Specifying the Agent Count

When you build a cube asynchronously, DeepSee sets up a pool of agents to do the work, if it is possible to use parallel processing. This pool consists of a set of agents with high priority and the same number of agents with low priority. By default, the total number of high priority agents (or low priority agents) is four times the number of cores detected on the machine where DeepSee is running.

Note: These agents are also used to execute queries.

To specify a different agent count, call the class method **%SetAgentCount()** in the class %DeepSee.Utils. The argument should be one of the following:

- The desired number of high priority agents — that is, half of the desired total agent count.

- Null — to restore the default behavior.

The method returns the current number of high priority agents — that is, half of the total agent count. Or it can return null, which means that the default behavior is in effect.

For example, the following command sets the total agent count to 10:

```
 d ##class(%DeepSee.Utils).%SetAgentCount(5)
```

To see the current number of agents, use the **%GetAgentCount()** method of the same class. This method returns the current number of high priority agents. Or it can return null, which means that the default behavior is in effect.

When the agents are running, they are visible in the Management Portal as separate processes. To see them, click **System Operation** and then click **Processes**. Look for instances of the %DeepSee.TaskMaster routine. They are not shut down automatically but are intended not to consume resources when idle.

On rare occasions, you might need to reset these agents. To do so, use the **%Reset()** method of %DeepSee.Utils. This method also clears any pending tasks and clears the result cache for the current namespace, which would have an immediate impact on any users. This method is intended for use only during development.

6.7 Build Errors

When you build a cube, pay attention to any error messages and to the number of facts that it builds and indexes. This section discusses the following topics:

- Where you can see build errors

- **Fact count**, which is a useful indicator of build problems in all scenarios

- **Possible causes of build errors**

- **How to recover from build errors**

For more information on troubleshooting options, see the InterSystems Developer Community.

6.7.1 Seeing Build Errors

When you build a cube in the Architect or in the Terminal, DeepSee indicates if there are any build errors. For example:

Status					
Deleting facts			100.00%	Complete	
Building facts	997 fact(s)		100.00%	Complete	
Building indices	997 fact(s)		100.00%	Complete	
Complete				Complete	
Error	ERROR #5001: Error inserting/updating fact: (Source ID:'100') Field 'DeepSee_Model_PatientsCube.Fact.DxBirthDate' (value '10000000') failed validation				

This user interface does not display all the build errors (the same is true if you build a cube in the Terminal).

To see *all* the recorded build errors, do either of the following:

- Look for the log file *install-dir*/mgr/DeepSeeUpdate_*cube_name_NAMESPACE*.log, where *cube_name* is the name of the cube, and *NAMESPACE* is the namespace in which this cube is defined.

  The time stamp in this file uses **$NOW** to write the local date and time, ignoring daylight saving time.

- Use the **%PrintBuildErrors()** method of %DeepSee.Utils, as follows:

  ```
  do ##class(%DeepSee.Utils).%PrintBuildErrors(cubename)
  ```

  Where *cubename* is the logical name of the cube, in quotes.

  This method displays information about all build errors. For example (with added line breaks):

  ```
  SAMPLES>do ##class(%DeepSee.Utils).%PrintBuildErrors("patients")
      1    Source ID: 13
           ERROR #5001: Error inserting/updating fact: (Source ID:'13')
  Field 'DeepSee_Model_PatientsCube.Fact.Dx3295243289' (value 'abc') failed validation

      2    Source ID: 22
           ERROR #5001: Error inserting/updating fact: (Source ID:'22')
  Field 'DeepSee_Model_PatientsCube.Fact.Dx3295243289' (value 'abc') failed validation

      3    Source ID: 37
           ERROR #5001: Error inserting/updating fact: (Source ID:'37')
  Field 'DeepSee_Model_PatientsCube.Fact.Dx3295243289' (value 'abc') failed validation
  ...
  81 build error(s) for 'patients'
  ```

Important: In some cases, DeepSee might not generate an error, so it is important to also check the fact count as discussed in the next section.

6.7.2 Checking the Fact Count

When you build a cube, DeepSee reports the number of facts that it builds and indexes. For example, in the Architect:

Status				
Deleting facts			100.00%	Complete	
Building facts	997 fact(s)		100.00%	Complete	
Building indices	997 fact(s)		100.00%	Complete	
Complete				Complete	
Error	ERROR #5001: Error inserting/updating fact: (Source ID:'100') Field 'DeepSee_Model_PatientsCube.Fact.DxBirthDate' (value '10000000') failed validation				

Each fact is a record in the fact table. The fact table should have the same as the number of records in the base table, except in the following cases:

- You limit the fact count as discussed earlier in this chapter.

- The cube class also defines the **%OnProcessFact()** callback, which you can use to exclude records from the cube. See "Using Advanced Features of Cubes and Subject Areas" in the *Advanced DeepSee Modeling Guide*.

Also, when DeepSee builds the indices, the index count should equal the number of records in the fact table. For example, the Architect should show the same number for **Building facts** and for **Building indices**. If there is a discrepancy between these numbers, check the log files.

6.7.3 Possible Causes of Build Errors

If you see build errors or unexplained discrepancies in the fact count, do the following:

1. Examine any levels that use range expressions, and verify that these levels do not drop records. See "Validating Your Levels," later in this book.

   An error of this kind affects the index count but not the fact count.

2. Try disabling selected dimensions or measures. Then recompile and rebuild to isolate the dimension or measure that is causing the problem.

6.7.3.1 <STORE> Errors

In some cases, the build log might include errors like the following:

```
ERROR #5002: Cache error: <STORE>%ConstructIndices+44^Cube.cube_name.Fact.1
```

This error can occur when a level has a very large number of members. By default, when DeepSee builds the indices, it uses local memory to store the indices in chunks and then write these to disk. If a level has a very large number of members, it is possible to run out of local memory, which causes the <STORE> errors.

To avoid such errors, try either of the following:

- Build the cube with a single process. To do so, use **%BuildCube()** in the Terminal, and use 0 for its second argument.

- In the `<cube>` element, specify `bitmapChunkInMemory="false"` (this is the default). When this cube is built using background processes, the system will use process-private globals instead of local variables (and will not be limited by local memory).

6.7.3.2 Missing Reference Errors

If your cubes have relationships to other cubes, the build log might include errors like the following:

```
ERROR #5001: Missing relationship reference in RelatedCubes/Patients: source ID 1 missing reference to
 RxHomeCity 4
```

This error can mean that you have built the cubes in the wrong order. See "Building Cubes That Have Relationships" in the *Advanced DeepSee Modeling Guide*. Note that if you use the Cube Manager, the Cube Manager determines an appropriate build order.

The `missing relationship reference` error can also occur when new source data becomes available during the cube build process — that is, after only *some* of the cubes have been built. For example, consider the sample cubes RelatedCubes/Cities and RelatedCubes/Patients (which are available in the SAMPLES namespace). Suppose that you build the cube RelatedCubes/Cities, and after that, the source table for RelatedCubes/Patients receives a record that uses a new city. When you build the cube RelatedCubes/Patients, there will be a `missing relationship reference` error.

If you are certain that you have built the cubes in the correct order, see the next section for information on recovering from the errors.

6.7.4 Recovering from Build Errors

DeepSee provides a way to rebuild only the records that previously generated build errors, rather than rebuilding the entire cube. To do this:

1. Correct the issues that cause these errors.

2. Use the **%FixBuildErrors()** methods of %DeepSee.Utils, as follows:

   ```
   set sc=##class(%DeepSee.Utils).%FixBuildErrors(cubename)
   ```

   Where *cubename* is the logical name of the cube, in quotes. This method accepts a second argument, which specifies whether to display progress messages; for this argument, the default is true.

   For example:

   ```
   Fact '100' corrected
   Fact '500' corrected
   Fact '700' corrected

   3 fact(s) corrected for 'patients'
   0 error(s) remaining for 'patients'
   ```

   Or rebuild the entire cube.

6.8 DeepSee Task Log

DeepSee creates an additional log file (apart from the previously described build logs). After it builds the cube or tries to build the cube, DeepSee also writes the DeepSeeTasks_*NAMESPACE*.log file to the directory *install-dir*/mgr. This file contains information about the background agents that DeepSee used during the build process. For example:

```
2009-11-04 16:53:35.648        2312 TaskMaster Background agents killed
2009-11-04 16:53:35.663        2312 TaskMaster Create background agents..
2009-11-04 16:53:35.739        6900 TaskMaster Agent started:1
...
2009-11-04 16:54:19.561        2312 TaskMaster Background agents killed
```

To see this file from the Management Portal, select **DeepSee** > **Tools** > **DeepSee Logs**.

Tip: This file also contains information about runtime errors of various kinds such as listing errors and KPI errors.

The time stamps in this files use the local date and time (taking daylight saving time into account).

7

Defining Dimensions, Hierarchies, and Levels

This chapter describes how to define dimensions and hierarchies, and describes the basics of defining levels. It discusses the following topics:

- Overview

- How to create a new dimension, hierarchy, and level

- How to create a new hierarchy and level

- How to create a new level

- How to specify the source values for a dimension or level

- Details for source expressions

- How to change the order of the dimensions in a cube

- How to change the order of the levels in a hierarchy

- How to validate a level

DeepSee provides many options that affect levels. The next chapter discusses levels in more details.

7.1 Overview

In the Architect, you can create model elements in either of two ways:

- By drag and drop. If you can drag a property from the Class Viewer to an appropriate target in the Model Viewer, the Architect creates a model element; as described in the table after this list. Then you can edit the definition in the Details Area, if needed.

  This technique makes it easy to define elements directly based on source properties. It is also useful as a starting point when you need to create elements based on source expressions.

- By using the **Add Element** link.

For reference, the following table describes the result of various drag-and-drop actions:

If you drop the property XYZ here...	The Architect creates this...
Measures heading or any existing measure	A measure named XYZ, XYZ1 or so on, as needed for uniqueness. The measure is based on the source property XYZ and is aggregated with SUM. See the chapter "Defining Measures."
Dimensions heading	A new data dimension named XYZ, XYZ1, or so on, as needed for uniqueness. This dimension contains the hierarchy H1, which contains the level XYZ. The level is based on the source property XYZ.
Existing dimension	A new hierarchy (H2, for example), which contains the level XYZ. The level is based on the source property XYZ.
Existing hierarchy	A new level named XYZ, XYZ1, or so on, as needed for uniqueness within this hierarchy. The level is based on the source property XYZ.
Existing level	A new level property named XYZ, XYZ1, or so on, as needed for uniqueness within this level. The level property is based on the source property XYZ. See the chapter "Defining Properties."

7.2 Creating a New Dimension, Hierarchy, and Level

To define a usable level, you must define the following, at a minimum:

- A dimension

- A hierarchy in that dimension

- A level in that hierarchy

In the Architect, you can use drag-and-drop actions as described in "Overview." Or, you can do the following:

1. Add a dimension:

   a. Click **Add Element**.

      The system displays a dialog box.

   b. For **Enter New Item Name**, type a dimension name.

      See "Names for Model Elements," earlier in this book.

   c. Click one of the following choices, depending on the type of dimension you want to create:

      - **Data Dimension** — Click this for most dimensions.

      - **Time Dimension** — Click this to create a dimension that groups data by a date or time value. For details, see "Defining a Time Level," later in this chapter.

      - **Age Dimension** — Click this to create a dimension that groups data by age, based on a date value. Note that age dimensions are not generally recommended. For details, see "Defining an Age Level," later in this chapter.

      For information on the **iKnow Dimension** option, see "Using Unstructured Data in Cubes (iKnow)" in the *Advanced DeepSee Modeling Guide*.

   d. Click **OK**.

The system creates the dimension and displays it in the Model Viewer. For example:

▼PatGrpD	data dimension
H1	**hierarchy**
New_Level1	level 1

Note that DeepSee has also created a hierarchy within this dimension and a level within that hierarchy.

2. Modify the automatically created level to use a more suitable name:

   a. Click the level in the Model Viewer.

   The system displays details in the Details Area.

   b. Make the following changes:

   • **Name** — Change this to the level name you want.

   See "Names for Model Elements," earlier in this book.

   • **Display Name** — Specifies the localized display name for this level. Either clear this (so that the system uses the value given for **Name** instead) or specify a display name.

   See also "Other Common Options," earlier in this book.

3. Specify the source values for this level, as described in the section "Defining the Source Values for a Dimension or Level."

4. Click **Save**.

5. When prompted, click **OK**.

Note: By default, the Analyzer does not display hierarchy names shown unless a dimension contains multiple hierarchies. Alternatively, a dimension can be defined so that its hierarchy names are always shown or never shown. See the reference for `showHierarchies` in "<dimension>" in the appendix "Reference Information for Cube Classes."

7.3 Adding a Hierarchy and Level

In the Architect, to add a hierarchy and a level to an existing dimension, you can use drag-and-drop actions as described in "Overview." Or, you can do the following:

1. Click the dimension in the Model Viewer.

2. Click **Add Element**.

   The system displays a dialog box.

3. For **Enter New Item Name**, type a hierarchy name.

   See "Names for Model Elements," earlier in this book.

4. Click **Hierarchy**.

5. Click **OK**.

   The system creates the hierarchy and displays it in the Model Viewer. It also creates one level within that hierarchy.

6. Optionally select the hierarchy in the Model Viewer and edit the details in the Details Area.

7. Select the level in the Model Viewer and edit the details in the Details Area.

7.4 Adding a Level

In the Architect, to add a level to an existing hierarchy, you can use drag-and-drop actions as described in "Overview." Or, you can do the following:

1. Click either the hierarchy or an existing level within that hierarchy, in the Model Viewer.

    This action indicates where the level is to be added:

    *   If you click the hierarchy, the new level will be added after all the other levels in this hierarchy.

    *   If you click a level, the new level will be added immediately before that level.

2. Click **Add Element**.

    The system displays a dialog box.

3. For **Enter New Item Name**, type a level name.

4. Click **Level**.

5. Click **OK**.

    The system creates the level and displays it in the Model Viewer.

6. Optionally select the level in the Model Viewer and edit the details shown in the Details Area.

Alternatively, drag a class property from the Class Viewer and drop it onto a hierarchy. The system adds a level based on this property; this level is added after any other levels in this hierarchy.

Important: The order of the levels in a hierarchy determines the structure of the hierarchy, as noted in "Defining Hierarchies Appropriately," earlier in this book. If you want to reorder the levels after defining them, see "Changing the Order of Levels in a Hierarchy."

7.5 Defining the Source Values for a Dimension or Level

Each level must have a specified source value. You can specify the source values within the dimension or within the level. Typically:

*   For data dimensions, you specify the source values within the level.

*   For time and age dimensions, you specify the source values within the dimension.

*   For iKnow dimensions, see "Using Unstructured Data in Cubes (iKnow)" in the *Advanced DeepSee Modeling Guide*. iKnow dimensions do not use the mechanism described here.

To specify a source value in the Architect:

1. Select the dimension or the level in the Model Viewer.

2. In the Details Area, specify a value for one of the following:

    *   **Property** — Specify the property name relative to the base class used by the cube; you can use dot syntax to refer to properties of properties. For example:

        ```
        Age
        ```

For another example:

```
HomeCity.PostalCode
```

The property must have a projection to SQL.

Also, you can refer to an object-valued property. When you do so, the numeric ID of the object is used as the source value.

You cannot directly use a stream property (to use such a property, create an expression that returns the contents of the stream or a selected part of the contents).

- **Expression** — Specify an ObjectScript expression on which the level is based. For example:

```
##class(Cubes.StudyPatients).GetFavoriteColor(%source.PatientID)
```

This expression is evaluated when the cube is built. For details, see the next section.

You can enter values into the **Property** and **Expression** fields in any of the following ways:

- By dragging a class property from the Class Viewer and dropping it into the field.

  The property that you drag and drop replaces any existing contents of the field.

- By clicking the Search button .

  - For **Property**, the system then displays a dialog box that shows the properties of this class. Click a property and then click **OK**.

  - For **Expression**, the system then displays a dialog box with a larger field you can type into. Type an expression and then click **OK**.

- By directly editing the value in the field.

The information on this section and in the next section also applies to level properties and to measures, which also require source values of some form.

7.5.1 Specifying a Source Value when Using a Data Connector

If the cube is based on a data connector, note the following restrictions:

- Specify the **Property** option but not the **Expression** option.

- You cannot use Caché dot syntax, because none of the properties of the data connector are object references.

For information on creating data connectors, see "Defining Data Connectors" in the *DeepSee Implementation Guide*.

7.6 Details for Source Expressions

As noted in the previous section, you can specify either a source value or a source expression to use as the basis of a dimension or level (or property or measure). You can create source expressions as follows:

- You can refer to a property in the source class. To do so, use the syntax %source.*PropertyName*, where *PropertyName* is the name of the property. When it builds the cube, DeepSee parses this expression and looks up the SqlFieldName of the given property.

- You can use dot syntax to refer to properties of properties.

- You can refer to an object-valued property. When you do so, the numeric ID of the object is used as the source value.

- You can use the variable *%cube* to refer to the cube class; this is useful if you have defined utility methods in the cube class that you want to use within source expressions.

- You can refer to the value of an `<expression>` element. To do so, use the following syntax:

  `%expression.expressionName`

  For information on expressions, see the *Advanced DeepSee Modeling Guide*

- You can use the utility methods **ToUpper()**, **ToLower()**, and **Log()** as in the following example:

  `..ToUpper(%source.HomeCity.Name)`

  DeepSee can use these methods in this way because the fact table class inherits from %DeepSee.CubeFunctionSet, which defines them. For details on these methods, see the class reference for that class.

- You can use the **%Lookup()** method of %DeepSee.CubeDefinition to invoke a term list. See the subsection "Using a Term List."

- You can use the **%Rule()** method of %DeepSee.CubeDefinition to invoke an Ensemble business rule. See the subsection "Using an Ensemble Business Rule."

Note: You cannot use a source expression if the cube is based on a data connector.

Also, if a level is based on a source expression (rather than a source property) and if the members have purely numeric names, InterSystems recommends that you also modify the cube in Studio and add `castAsNumeric="true"` to that level definition. This option causes DeepSee to treat the members as numbers when searching for a replacement for a member that does not exist, when a query that uses an MDX range expression (a specific kind of set expression).

7.6.1 Using a Term List

Term lists provide a way to customize a DeepSee model without programming. A term list is a simple (but extendable) list of key and value pairs. (See "Defining Term Lists" in the *Advanced DeepSee Modeling Guide*.)

You can invoke a term list within a source expression. To do so, use the **%Lookup()** method of %DeepSee.CubeDefinition. This method has the following signature:

`%Lookup(term_list_name, lookup_value,default,alternative_field)`

Where the arguments are strings, and their values are as follows:

- *term_list_name* evaluates to the name of a term list.

- *lookup_value* evaluates to the string to look up in the term list.

- *default*, which is optional, evaluates to the value to return if *lookup_value* is not found in the term list.

- *alternative_field*, which is optional, is the name of the field to return. The default is `"value"`.

  This argument is not case-sensitive.

This function examines the given term list, finds the term whose `"key"` field equals the string given by *lookup_value* and then returns the value contained in the field identified by *alternative_field*.

All term lists have at least two fields: `"key"` and `"value"`. You can add additional fields. For information, see "Defining Term Lists" in the *Advanced DeepSee Modeling Guide*.

Note: Because your cube definition class inherits from %DeepSee.CubeDefinition, which defines the **%Lookup()** method, you can use a source expression like the following:

```
%cube.%Lookup(term_list_name,lookup_value,default,alternative_field)
```

For example, suppose that you have the following term list, called LocalTeams:

Terms

key	value
Atlanta	Braves
Boston	Red Sox
New York	Yankees

You could add a property to the City level in HoleFoods as follows:

```
<property name="Team" sourceExpression='%cube.%Lookup("LocalTeams",%source.Outlet.City,"No Team")'/>
```

7.6.2 Using an Ensemble Business Rule

Business rules allow nontechnical users to change the behavior of Ensemble business processes. You can also use them in source expressions in DeepSee cubes. (For details on Ensemble business rules, see *Using Business Rules with Ensemble*.)

To access an Ensemble business rule within a source expression, use the **%Rule()** method of %DeepSee.CubeDefinition. This method has the following signature:

```
%Rule(rule_name)
```

Where *rule_name* is the name of an Ensemble business rule.

When this function is evaluated (for a given source record) during a cube build, the system passes to it an instance of the cube source class as the context object. The system uses this object, evaluates the rule, and then accesses the value returned by the rule.

Note: Because your cube definition class inherits from %DeepSee.CubeDefinition, which defines the **%Rule()** method, you can use a source expression like the following:

```
%cube.%Rule(rule_name)
```

7.7 Changing the Order of Dimensions in the Cube

In the Architect, to change the order of dimensions in the cube:

1. Click **Reorder**.

    The system displays a dialog box.

2. Click **Dimensions**.

3. Optionally click **Alphabetize** to alphabetize them.

    This affects the list immediately. You can then reorganize the list further if needed. Also, when you add dimensions, they are not automatically alphabetized.

4.   Click the name of a dimension and then click the up or down arrows as needed.

5.   Repeat as needed for other dimensions.

6.   Click **OK**.

The order of the dimensions in the cube affects how they are displayed in the Analyzer. It does not have any other effect. Some customers choose to alphabetize their dimensions for convenience; others put more-often used dimensions at the top of the list.

7.8 Changing the Order of Levels in a Hierarchy

In the Architect, to change the order of levels in a hierarchy:

•   To move a level up in the hierarchy, click the up arrow in the row for that level.

•   To move a level down in the hierarchy, click the down arrow in the row for that level.

Important: The order of the levels in a hierarchy determines the structure of the hierarchy, as noted in "Defining Hierarchies Appropriately," earlier in this book.

7.9 Validating Your Levels

After defining levels, you should build the cube and validate that the levels behave appropriately. For each level, use the Analyzer and create a new pivot table that displays the level as rows.

1.   Click **DeepSee**, click **Analyzer**, and then click **Go**.

   Tip: If the Analyzer is already open, just click the **Analyzer** link at the top of the page.

2.   If the left area is not currently displaying the cube you are validating, click **Open** and choose the cube.

3.   In the left area, find the dimension that contains the level, and expand that dimension.

4.   Drag and drop the level to the **Rows** box.

In this pivot table, look for the following items:

•   Make sure that the level has the correct number of members.

   If the level has duplicate members with the same name, see the section "Duplicate Member Names," earlier in this book.

   If a member is missing, use the Management Portal to find the source records that should have been associated with this member. Make sure that the source property is in the expected form needed by your source expression.

•   Make sure that the member names are in the correct form, as needed by the business users.

•   Make sure that the null member, if any, has a suitable name, as needed by the business users.

•   Make sure that the members are sorted in the desired order.

   If not, see the preceding section.

- If the level uses ranges, do the following to validate that the level is not dropping any records because of errors with range ends:

    1. Click the Pivot Options button ![wrench icon].

    2. In the **Row Options** area, click **Show Totals** and then click **OK**.

    The total line shows the total of the values above it. This total should equal the total number of records in the cube (unless you do not want it to do so). If it does not, examine the ranges carefully to find any gaps. For example, consider the following pivot table (for a cube that contains 10000 patients):

Age Group	
0 to 29	4,179
30 to 59	4,001
60+	1,573
Total	9,753

This version of the Age Group level uses the following ranges:

From	To		Caption (Required)	
(		29	)	0 to 29
(	30	59	)	30 to 59
(	60		)	60+

Patients who are 29, 30, 59, or 60 are not included in any member in this incorrectly defined level.

Tip: In the Analyzer, you can access the generated MDX query for a pivot table (to do so, click the Query Text button ![query text icon]). Then you can save that query to a text file and rerun the query later programmatically. This technique enables you to revalidate the model again later, if the data has changed. Sometimes new data contains values that you did not consider when creating the model.

For information on running queries programmatically, see the *DeepSee Implementation Guide*.

Another useful tool is the **%AnalyzeMissing()** method of %DeepSee.Utils. This method analyzes every non-computed level, measure, and relationship within the given cube and return the number of facts that have no value for each. For example:

```
Do ##class(%DeepSee.Utils).%AnalyzeMissing("patients")

---------------------------------------------------------------------------
    Level                                            #Missing  %Missing
Lvl [AgeD].[H1].[Age]                                       0    0.00%
Lvl [AgeD].[H1].[Age Bucket]                                0    0.00%
Lvl [AgeD].[H1].[Age Group]                                 0    0.00%
Lvl [AllerD].[H1].[Allergies]                             388   38.80%
Lvl [AllerSevD].[H1].[Allergy Severities]                388   38.80%
Lvl [BirthD].[H1].[Date]                                    0    0.00%
Lvl [BirthD].[H1].[Decade]                                  0    0.00%
Lvl [BirthD].[H1].[Period]                                  0    0.00%
Lvl [BirthD].[H1].[Quarter Year]                            0    0.00%
Lvl [BirthD].[H1].[Year]                                    0    0.00%
Lvl [BirthQD].[H1].[Month]                                  0    0.00%
Lvl [BirthQD].[H1].[Quarter]                                0    0.00%
Lvl [BirthTD].[H1].[Birth Time]                             0    0.00%
Lvl [ColorD].[H1].[Favorite Color]                       221   22.10%
Lvl [DiagD].[H1].[Diagnoses]                              831   83.10%
Lvl [DocD].[H1].[Doctor]                                  159   15.90%
Lvl [DocD].[H1].[Doctor Group]                           271   27.10%
Lvl [GenD].[H1].[Gender]                                    0    0.00%
Lvl [HomeD].[H1].[City]                                     0    0.00%
Lvl [HomeD].[H1].[ZIP]                                      0    0.00%
```

```
Msr [Measures].[Age]                              0      0.00%
Msr [Measures].[Allergy Count]                  388     38.80%
Msr [Measures].[Avg Age]                          0      0.00%
Msr [Measures].[Avg Allergy Count]              388     38.80%
Msr [Measures].[Avg Enc Count]                    0      0.00%
Msr [Measures].[Avg Test Score]                 200     20.00%
Msr [Measures].[Encounter Count]                  0      0.00%
Msr [Measures].[Test Score]                     200     20.00%
Lvl [PatGrpD].[H1].[Patient Group]                0      0.00%
Lvl [PatGrpD].[H1].[Tested]                       0      0.00%
Lvl [ProfD].[H1].[Industry]                     564     56.40%
Lvl [ProfD].[H1].[Profession]                   564     56.40%

    TOTAL FACTS                               1,000
```

8

Details of Defining Levels

This chapter provides details on defining levels. It discusses the following topics:

- How to ensure that member keys are unique

- How to override the default null replacement string for level

- How to define a level based on a list

- How to define a time level

- How to define a custom time level

- How to define an age level

- How to specify a range expression for a level

- How to configure a level to use the DISPLAYLIST values of a class property

- How to use property values as member names

- How to specify tooltips for members

- How to control the sort order for the members

- How to make member names localizable in the same way as other cube elements

- How to define dependencies between levels in different hierarchies

- How to specify field names in the fact table

See the previous chapter for basic information, including information on specifying the source values.

8.1 Ensuring Uniqueness of Member Keys

If a level has a parent level, it is possible for member keys *not* to be unique. For details, see "Defining Member Keys and Names Appropriately," earlier in this book. InterSystems recommends that you ensure that member keys are unique for each level, so that you can easily access any member directly. Also, member keys must be unique within the first 113 characters.

If it is possible, considering your source values and your hierarchies, to have duplicate member keys, do the following:

- Instead of using just the source value alone, concatenate the source value with some other value that is unique to the parent member. For example, suppose that you want to base the level on a source property called CityName, but you

could have multiple cities with the same name in different countries. Instead of specifying **Property**, specify **Expression**. Use an expression like the following:

```
%source.CountryName_%source.CityName
```

- If you do not want to use these concatenated strings as member names, use a unique value (such as the ID) as the source value. Then add a property to the level and use the property values as the member names. See "Using Property Values as the Member Names," later in this chapter.

8.2 Specifying the Null Replacement String for a Level

For a level, the **Null replacement string** option specifies a string to use in place of any null values. This string takes precedence over the null replacement string that you specify for the cube.

8.3 Defining a List-Based Level

You can define a level that is based on a list, an array, or a Caché class relationship. Each distinct list item is indexed as a separate member of the level. (For such levels, be sure to read "Using List-Based Levels," earlier in this book.)

DeepSee can directly use a source value that has the type %List, that is in the format returned by the **$List** function, or that is a character-delimited list. For other formats, you must convert the source value and then use it.

Note: A list-based level must be contained in its own hierarchy; there cannot be another level in this hierarchy.

8.3.1 Defining a Level When the Source Value Is in a Standard Format

If the source value is available in **$List** format, in %List format, or in a character-delimited list, define the level as described earlier, with the following additional steps:

1. For **Source value list type**, select one of the following:

    - **$List structure** — Use this if the source value is in the format returned by the **$List** function or has the type %List.

    - **Comma delimited** — Use this if the source value is a comma-separated list.

    - **Other delimited** — Use this if the source value is a list where the delimiter is some other character.

2. If you selected **Other delimited**, then for **List delimiter**, specify the character used to delimit the list.

8.3.2 Defining a Level When the Source Value Is in Another Format

If the source value is not available in **$List** format or as a character-delimited list, use the following approach:

1. Create a utility method that converts it into one of the required formats. For example:

```
ClassMethod GetAllergies(ID As %Numeric) As %List
{
    Set allergies=##class(DeepSee.Study.Patient).%OpenId(ID,0).Allergies
    If (allergies.Count()=0) {Quit $LISTFROMSTRING("No Data Available")}
    Set list=""
    For i=1:1:allergies.Count() {
        Set $LI(list,i)=allergies.GetAt(i).Allergen.Description
        }
    Quit list
}
```

Notice the following points:

- The %List class is equivalent to the **$List** format.

- The second argument to **%OpenId()** specifies concurrency locking. If this argument is zero, you are opening the object without checking for other users of the object, for speed. Because you are only reading a value from the object, this technique is safe even in a multi-user environment.

- If there are no allergies, this method returns the string `No Data Available`, which becomes a member of the level, in exactly the same way as other returned strings.

Tip: You can place this utility method within the cube class. Also, you can easily test the method within the Terminal.

2. Within the Architect, when defining the level, do the following:

a. For **Expression**, invoke the utility method and pass *%source.%ID* to it as an argument. For example:

```
##class(Cubes.StudyPatients).GetAllergies(%source.%ID)
```

b. Select **List**.

c. If the source value is a character-separated list, then for **List delimiter**, specify the character used to delimit the list.

If this is null, DeepSee assumes that the source value is in **$List** format.

8.3.3 Examples

In the SAMPLES namespace, the Patients cube provides the following examples of list-based levels:

- The `Allergies` level is based on a source expression that uses a utility method to return a list.

- The `Allergy Severities` level is based on a source expression that uses a utility method to return a list.

- The `Diagnoses` level uses a property that is in **$List** format.

8.4 Defining a Time Level

A time level groups the data according to a date/time value in the data (such as a birth date, order date, response date, and so on). This section describes how to create this type of level. It discusses the following topics:

- Introduction

- How to define a time level

- Time levels and hierarchies

- Handling a calendar that has a date offset

- Where to find examples

8.4.1 Introduction

A time level is a level defined within a time dimension. For an example of a set of time levels, see the `BirthD` dimension in the Patients cube. The `BirthD` dimension is based on the BirthDate property in the source class. This property contains the patient's birth date.

Each level in this dimension uses an option called **Time function** to extract a specific part of the birth date. For example, the `Year` level extracts the year part of the date.

As a result:

- The `Decade` level has members such as `2010s`, `2000s`, `1990s`, and so on.

- The `Year` level has members such as `2009`, `2008`, and so on.

- The `Quarter Year` level has members such as `Q1 2009`, `Q2 2009`, and so on.

- The `Period` level has members such as `2009-01`, `2009-02`, and so on.

- The `Date` level has members such as `Jan 1 2009`, `Jan 2 2009`, and so on.

- The order of levels within the hierarchy `H1` establishes that the `Decade` level is the parent of the `Year` level. For example, the member `1990s` is the parent of the members `1990`, `1991`, `1992`, and so on.

    Similarly, the `Year` level is the parent of the `Quarter Year` level. For example, the member `2009` is the parent of the members `Q1 2009`, `Q2 2009`, and so on.

With time levels, you can use a special member called NOW, which uses the current date (runtime) and accesses the appropriate member of the level.

Note: By default, DeepSee uses the Gregorian calendar. You can instead use a Hijri calendar. In that case, the member names are different from those listed here (and the records are assigned to members as suitable for that calendar).

8.4.1.1 Using the Locale to Control the Names of Time Members

When you build a cube, DeepSee can use the current server locale to determine the names of members of any time dimensions. It is useful to check the relevant option, as follows:

1. In the Management Portal, select **System Administration** > **Configuration** > **National Language Settings** > **Locale Definitions**.

    This page displays the current server locale.

2. Examine the setting of the option **Use locale date/time/number formats** and change it if needed.

    If this option is **Yes**, then DeepSee considers the locale shown on this page when it builds or synchronizes the members of time dimensions.

If you later change the locale, it is necessary to recompile and rebuild the cube.

Also, see "Locale Definitions" in the *Caché Additional Configuration Settings Reference*.

8.4.2 Defining a Time Level

To define a time level:

1. Define a dimension as described earlier, with two changes:

    - For **Dimension Type**, choose **time**.

    - For **Calendar**, choose **gregorian** (the default) , **hijri (tabular)**, or **hijri (observed)**.

This option specifies the calendar to use when assigning source records into members of levels of this dimension.

- Specify either **Property** or **Expression**.

  If you specify **Property**, the property should be one of the following types: %Date, %MV.Date, %Time, %TimeStamp, %FilemanDate, %FilemanTimeStamp, or a custom data type that meets the requirements given in the second sub-section.

  If you specify **Expression**, the source expression must return data in **$Horolog** format. The value can be null but cannot equal 0 (the value 0 causes a validation error when the cube is built).

  When you define the dimension, the system creates the dimension and a hierarchy.

2. Define a level within this hierarchy.

   In this case, you will see different options than you saw for a data dimension.

3. For **Extract value with function**, choose a function that extracts the desired part of the date/time value. Use one of the values in the following table. Note that for a Hijri calendar, not all functions are supported.

Time function	How Value Is Stored	How Value Is Displayed	Available in Hijri Time Dimensions?
MinuteNumber	60958,5083	01:24	*Yes*
HourNumber	60958,5083	1am	Yes
DayMonthYear	60958	Nov 24 2007	Yes
DayNumber	24	24	Yes
WeekYear	2007W47	2007W47	No
WeekNumber	47	47	No
MonthNumber	11	November	*Yes*
MonthYear	200711	November 2007	Yes
QuarterNumber	4	Q4	Yes
QuarterYear	20074	Q4 2007	Yes
Year	2007	2007	Yes
Decade	1950	1950s	*Yes*
DayOfWeek	1	Sunday	*No*

4. For **Time Format**, optionally specify a format string that describes how the dates should be displayed. For details, see the first subsection.

5. If you are using the Hijri (observed) calendar, use the class %Calendar.Hijri to add lunar observations.

   For details, see the class reference for %Calendar.Hijri.

For the **WeekYear** and **WeekNumber** functions, DeepSee follows the ISO 8601 standard, which assigns a week number (from 1 to 53) for every date within a year. Because 7 is not a factor of 365 or 366, some years have 53 weeks. The first week of a year is defined as the week that contains the first Thursday of the year. That means for some years, January 1 is in the last week of the prior year. Similarly, December 31 may be in the first week of the next year.

8.4.2.1 Specifying a Time Format

For a level within a time-type dimension, the **Time Format** specifies the format of the display names of the members of the level. DeepSee applies this formatting at query execution time. There is no effect on how values are stored or indexed.

For the **Time Format** attribute, you specify a string that consists of the following case-sensitive date pieces and other pieces:

Piece	Replaced By
y	Year number
q	Quarter number
mmmm	Full month name
mmm	Short month name
mm	Month number with leading 0 if needed
m	Month number without a leading 0
dddd	Full day name
ddd	Short day name
dd	Day number of the month with leading 0 if needed
d	Day number of the month without a leading 0
\x	x
period (.), slash (/), hyphen (-). space	Unchanged
other characters	Ignored

All names are based on the current server locale. See "Using the Locale to Control the Names of Time Members," earlier in this chapter.

Important: For time levels, member display names must be unique. Also, a member can have exactly one display name. These rules mean that not all the preceding date pieces are appropriate for all time levels.

The following table lists the date pieces that are suitable for different time levels.

Kind of Level (Setting for **Extract value with function**)	Suitable Date Pieces	Default `timeFormat`	Other Examples
Year	y	y (2004)	\F\Y y (FY 2004) \F\Y (FY2004)
QuarterYear	y, q	\Qq y (Q3 2004)	\Qq \F\Y y (Q1 FY 2004) \Qq \F\yy (Q1 FY2004)
QuarterNumber	q	\Qq (Q3)	

Kind of Level (Setting for **Extract value with function**)	Suitable Date Pieces	Default `timeFormat`	Other Examples
`MonthYear`	y, mmmm, mmm, mm, m	`mmmm y` (February 2004)	`mmm y` (Feb 2004) `y-mm` (2004–02) `mm/y` (02/2004) `m/y` (2/2004) `\F\Yy-mm` (FY2004-02)
`MonthNumber`	mmmm, mmm, mm, m	`mmmm` (February)	`mmm` (Feb) `mm` (02) `m` (2)
`DayMonthYear`	y, mmmm, mmm, mm, m, dddd, ddd, dd, d	`mmm dd y` (Feb 1 2004)	`mmmm dd y` (February 03 2010) `y-mm-dd` (2010-02-03) `mm/dd/y` (02/03/2010) `m/d/y` (2/3/2010) `dddd, mmmm dd y` (Wednesday, February 03 2010) `ddd, mmm dd y` (Wed, Feb 03 2010)
`DayNumber`	dd, d	`d` (1)	`dd` (01)
`DayOfWeek`	dddd, ddd, dd, d	`dddd` (Tuesday)	`dd` (03) `d` (3)

8.4.2.2 Using a Custom Data Type

You can base a time level directly on a property that uses a custom data type, in any of the following scenarios:

- In the first scenario, the data type stores data as **$Horolog** *date* (that is, as the *first* part of a **$Horolog** string). The results depend on the ClientDataType of the data type class:

  - If ClientDataType is DATE or is unspecified, DeepSee does not convert the value.

  - If ClientDataType is MVDATE, DeepSee treats the value as an MV date and converts it to a **$Horolog** date (with no time).

  - If ClientDataType is FDATE, DeepSee treats the value as a Fileman date (%Library.FilemanDate) and converts it to a *full* **$Horolog** date.

  - If ClientDataType is FTIMESTAMP, DeepSee treats the value as a Fileman timestamp (%Library.FilemanTimeStamp) and converts it to a *full* **$Horolog** date.

- In the second scenario, the data type stores data in ODBC date format. The ClientDataType of the data type class must be TIMESTAMP. DeepSee converts the value to a *full* **$Horolog** date.

- In the third scenario, the data type stores data as a **$Horolog** *time* (that is, as the *second* part of a **$Horolog** string). The ClientDataType of the data type class must be TIME. DeepSee converts the value to a *full* **$Horolog** date. (The date is arbitrarily chosen, so you should use this data only for hour and minute levels.)

See the implementation of **%ConvertDate()** in %DeepSee.Utils.

8.4.3 Time Levels and Hierarchies

As noted earlier, the order of levels in a hierarchy affects how the members of the levels are created. For any two adjacent levels in the same hierarchy, the first level (level A) becomes the parent of the second level (level B). The hierarchy is a parent-child hierarchy, which means the following:

- Any member of level A is the parent of one or more members of level B.

- Any member of level B is the child of exactly one member of level A.

- Any record that belongs to a given member of level B must always belong to the same member of level A.

Therefore, a level based on **Year** cannot be the parent of a level based on **Quarter**, for example. Consider two patients, one born in Q3 2007 and one born in Q3 1982. These two patients both belong to the same member (Q3) of the Quarter level but belong to different members (2007 and 1982, respectively) of the Year level.

As another example, a level based on **Month** cannot be the parent of a level based on **WeekNumber**. Since a week can fall between two months, any record that belongs to a week member does not necessarily belong to the same month member.

The following table lists time levels and their typical child levels. For reference, it also shows example members of each level:

A Level Based on the Time Function...	Has Members Like This...	Child Level Is Typically Based on the Time Function...	Notes
Decade	1950s	**Year**, **QuarterYear**, **MonthYear**, or **DayMonthYear**	These levels use all parts of the date
Year	2007	**QuarterYear**, **MonthYear**, or **DayMonthYear**	
QuarterYear	Q4 2007	**MonthYear** or **DayMonthYear**	
MonthYear	November 2007	**DayMonthYear**	
WeekYear	2007W47	**WeekNumber**	
DayMonthYear	Nov 24 2007	No typical child level	
QuarterNumber	Q4	**MonthNumber**	These levels are independent of the year
MonthNumber	November	No typical child level	
WeekNumber	47	No typical child level	
DayNumber	24	No typical child level	These levels are independent of the year and independent of the part of the year
DayOfWeek	Wednesday	No typical child level	
HourNumber	1am	**MinuteNumber**	These levels are independent of the day
MinuteNumber	01:24	No typical child level	

8.4.4 Handling a Calendar That Has a Date Offset

In some cases, you may need a time level to match a financial calendar that includes a date offset. For example, in many companies, the financial year starts 1 Oct. Consider the following pivot table:

FY 2002	FY 2002-Q1	Oct-2001	14
		Nov-2001	11
		Dec-2001	7
	FY 2002-Q2	Jan-2002	12
		Feb-2002	9
		Mar-2002	10
	FY 2002-Q3	Apr-2002	9
		May-2002	12
		Jun-2002	12
	FY 2002-Q4	Jul-2002	13
		Aug-2002	9
		Sep-2002	15

In the innermost grouping, this pivot table groups the records by the actual period (year and month, combined). For example, there are 14 records associated with the actual period October 2001. The periods are grouped into fiscal quarters, and the fiscal quarters are grouped into fiscal years. Note that the quarter FY 2002–Q1 includes October, November, and December of 2001.

For such a time level, edit the cube class in Studio and specify the `timeOffset` and `timeFormat` properties of the `<level>` element. For the example shown here:

- For the first two levels (the levels with members such as FY 2002 and FY 2002–Q1), `timeOffset` is `"-3m"` (which subtracts three months from the date values used by these levels).

- For the third level (the level with members such as Oct-2001), `timeOffset` is not specified, and thus this level uses the actual date values.

For information, see "Specifying a Date Offset" in the appendix "Reference Information for Cube Classes."

8.4.5 Examples

The SAMPLES namespace provides the following examples of time levels:

- In the Patients cube, see the `BirthD`, `BirthQD`, and `BirthTD` dimensions.

- In the HoleFoods Sales cube, see the `DateOfSale` level.

8.5 Defining Custom Time Levels

You can define custom levels that use time in other ways. To do so:

- Create a dimension whose **Type** is **data** (not **time**).

- For each level, specify a value for **Expression** that returns the desired value.

- See "Specifying the Sort Order for the Members," later in this chapter.

  By default, the members are sorted alphabetically.

8.5.1 Example

In the Patients cube in the SAMPLES namespace, see the `BirthWeekdayD` dimension. For this dimension, **Expression** is as follows:

```
$system.SQL.DAYNAME(%source.BirthDate)
```

This method executes the **DAYNAME()** method in the %SYSTEM.SQL class. That class provides a large set of methods that handle date values.

With this dimension, you can create pivot tables like this:

Weekday	Patient Count
Sunday	147
Monday	140
Tuesday	141
Wednesday	141
Thursday	140
Friday	150
Saturday	141

This sample dimension defines only a single level. With the appropriate logic, you can define a hierarchy with weeks, months, and quarters in a 4–4–5 calendar, for example.

8.6 Defining an Age Level

An age level groups the records according to an age value in the data, relative to the cube build date. For example, it could group patients by their ages (as computed at cube build time).

Note: To keep age levels accurate, it is necessary to rebuild the cube; the DeepSee cube synchronization feature has no effect on them. Thus, age levels are not generally recommended. If you need to group records by age, InterSystems recommends that you instead define calculated members; see "Defining Age Members," later in this book.

To define an age level:

1. Define a dimension as described earlier, with two changes:

    - For **Type**, choose **age**.

    - Specify either **Property** or **Expression**.

      Important: If you specify **Property**, the property must be in **$Horolog** format or must be of type %TimeStamp (or a subclass).

      If you specify **Expression**, the source expression must return data in **$Horolog** format.

    When you define the dimension, the system creates the dimension and a hierarchy.

2. Define a level within this hierarchy.

In this case, you will see different options than you saw for a data dimension.

3. For **Extract value with function**, choose a function that extracts the desired part of the time value. Use one of the following values:

   - `"Days"` – Use this to determine the age in days.

   - `"Months"` – Use this to determine the age in months.

   - `"Years"` – Use this to determine the age in years.

Note that the Patients sample does *not* demonstrate this type of level; the `Age`, `Age Bucket`, and `Age Group` levels group patients in the fictitious study according to their ages *at the date of the study*; this is common practice with retrospective studies.

8.7 Specifying a Range Expression
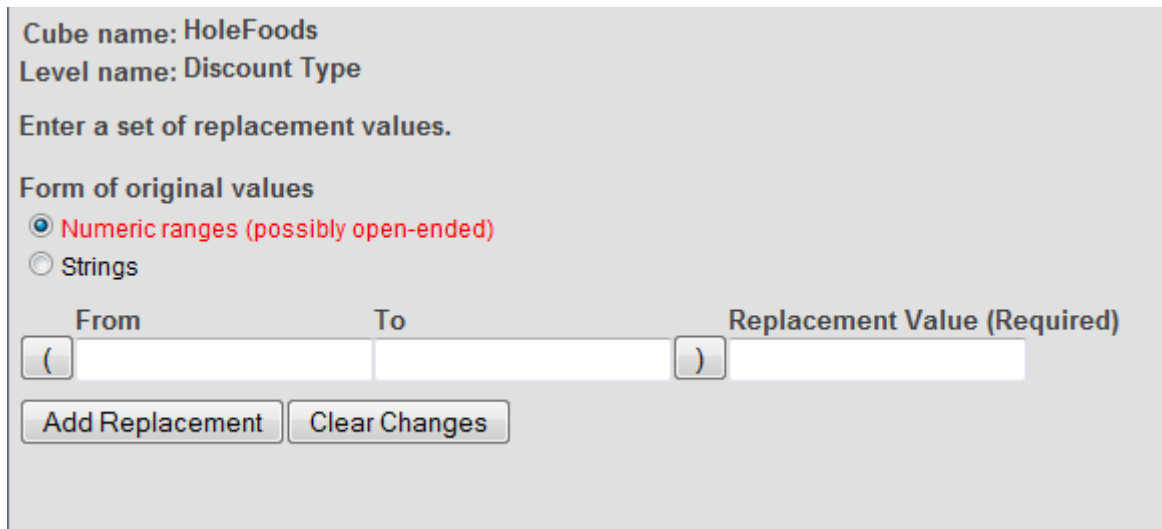
For any level in a data dimension, you can use the **Range expression** option. This option lets you use new values in the place of the actual source values. For numeric data, this transformation can replace source values with discrete bins. For any data, this transformation can specify replacement strings.

In either case, the new values become the names of the members of the level. The values also become the keys for the members.

To specify a range expression:

1. Click the Find button 🔍 next to the **Range expression** field.

   The Architect displays a page like the following:

2. For **Form of original values**, choose either **Numeric** or **Strings**.

   This choice affects the form of this page.

3. Add a series of replacements as follows:

   - To add a row to the end of the table, click **Add Replacement**.

     Then enter details as given in the following subsections.

- To remove a row, click the X button for that row.

The order of the rows determines the default sort order of the members of this level.

4. Click **OK** to close this page.

    The **Range expression** field now shows something like this (for numeric bins):

    ```
    (,0]:None;(0,0.2):1-19%;[0.2,0.5):20-49%;[0.5,1]:50%+;
    ```

    Or this (for string data):

    ```
    NONE:None;MINR:Minor;SERS:Serious;FATL:Fatal;UNK:Unknown;
    ```

To make member names localizable, see "Manually Specifying the Members of a Level," in the chapter "Using Additional Advanced Features of Cubes and Subject Areas" in the *Advanced DeepSee Modeling Guide*.

8.7.1 Defining Numeric Bins

When you define numeric bins, the Range Expression editor looks like this:

Each row in this table defines a numeric bin as follows:

-

  The button at the left end of the line indicates the form of the lower limit — inclusive [or exclusive (.

  When there is no lower limit, this option has no effect.

  If the lower limit is *inclusive*, that means that exact value of **From** is included in the range. If the lower limit is *exclusive*, that means that **From** is not in the range.

- **From** indicates the lower limit of the bin, if any.

- **To** indicates the upper limit of the bin, if any.

-

  The button at the right end of the line indicates the form of the upper limit — inclusive] or exclusive).

  When there is no upper limit, this option has no effect.

- **Replacement Value** specifies the string to use as a member name for this level. Any record whose source value falls within the defined range is assigned to this member.

  This value is also the key for the member.

For this level, DeepSee ignores records whose source values do not fall into any of the given bins.

8.7.2 Defining String Replacements

When you define string replacements, the Range Expression editor looks like this:

Each row in this table defines a string replacement as follows:

- **Original Value** specifies a possible value of the source property or source expression of this level.

- **Replacement Value** specifies the string to use as a member name for this level. Any record whose source value matches the given **Original Value** is assigned to this member.

  This value is also the key for the member.

For this level, DeepSee ignores records whose source values do not match any of the given original values.

8.7.3 Examples

The SAMPLES namespace provides the following examples of range expressions:

- In the Patients cube, see the `Age Group` level.
- In the HoleFoods Sales cube, see the `Discount Type` level.

8.8 Configuring a Level to Use Display Values

A Caché class property can have both a stored value and a displayed value, via the *VALUELIST* and *DISPLAYLIST* property parameters. For example:

```
Property Gender As %String(VALUELIST = ",F,M", DISPLAYLIST = ",Female,Male");
```

When you use such a property as a level, by default, the system uses the value given in the *VALUELIST* parameter.

In the level definition, if you select the **Use value in DISPLAYLIST** option, the system uses the value given in the *DISPLAYLIST* parameter instead. For example:

8.8.1 Examples

The SAMPLES namespace provides the following examples of levels that use the **Use value in DISPLAYLIST** option:

- The `Gender` level in the Patients cube.

- The `Channel Name` level in the HoleFoods Sales cube.

8.9 Using Property Values as the Member Names

By default, the source values for a level become the names for the members of the level.

You can instead specify the member names by defining a property for this level and using the values of that property as the member names. This is useful in the following scenarios:

- *To enable member names to be accessed at runtime* — Values of properties can be accessed at runtime (the same is not true for levels).

- *To provide user-friendly member names for members* — In some cases, you must base a level on a unique value that is not user-friendly. For example, suppose that you have a level based on the patient's primary care physician. People's names are not reliably unique, so you would have to base the level on a unique doctor identifier instead, which might have no meaning for the users. In this case, you would also define a property that accesses the doctor's name, and you would use that property as the name of the member.

To use property values as the names for members of a level:

1. Define a property for this level, as described in the next chapter.

    Each member of the level has a value for this property.

2. For this property, select the **Use as member names** option.

3. Optionally, to retrieve the property value at runtime, select the **Get value at run time** option.

    If you use this option, note the following requirement: For the parent level (the level that contains the property), the source property or source expression of that level must evaluate to an ID. The system assumes that (at least for this level), the source data is normalized. That is, for the level, the data is in a different table and the source table contains a link to that table.

8.9.1 Examples

The SAMPLES namespace provides the following examples of levels that have properties with **Use as member names**, **Get value at run time**, or both:

- The `DxDoc` level in the Patients cube. This level has the property `Name`, which is configured with **Use as member names**. The `DxDoc` level is based on the unique ID of the doctor. Depending on the number of patients you generate, you may see multiple doctors with the same name. Because of how this level is defined, those doctors are not combined with each other.

- The `City` level in the Patients cube. This level has the property `Name`, which is configured with **Use as member names** and with **Get value at run time**.

- The `City` and `Channel Name` levels in the HoleFoods Sales cube. These levels have properties that are configured with **Use as member names**.

- The `Product Name` level in the HoleFoods Sales cube. This level has a property that is configured with **Use as member names** and with **Get value at run time**.

8.10 Specifying Tooltips for Members

When a user hovers the cursor near a member name in the Analyzer, DeepSee displays a tooltip. By default, the member name is the tooltips. You can specify additional information to display in these tooltips for any level. To do so:

1. Add a property to that level. For the source value, use a value that contains the desired tooltips.

2. In Studio, edit the cube definition and modify the newly added property. For this property, specify the `isDescription` attribute as `"true"`. See <property> in the appendix "Reference Information for Cube Classes."

Now the tooltip is the member name, followed by a hyphen, followed by the value of the specified property. For example:

This is useful when you want to use short member names, for reasons of space, but also need to provide a fuller description of each member.

8.11 Specifying the Sort Order for Members

This section describes how to control the sort order for members of a level.

8.11.1 For a Data Level

For a data level, by default, members are sorted as follows:

• If the level does not use a range expression, the members are sorted in increasing order by name (using alphabetic sort, no matter what form the names have).

- If the level uses a range expression, the members are sorted in the order determined by the range expression. That is, the first member is determined by the first replacement in the range expression, the second member is determined by the second replacement, and so on.

To change the sort order for the members of a data level, use any of the techniques described in the following subsections.

8.11.1.1 Specifying the Level Sort Option

The simplest way to specify the sort order for members of a level is to set the **Sort** option for the level. Select **asc**, **asc numeric**, **desc**, or **desc numeric**.

asc and **desc** sort members *alphabetically*, in ascending or descending order, and **asc numeric** and **desc numeric** sort *numerically*, in ascending or descending order.

8.11.1.2 Sorting Members by Property Values

To sort the members of a level in order by property values:

1. Define a property for this level, as described in the next chapter.

   Each member of the level has a value for this property.

2. For this property, select **asc**, **asc numeric**, **desc**, or **desc numeric** for **Sort members by property value**.

   asc and **desc** sort members alphabetically, in ascending or descending order; **asc numeric** and **desc numeric** sort numerically, in ascending or descending order.

   (By default, no value is selected, and the property does not affect the sort order of the members.)

If you specify **Sort members by property value** within more than one property of a level, the members are sorted by the first of those properties. Then they are subsorted by second of those properties, and so on.

For an example, see the Patients cube in SAMPLES namespace. In this cube, the `Allergy Severities` level has a property that is configured with **Sort members by property value**. This property is based on a source expression that returns a numeric value.

8.11.1.3 Using Property Names as Member Names

An alternative approach, similar to the preceding, is as follows:

1. Use a source expression for the level that forces the members in the desired order, even if the member names are not desirable.

2. Define a property for this level, as described in the next chapter.

3. For this property, specify a source value or source expression that returns a better name for each. For this property, select the option **Use as member names**.

8.11.1.4 Listing the Members in the Desired Order

To list the members in the desired order, edit the cube class to add `<member>` elements to the level. For information, see "Manually Specifying the Members of a Level," in the chapter "Using Additional Advanced Features of Cubes and Subject Areas" in the *Advanced DeepSee Modeling Guide*.

8.11.2 For a Time Level

By default, members of a time level as sorted in increasing order by time.

To specify a different sort order for the members, specify **Sort option**. Choose either **asc** (ascending in time) or **desc** (descending in time).

8.12 Making the Member Names Localizable

For cubes as well as dimensions, hierarchies, levels, and other cube elements, you can specify the **Display name** value, for which you can later define translations in different languages. The same technique is available for members of a level, if you know all the members in advance and if you edit the cube class in Studio. See "Manually Specifying the Members of a Level," in the chapter "Using Additional Advanced Features of Cubes and Subject Areas" in the *Advanced DeepSee Modeling Guide*.

For a time dimension, the current locale determines the names of members of a time dimension. (See "Using the Locale to Control the Names of Time Members," earlier in this book.) If you change the locale, it is necessary to recompile and rebuild the cube.

8.13 Defining Dependencies Between Levels in Different Hierarchies

In some cases, there is a virtual dependency between levels that are not in a hierarchy with each other. For example, you might have a cube with a Country level and a Product level. These levels are logically independent from each other; theoretically any product could be sold in any country. It would not make sense to put these levels into a hierarchy in the same dimension.

But if specific products are sold only in specific countries, there is a virtual dependency between these levels. When a user selects a country, it is desirable to show only the products appropriate for that country. In such a case, you can add a dependency between the levels as follows:

1. Open the cube class in the Architect.

2. Find the definition of the dependent level.

3. Edit the value of the **Depends on** option. Use a value like the following:

   ```
   [dim].[hier].[level]
   ```

   Where *dim* is the name of the other dimension, *hier* is the name of the hierarchy, and *level* is the name of the level. `[dim].[hier].[level]` is the MDX identifier of the level.

   For example:

   ```
   [Region].[H1].[Country]
   ```

   If the level depends on multiple levels, specify a comma-separated list. For example:

   ```
   [dim1].[hier1].[level1],[dim2].[hier2].[level2]
   ```

   You could also specify a relationship instead of a level.

This option attribute adds another index to the fact table and consequently consumes more disk space. Use this attribute sparingly. Also, this option is completely unrelated to the DependsOn compiler keyword.

8.13.1 Example

To see how this feature works, try the following demonstration:

1. In the Architect, open the Patients cube.

2. Edit the `Patient Group` level of the `PatGrpD` dimension. Specify **Depends on** as follows:

    ```
    [HomeD].[H1].[ZIP]
    ```

3. Save and compile the cube.

4. In the Terminal, execute the **ReassignPatients()** method in DeepSee.Populate, in the SAMPLES namespace:

    ```
    d ##class(DeepSee.Populate).ReassignPatients()
    ```

    This method modifies the data for this sample so that there is a virtual dependency between the Patient Group level and the ZIP code level. Patients in specific ZIP codes belong either to Patient Group A or no group; patients in the other ZIP codes belong either to Patient Group B or no group.

    The method synchronizes the cube, so there is no need to rebuild this cube after executing this method.

Then open the Basic Dashboard Demo and try the Home ZIP Code and Patient Group filters. When you choose a ZIP code, that affects the list in the Patient Group filter.

8.14 Specifying the Field Names in the Fact Table

When you compile a cube class, DeepSee generates a fact table class and some related classes. When you build a cube, DeepSee populates these tables, which are described in "Details for the Fact and Dimension Tables."

By default, DeepSee generates the names of the fields in the fact table, but you can specify the field names to use instead. To do so, specify a value for the **Field name in fact table** option for each applicable level. Take care to use unique names. This option is not available for time levels or iKnow levels.

Important: For **Field name in fact table**, be sure not to use an SQL reserved word. For a list of the SQL reserved words, see "Reserved Words" in the *Caché SQL Reference*. The name must start with either a letter or a percent sign (`%`). If the first character is `%`, the second character must be Z or z. For more details on restrictions, see "Class Members" in the appendix "Rules and Guidelines for Identifiers" in the *Caché Programming Orientation Guide*. Also, do not use `fact` or `listing`, in any combination of lowercase and uppercase characters.

9

Defining Properties

This chapter describes how to define properties. It discusses the following topics:

- How to create a property

- How to get the value of a property at runtime

- How to change the order of properties in a level

- How to specify column names in the dimension tables

- A summary of special uses for properties

Note: Time and age dimensions cannot contain properties.

9.1 Adding a Property

To add a property to an existing level:

1. Click either the level or an property within that level, in the Model Viewer.

    This action indicates where the property is to be added:

    - If you click the level, the new property will be added after all the other properties in this level.

    - If you click a property, the new property will be added immediately before that property.

    Note: The order of the properties in a level is important if the properties are used to sort members. See "Specifying the Sort Order for Members of a Data Level," in the previous chapter.

2. Click **Add Element**.

    The system displays a dialog box.

3. For **Enter New Item Name**, type a property name.

    Note that there are reserved names for properties, unlike for other kinds of cube elements. For details, see "Names for Model Elements," earlier in this book.

4. Click **Property**.

5. Click **OK**.

6. Select the property in the Model Viewer.

7. In the Details Area, specify a value for **Property** or **Expression**. See "Defining the Source Values for a Dimension or Level" and "Details for Source Expressions," earlier in this book.

Alternatively, drag a class property from the Class Viewer and drop it onto a level in the Model Viewer. Then make changes if needed in the Details Area.

Also see "Other Common Options," earlier in this book.

9.1.1 Defining a Property for a List-Based Level

If the associated level is list-based, you must define the property as follows:

*   Specify a value for **Expression**.

*   In the expression, refer to the *%value* variable. This variable contains the value of the associated list element.

For example, consider the Allergy Severities level in the Patients cube. This level has a property that controls the sort order of the members of this level. This property (SeveritySort) is defined by the following expression:

```
%cube.GetSeveritySort(%value)
```

This executes the **GetSeveritySort()** method in the cube class and the allergy severity (a string) as a argument. The method returns an integer.

9.1.2 Specifying a Format String

In Studio, you can specify a format string, which enables you to specify the display format for the property. You can override this formatting in the Analyzer (or in manually written MDX queries). For details, see "<property>" in the first appendix.

9.2 Getting a Property Value at Runtime

In some cases, you can define a property so that its value is retrieved at runtime from the appropriate source table. The requirements are as follows:

*   For the property, select the **Get value at run time** option.

*   For the parent level (the level that contains the property), the source property or source expression of that level must evaluate to an ID.

    The system assumes that (at least for this level), the source data is normalized. That is, for the level, the data is in a different table and the source table contains a link to that table.

    This requirement means that, by default, the member names for this level are IDs, which is not generally suitable for your users. In this case, also configure a property of this level for use as display names for the members. This can be the same property or a different property.

    See "Using Property Values as the Member Names," in the previous chapter.

9.3 Changing the Order of Properties in a Level

To change the order of properties in a level:

- To move a property up in the level, click the up arrow in the row for that level.

- To move a property down in the level, click the down arrow in the row for that level.

The order of properties in a level can affect the sort order of the members of the level. See "Specifying the Sort Order for Members of a Data Level," in the previous chapter.

9.4 Specifying the Column Names in the Dimension Tables

When you compile a cube class, DeepSee generates a fact table class and related classes for the dimensions. When you build a cube, DeepSee populates these tables, which are described in "Details for the Fact and Dimension Tables." The properties for a level are stored in the corresponding dimension table.

By default, DeepSee generates the names of the columns in the fact table, but you can specify the column names to use instead. To do so, specify a value for the **Field name in level table** option for each property. Take care to use unique names.

Important: For **Field name in fact table**, be sure not to use an SQL reserved word. For a list of the SQL reserved words, see "Reserved Words" in the *Caché SQL Reference*. The name must start with either a letter or a percent sign (%). If the first character is %, the second character must be Z or z. For more details on restrictions, see "Class Members" in the appendix "Rules and Guidelines for Identifiers" in the *Caché Programming Orientation Guide*. Also, do not use fact or listing, in any combination of lowercase and uppercase characters.

9.5 Special Uses for Properties

By default, properties have no effect on the members of the level to which they belong. You can, however, use properties to modify the members in the following ways (which you can combine):

- You can use the property values as the member names, overriding the default member names defined by the level definition.

  To do this, use the **Use as member names** option and (optionally) **Get value at run time** options; see "Using Property Values as the Member Names," in the previous chapter.

  Note that if you select **Use as member names**, and if the source value is null, the system instead uses the appropriate null replacement string; this is either the null replacement string for the level (if defined) or for the cube (if defined). See "Specifying the Null Replacement String for a Level" and "Specifying Cube Options."

- You can use the property values to specify the default sort order of the members within the level.

  To do this, use the **Sort members by property value** option; see "Specifying the Sort Order for Members of a Data Level," in the previous chapter.

  The **Sort members by property value** option has no effect if the level uses a range expression.

- You can use the property values as tooltips for the members. This is useful when you want to use short member names, for reasons of space, but also need to provide a fuller description of each member. To use a property this way, specify the isDescription attribute as "true". See <property> in the appendix "Reference Information for Cube Classes."

Tip: Within a level, member names are not required to be unique, as noted in the section "Defining Member Keys and Names," earlier in this book. In a well-defined cube, however, each member of a given level does have a unique key. When a user creates a query in the Analyzer, DeepSee automatically uses the member keys instead of the names. The user can expand the list of members and separately drag and drop different members that have the same names. For convenience of the users, InterSystems suggests that you also add a level property whose value is the key. For such a property, simply base the property on the same source property or source expression that the level uses. The users can then display this property in order to determine the unique identifier that DeepSee uses for the member.

10

Defining Measures

This chapter describes how to define measures. It discusses the following topics:

- How to create a measure

- How to specify the type of measure

- How to specify the aggregation for a measure

- How to specify a searchable measure

- How to specify a format string

- How to specify a format string for a date measure

- How to change the order of measures in a cube

- How to specify column names in the fact table

- How to specify an additional filter to apply to detail listings

DeepSee automatically creates a measure whose default name is Count. To override this default name, specify the **Count measure caption** option for the cube; see "Specifying Cube Options," earlier in this book.

10.1 Adding a Measure

To add a measure, drag a class property from the Class Viewer and drop it onto the **Measures** label in the Model Viewer. Then make changes if needed in the Details Area.

Or do the following:

1. Click **Add Element**.

    The system displays a dialog box.

2. For **Enter New Item Name**, type a measure name.

    See "Names for Model Elements," earlier in this book.

3. Click **Measure**.

4. Click **OK**.

5. Select the measure in the Model Viewer.

6. Specify the following options, at a minimum:

- **Type** — Specifies the measure type, as described in "Specifying the Measure Type."

- **Property** or **Expression** — Specifies the source values. See "Defining the Source Values for a Dimension or Level" and "Details for Source Expressions," earlier in this book.

- **Aggregate** — Specifies how to aggregate the measure values, as described in "Specifying How to Aggregate a Measure."

10.2 Specifying the Measure Type

The **Type** option specifies the kind of data the measure expects in the source data, as well as the type used in the generated fact table, as follows:

Measure Type	Expected Type of Source Data	Type Used by Measure	Measure Details
number (this is the default)	Numeric data	%Double	Numeric value. Default aggregation is SUM.
integer	Numeric data (any fractional values are removed by truncation)	%Integer	Integer value. Default aggregation is SUM.
age	Date/time data in **$Horolog** format	%Integer	Age value in days. Can be used only with AVG (default), MIN, and MAX aggregations. Age measures are not generally recommended, unless you perform nightly rebuilds.
date	Date/time data in **$Horolog** format	%DeepSee.Datatype.dateTime	Date value (in **$Horolog** format, with seconds removed). Can be used only with AVG, MIN, and MAX (default) aggregations.
boolean	0 or 1	%Boolean	Boolean value that can be aggregated. Default aggregation is COUNT.
string	Any	%String	String values to be stored in the fact table; these are not indexed. Can be used only with COUNT. This measure cannot be dragged and dropped in the Analyzer.
text	Any	%Text	String values to be stored in the fact table; these are indexed. This measure cannot be dragged and dropped in the Analyzer.
iKnow[*]	Text values	%GlobalCharacterStream or %String, depending on the selected source	Text values to be processed and indexed using the iKnow Smart Indexing API. This measure cannot be dragged and dropped in the Analyzer.

[*]For information on the **iKnow** type, see the *Advanced DeepSee Modeling Guide*.

10.3 Specifying How to Aggregate a Measure

The **Aggregate** option specifies how to aggregate values for this measure, whenever combining multiple records. If you specify this, use one of the following values:

- SUM (the default) — Adds the values in the set.

- COUNT — Counts the records for which the source data has a non-null (and nonzero) value.

- `MAX` — Uses the largest value in the set.

- `MIN` — Uses the smallest value in the set.

- `AVG` — Calculates the average value for the set.

For a boolean or a string measure, select `COUNT`.

10.4 Specifying a Searchable Measure

You can specify that a measure is *searchable*; if so, you can filter records used in a pivot table by the value of that measure.

To specify a measure as searchable, select the **Searchable** check box in the Details Area.

Note: A searchable measure cannot include square brackets or commas (`[ ] ,`) in its name.
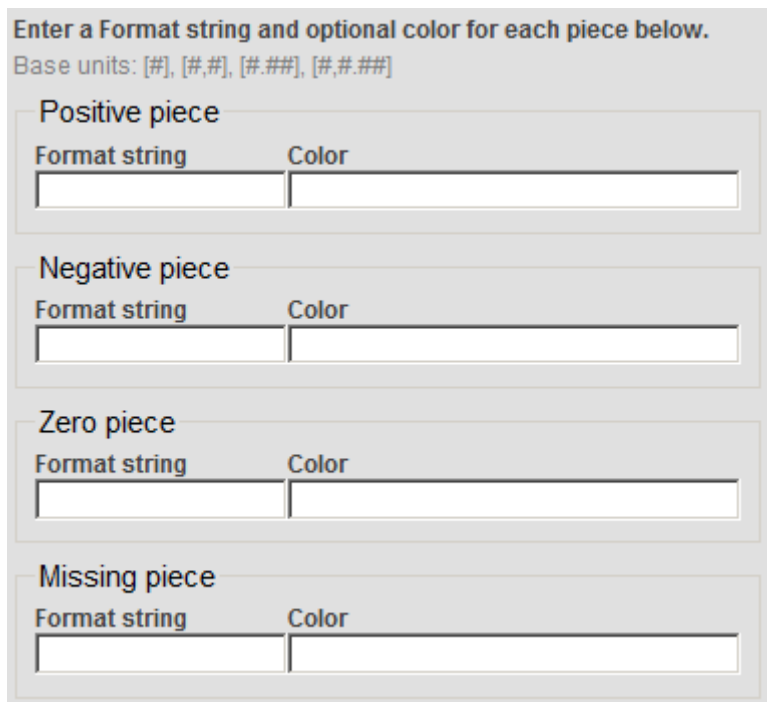
10.5 Specifying a Format String

Note: For date measures, see the next section.

The **Format string** option enables you to specify the display format for the data. You can override this formatting in the Analyzer (or in manually written MDX queries). To specify the formatting for a measure in the Architect, do the following while the measure is displayed:

1.
   Click the Find button 🔍.

   The system displays a dialog box that includes the following fields:

   Enter a Format string and optional color for each piece below.
   Base units: [#], [#,#], [#.##], [#,#.##]

   Positive piece
   Format string   Color

   Negative piece
   Format string   Color

   Zero piece
   Format string   Color

   Missing piece
   Format string   Color

Here:

- **Positive piece** specifies the format to use for positive values.

- **Negative piece** specifies the format to use for negative values.

- **Zero piece** specifies the format to use for zero.

- **Missing piece** specifies the format to use for missing values; this is not currently used.

In each of these, **Format string** specifies the numeric format, and **Color** specifies the color.

The details are different for date-type measures; see the next section.

2. Specify values as needed (see the details after these steps).

3. Click **OK**.

10.5.1 Format String Field

The **Format string** field is a string that includes one of the following base units:

Base Unit	Meaning	Example
#	Display the value without the thousands separator. Do not include any decimal places.	12345
#,#	Display the value with the thousands separator. Do not include any decimal places. This is the default display format for positive numbers.	12,345
#.##	Display the value without the thousands separator. Include two decimal places (or one decimal place for each pound sign after the period). Specify as many pound signs after the period as you need.	12345.67
#,#.##	Display the value with the thousands separator. Include two decimal places (or one decimal place for each pound sign after the period). Specify as many pound signs after the period as you need.	12,345.67
%time%	Display the value in the format hh:mm:ss, assuming that the value indicates the number of seconds. This is useful for measures that display time durations.	00:05:32

Note that Caché *displays* the thousands separator and the decimal separator as determined by the server locale (see "Using the Locale to Control the Names of Time Members," earlier in this book). The locale, however, *does not* affect the syntax shown in the first column of the preceding table.

You can include additional characters before or after the base unit.

- If you include a percent sign (%), the system displays the value as a percentage. That is, it multiplies the value by 100 and it displays the percent sign (%) in the position you specify.

- Any other characters are displayed as given, in the position you specify.

The following table shows some examples:

Example formatString	Logical Value	Display Value
formatString="#,#;(#,#);" Note that this corresponds to the default way in which numbers are displayed.	6608.9431	6,609
	−1,234	(1,234)
formatString="#,#.###;"	6608.9431	6,608.943
formatString="#%;"	6	600%
formatString="$#,#;($#,#);"	2195765	$2,195,765
	−3407228	($3,407,228)

10.5.2 Color Piece

For the *Color* field, specify either of the following:

- A CSS color name such as MediumBlue or SeaGreen. You can find these at https://www.w3.org/TR/css3-color/ and other locations on the Internet.

- A hex color code such as #FF0000 (which is red).

- An RGB value such as rgb(255,0,0) (which is red).

10.6 Specifying a Format String for a Date Measure

To specify the format string for a date measure, enter one of the following into the **Format string** field in the Details pane of the Architect:

Format string	How This Affects the Format of the Measure	Example formatting
%date%	Date uses the default date format for the current process.	
%date%^color Where *color* is a color as described in "Color Piece," in the previous section.	Date uses the default date format for the current process and also is displayed in the given color.	
^color	Date is displayed in the given color.	

10.7 Changing the Order of Measures in the Cube

To change the order of measures in the cube:

1. Click **Reorder**.

   The system displays a dialog box.

2. Click **Measures**.

3. Optionally click **Alphabetize** to alphabetize them.

   This affects the list immediately. You can then reorganize the list further if needed. Also, when you add measures, they are not automatically alphabetized.

4. Click the name of a measure and then click the up or down arrow as needed.

5. Repeat as needed for other measures.

6. Click **OK**.

The order of the measures in the cube affects how they are displayed in the Analyzer. It does not have any other effect. Some customers choose to alphabetize their measures for convenience; others put more-often used measures at the top of the list.

10.8 Specifying the Field Names in the Fact Table

When you compile a cube class, DeepSee generates a fact table class and some related classes. When you build a cube, DeepSee populates these tables, which are described in "Details for the Fact and Dimension Tables."

By default, DeepSee generates the names of the columns in the fact table, but you can specify the column names to use instead. To do so, specify a value for the **Field name in fact table** option for each measure. This option is not available for iKnow measures. Take care to use unique names.

Important: For **Field name in fact table**, be sure not to use an SQL reserved word. For a list of the SQL reserved words, see "Reserved Words" in the *Caché SQL Reference*. The name must start with either a letter or a percent sign (`%`). If the first character is `%`, the second character must be `Z` or `z`. For more details on restrictions, see "Class Members" in the appendix "Rules and Guidelines for Identifiers" in the *Caché Programming Orientation Guide*. Also, do not use `fact` or `listing`, in any combination of lowercase and uppercase characters.

10.9 Specifying Additional Filtering for Listings

By default, when a user displays a detail listing, DeepSee displays one row for each source record that is used in the current context (that is, the context in which the listing was requested). For a given measure, you can specify an additional filter for DeepSee to use when displaying the detail listing. For example, consider the Avg Test Score measure in the Patients sample. This measure is based on the TestScore property, which is null for some patients. You could redefine this measure to filter out those patients, when the user starts on the Avg Test Score measure and then displays a listing.

If you need such a filter, you include it as part of the measure definition. In most cases, the filter has the following form:

```
measure_value operator comparison_value
```

This filter is added to the detail listing query and removes any records that do not meet the filter criteria.

The other form of listing filter is MAX/MIN. If you use such a listing filter, the detail listing shows only the records that have the maximum (or minimum) value of the measure. The measure must use the same kind of aggregation as does the listing filter (if a listing filter is included).

To specify an additional filter for listings, for a specific measure:

1. Select the measure in the Model Viewer.

   If you intend to use **MAX** for the listing filter, select a measure that is defined with **Aggregate** as **MAX**.

   Similarly, if you intend to use **MIN**, select a measure that is defined with **Aggregate** as **MIN**.

2. In the section **Measure-Specific Listing Filter** (in the Details Area), specify the following values:

   - **Operator** — Select one of the following: **< <= > >= <> = MAX MIN**

   - **Value** — Specify the comparison value. Omit this option if **Operator** is **MAX** or **MIN**

When you use this option, the Architect automatically enables the **Searchable** check box, because this measure must be searchable.

10.9.1 Example

Suppose that we modify the Avg Test Score measure in the Patients cube, and we specify **Operator** as **<>** and **Value** as " ". That is, we want to filter out the patients that have null test scores. Then consider the following pivot table:

Diagnoses	Patient Count	Avg Age	Avg Test Score
None	833	32.84	73.95
asthma	74	32.72	74.12
CHD	28	70.57	76.52
diabetes	50	60.62	74.36
osteoporosis	30	81.50	71.74

If we click the Patient Count cell (or the Avg Age cell) in the CHD row and then display a detail listing, we see something like this:

#	PatientID	Age	Gender	Home City	Test Score
1	SUBJ_100446	44	M	Elm Heights	94
2	SUBJ_101284	56	F	Spruce	
3	SUBJ_100357	56	M	Centerville	93
4	SUBJ_100497	56	M	Pine	89
5	SUBJ_100597	58	F	Cedar Falls	
6	SUBJ_101281	58	F	Spruce	84
7	SUBJ_100733	58	M	Redwood	88
8	SUBJ 101265	59	F	Elm Heights	88

But if we click the Avg Test Score cell in the pivot table and then display a detail listing, we see fewer records, like this:

#	PatientID	Age	Gender	Home City	Test Score
1	SUBJ_100446	44	M	Elm Heights	94
2	SUBJ_100357	56	M	Centerville	93
3	SUBJ_100497	56	M	Pine	89
4	SUBJ_101281	58	F	Spruce	84
5	SUBJ_100733	58	M	Redwood	88
6	SUBJ 101265	59	F	Elm Heights	88

11

Defining Listings

This chapter describes how to define listings. It discusses the following topics:

- How to add a listing

- Options for a simple listing

- Options for a data connector listing

- Options for a custom listing

- How to define a map listing

In Studio, you can define formatting for a listing. See "<listing>," in the appendix "Reference Information for Cube Classes."

You can also define *individual* listing fields with which users can create custom listings in the Analyzer. See the next chapter.

Tip: You can also define listings outside of cube definitions (and without needing access to the Architect). See the chapter "Defining Listing Groups."

11.1 Adding a Listing

To add a listing:

1. Click **Add Element**.

   The system displays a dialog box.

2. For **Enter New Item Name**, type a listing name.

   See "Names for Model Elements," earlier in this book.

3. Click **Listing**.

4. Click **OK**.

5. Select the listing name in the Model Viewer, in the **Listings** section.

6. Optionally specify the following details:

   - **Display name** — Localizable name of the listing. If you do not specify this, the user interface instead displays the logical name.

- **Description** — Description of the listing.

- **Listing type** — Type of the listing. Select **table** (the default) or **map**.

  If you choose **map**, see the section "Defining a Map Listing," later in this chapter.

- **Resource** — Specify the Caché resource that secures the listing.

  For information on how this is used, see "Setting Up Security" in the *DeepSee Implementation Guide*.

7. Define the listing as described in the following sections.

Note: When you define listings, it is important to consider whether the cube itself is based on a data connector. If it is, then all listings for that cube must be based on data connectors.

　　　　If the cube is not based on a data connector, however, its listings can be based on data connectors *or* on the same source class used by the cube.

11.2 Defining a Simple Listing

A simple listing uses fields in the source table that is used by the cube. To define a simple listing, add a listing as described earlier in this chapter and specify the following options:

- **Field list** — Specifies a comma-separated list of fields in the source table to display. To specify this option, click the

  Search button 🔍 next to **Field list** to display a dialog box where you can select the fields. If you do so, the Architect displays the following:

To use this dialog box:

– If necessary, expand the items in the **Source Class** tree.

– To add a property, double-click the property name in the **Source Class** tree. The property is then added to the end of the list shown in **Field List**. Properties with a default caption are automatically assigned an alias when added. For more information on the CAPTION property, see Core Property Parameters in *Using Caché Objects*. Note that the value shown is the SqlFieldName for the property, which might be different from the property name.

– To move an item up or down, click it in **Field List** and then click the up or down arrows as needed.

– To edit an item, click it in **Field List**, make changes in **Edit Field:** and then click **Update**. For example, you can add an SQL alias. See the subsection "Additional Options."

– To delete an item, click it in **Field List** and then click the X button.

When you are done, click **OK**.

You can also type directly into **Field list**. For example:

```
PatientID,Age,Gender,HomeCity->Name AS "Home City",TestScore AS "Test Score"
```

Note that upon use of the **Listing — Field List** dialog box, any headers will be automatically wrapped with the $$$TEXT token. The cube's domain will be added using $$$TEXT, such as in the following example:

```
Product->Name AS "$$$TEXT["Product","HOLEFOODS"]"
```

You can override the cube's default domain by entering a different domain directly into **Field list**.

- **Order by** — Specifies a comma-separated list of fields in the source table by which to sort the listing (these do not need to be included in **Field list**). The overall sort is controlled by the first field in the list, the secondary sort is controlled by the second field, and so on.

    To specify this option, click the Search button ￼ next to **Field list** to display a dialog box where you can select the fields. This dialog box is a simpler version of the one for **Field list**.

    Or type directly into **Order By**. For example:

    ```
    Age,Gender
    ```

    After a field name, you can include the ASC or DESC keyword to sort in ascending or descending order, respectively.

- Ignore the **Data Connector** field and the **Custom SQL query** field.

11.2.1 Additional Options

Note the following points:

- You can use Caché arrow syntax to refer to a property in another table. See "Implicit Joins (Arrow Syntax)" in *Using Caché SQL*.

    ```
    PatientID,HomeCity,PrimaryCarePhysician->DoctorGroup
    ```

- You can include aliases.

    ```
    PatientID,Age,Gender,HomeCity->Name AS "Home City",TestScore AS "Test Score"
    ```

    Or:

    ```
    PatientID,Age,Gender,HomeCity->Name "Home City",TestScore "Test Score"
    ```

- You can localize an alias by providing using the special token $$$TEXT[]. For example:

    ```
    %ID,DateOfSale As "$$$TEXT["Date Of Sale"]"
    ```

- You can use standard SQL and Caché SQL functions, if you enclose the function name within parentheses so that it is not interpreted as a field name.

    ```
    (UCASE(PatientID)),%EXTERNAL(Gender)
    ```

- You can use more advanced SQL features if you use source.*field_name* rather than *field_name*.

    ```
    %ID,'$'||source.Sales AS Sales
    ```

    In this case, the Sales column displays the Sales field, preceded by a dollar sign ($).

Tip: By default, Caché uses the logical values for the fields. To instead use the display values, use the Caché SQL function %EXTERNAL.

11.3 Defining a Data Connector Listing

A data connector listing uses fields in a data connector. To define such a listing, add a listing as described earlier in this chapter and specify the following options:

- **Data Connector** — Select the data connector class to use.

- **Field list** — Specify the fields to include. These must be fields in the data connector. You cannot include aliases, SQL functions, or Caché arrow syntax.

  By default, the listing displays only a property named `%ID`, if that exists.

- **Order by** — Not used.

- Ignore the **Custom SQL query** field.

Note: If a cube is based on a data connector, all its listings must also be based on that same data connector.

11.4 Defining an SQL Custom Listing

An SQL custom listing can use fields from tables other than the source table used by the cube. (Note that it cannot use fields from a data connector.)

Note: DeepSee supports another kind of "custom" listing — a listing defined by users in the Analyzer. See "<listing-Field>" in the appendix "Reference Information for Cube Classes."

To define an SQL custom listing, add a listing as described earlier in this chapter and specify the following options:

- **Custom SQL query** — Select this.
- **Custom SQL** — Specify an SQL SELECT query as described in this section.

Before looking at the details of the **Custom SQL** query, it is necessary to understand how DeepSee creates listings. For any listing, the system creates a temporary listing table that contains the set of source ID values that correspond to the facts used in the current context (the context in which the user requests the listing). Your custom SQL query selects fields where the records match the source IDs in the temporary listing table. Internally, the overall query might be as follows:

```
SELECT source.Field1,source.Field2
FROM DeepSee_Study.Patient source,internal-listing-table-name list
WHERE source.%ID=list.sourceID AND list.queryKey='2144874459'
```

When you specify an SQL custom query, you specify tokens to replace some of these details. Specifically, for **Custom SQL**, you provide a query with the following basic form:

```
SELECT list of field names  FROM $$$SOURCE, othertable AS alias  WHERE $$$RESTRICT
AND otherrestriction
```

Where:

- The FROM clause specifies multiple tables to query. In this clause, the `$$$SOURCE` token establishes `source` as the alias for the source table of the cube, and it establishes `list` as the alias for the temporary listing table.

  Internally `$$$SOURCE` is replaced by something like the following: `DeepSee_Study.Patient source,internal-listing-table-name list`

  Also, *othertable AS alias* specifies another table to query, and an alias for this table (if wanted). You can specify additional tables as well.

- The WHERE clause specifies conditions that join the source table to the listing table and to your additional table (or tables).

The `$$$RESTRICT` token is replaced by a condition that joins the source table to the temporary listing table; internally this is replaced by something context-dependent like the following: `source.%ID=list.sourceID AND list.queryKey='2144874459'`

Also *otherrestriction* specifies another SQL restriction. In the syntax shown above, the restrictions are combined via AND. You could use OR instead or create a more complex WHERE clause if wanted.

- For *list of field names*, use a comma-separated list of field names, which can be fields in any of the tables listed in the FROM clause. For a field in the source table of the cube, use a reference of the form `source.`*fieldname*. You can include Caché arrow syntax as well.

- You can include aliases, and you can localize an alias by providing using the special token `$$$TEXT[]`. For example:

```
SELECT ID,UnitsSold As "$$$TEXT["Units Solds"]" FROM $$$SOURCE WHERE $$$RESTRICT
```

- You can also include the ORDER BY clause at the end of the query.

As a simple example (which uses only the source table of the cube):

```
SELECT source.PatientID,source.Age,source.HomeCity->Name
FROM $$$SOURCE
WHERE $$$RESTRICT
```

For another example (using data from other tables):

```
SELECT source.PatientID,FavoriteColor
FROM $$$SOURCE, DeepSee_Study.PatientDetails AS details
WHERE $$$RESTRICT AND source.PatientID=details.PatientID
```

11.5 Defining a Map Listing (Geo Listing)

By default, **Listing type** is **table**, and the listing displays a table of information. If you have suitable data, you can instead specify **Listing type** as **map**. In this case, the listing is a map with markers to indicate geographic locations contained in the listing data. For example, the map could highlight sales locations or customer locations. For example:
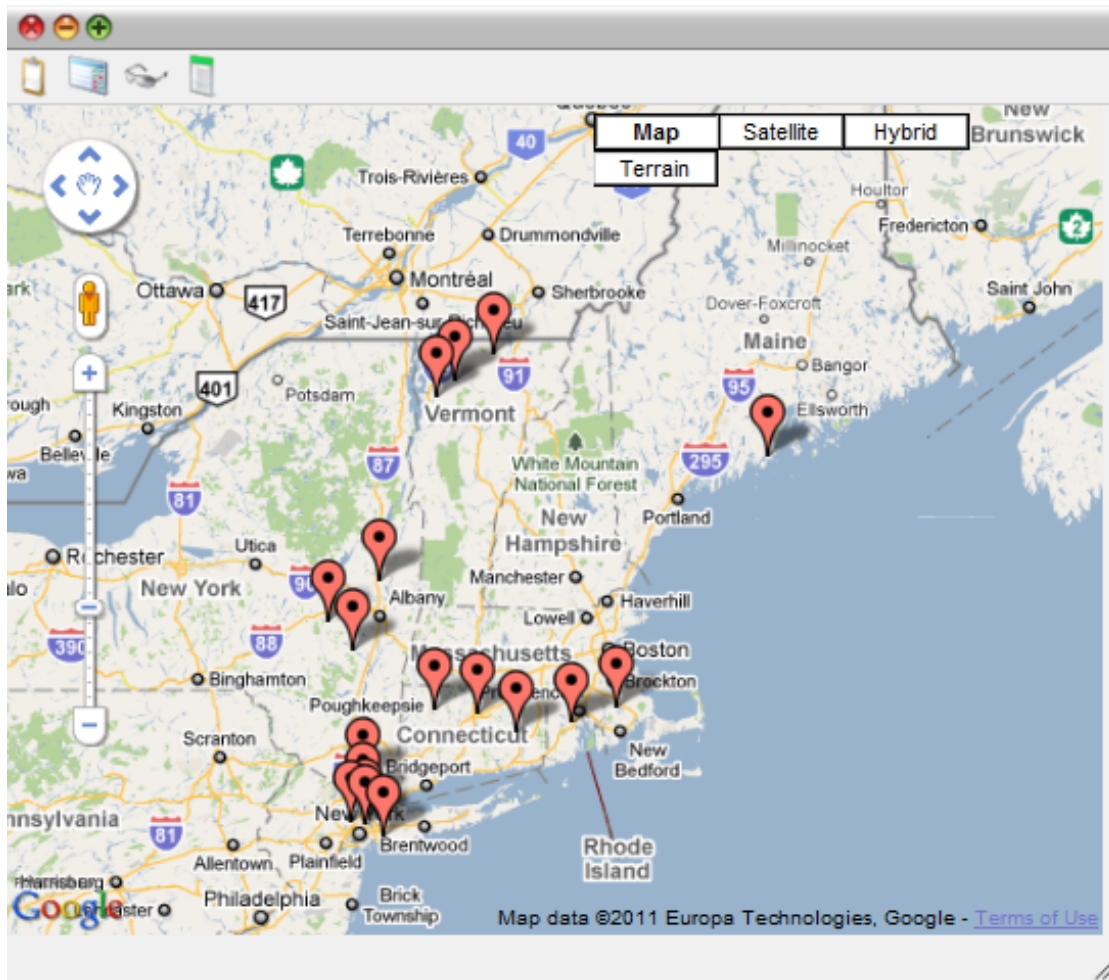
Important: A map listing uses the Google Maps API. Be sure that your usage of this API is consistent with the Terms of Use, which you can access via a link displayed in this listing, as shown in the previous picture.

Note that in order to use the Google Maps API, you must obtain an API key. See Specifying Basic Settings in the *DeepSee Implementation Guide* for more information.

To define such a listing, use the general instructions earlier in this chapter and do the following:

- Specify **Listing type** as **map**.

- Define the listing query so that it contains the fields `Latitude` and `Longitude` (case-sensitive). These fields should contain, respectively, the applicable latitude and longitude in decimal format (rather than degree/minute/second format).

    For example, **SQL Query** could be as follows:

    ```
    EventId, $$$IKSUMMARY as Summary, LocationCoordsLatitude As Latitude, LocationCoordsLongitude As
    Longitude
    ```

    This example is from the Aviation demo; see "Using Unstructured Data in Cubes (iKnow)" in the *Advanced DeepSee Modeling Guide*.

The listing query can also contain other fields. Any additional fields are displayed in a balloon when the user clicks a map position. For example:
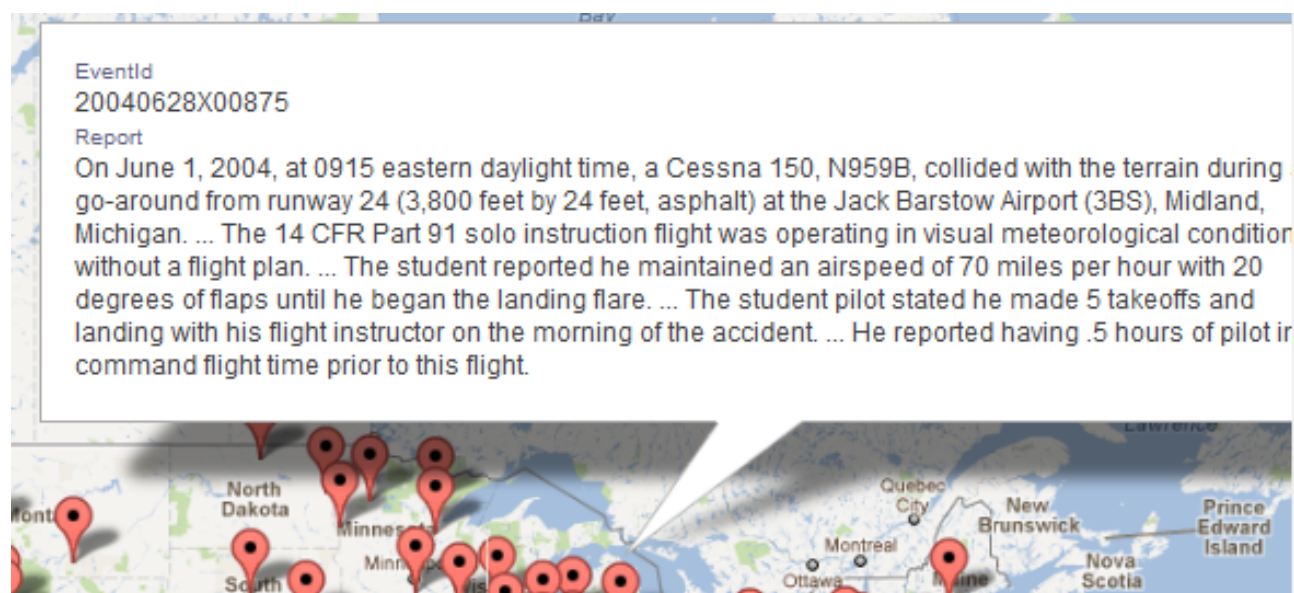
This example is from the Aviation demo; see "Using Unstructured Data in Cubes (iKnow)" in the *Advanced DeepSee Modeling Guide*.

12

Defining Listing Fields

This chapter describes how to add individual listing fields to a cube. With individual listing fields, users can create custom listings in the Analyzer. This chapter discusses the following topics:

- Summary of how users create custom listings

- How to create a listing field

Also see the chapter "Defining Listings."

12.1 About User-Defined Custom Listings

The purpose of individual listing fields is to enable users to define custom listings in the Analyzer. This process is described in "Creating a Custom Listing" in *Using the DeepSee Analyzer*, but the following steps provide a summary:

1. To start, the user displays the default listing.

2. The user selects **Detail Listings** in the left area of the Analyzer.

3. The user then expands the **Custom Listing** folder. For example:
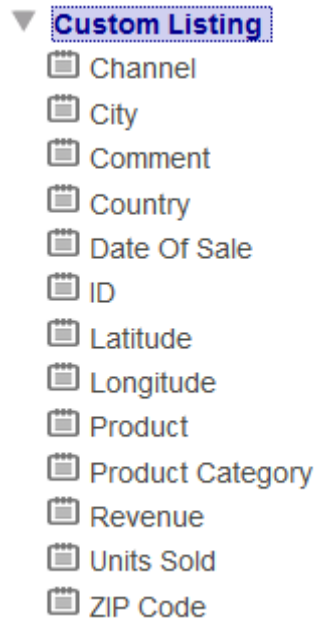
4. The user can then drag listing fields from this area to the **Custom Listing Fields** box. DeepSee then executes the listing query and displays the results.

12.2 Creating a Listing Field

To create a listing field:

1. Click **Add Element**.

    The system displays a dialog box.

2. For **Enter New Item Name**, type a listing name.

    See "Names for Model Elements," earlier in this book.

3. Click **ListingField**.

4. Click **OK**.

5. Select the listing name in the Model Viewer, in the **Listing Fields** section.

6. Optionally specify the following details:

    *   **Display name** — Localizable name of the listing field. If you do not specify this, the user interface instead displays the logical name.

    *   **Description** — Description of the listing field.

    *   **Field Expression** — SQL expression that refers to a field in the source table or (via arrow syntax, for example) to a field in a related table. The following example (from the Hole Foods Sales cube) uses Caché arrow syntax:

        ```
        (Product->Category)
        ```

        If you do not specify an alias, you must put parentheses around the expression as shown here.

        If you do specify an alias, the parentheses are not required. For example:

```
Product->Category Category
```

- **Resource** — Specify the Caché resource that secures the listing field.

    For information on how this is used, see "Setting Up Security" in the *DeepSee Implementation Guide*.

In **Field Expression**, you can use SQL functions, including Caché SQL functions such as %EXTERNAL. When a user creates a custom listing in the Analyzer, DeepSee creates an SQL SELECT statement that contains a comma-separated list of the **Field Expression** of the selected listing fields.

You can specify an SQL alias within **Field Expression**. However, if you do so, **Display name** is ignored and the field name cannot be localized.

You can also create a listing field via drag-and-drop action. To do so, drag a field from the left area and drop it onto the **Listing Fields** heading. Then select the new listing field and make any needed edits in the right-hand area.

You can also drag a different source field and drop it into **Field Expression**. If you do so, the value there is replaced with the new field.

13

Defining Calculated Members

This chapter describes how to add calculated members (including calculated measures) to a cube. It discusses the following topics:

- How to define a calculated measure
- MDX recipes for calculated measures
- How to define calculated members that are not measures
- MDX recipes for non-measure calculated members
- How to define additional filtering for listings for a calculated measure

Note: Users can create additional calculated measures, calculated members, and named sets within the Analyzer.

In MDX terminology, a calculated measure is simply another form of calculated member. This documentation uses the nonstandard phrase *calculated measure* for brevity.

13.1 Defining a Calculated Measure

To add a calculated measure:

1. Click **Add Element**.

   The system displays a dialog box.

2. For **Enter New Item Name**, type the name of the measure.

   See "Names for Model Elements," earlier in this book.

3. Click **Calculated Member (Measure)**.

4. Click **OK**.

5. Select the calculated measure in the Model Viewer (in the section **Calculated Members**).

6. Specify the MDX expression that defines the new member. To do so, you can do either of the following:

   - Type the expression directly into **Value Expression**.

   - Use the Expression Builder. This tool is intended to make it easy to obtain the correct MDX identifier for any part of the cube. To access this tool, click the magnifying glass next to **Expression**. The left area lists the contents of the cube, including all measures and levels. The right area displays the expression that you are creating. To add

an item to the expression, drag and drop it from the left area to the expression. The item is added to the end of the expression, and you might need to move it to a different part of the expression.

The following section has some example expressions. For additional samples, see the HoleFoods and Patients cubes.

The Architect displays this new measure with the other measures, in the **Measures** group.

Also see "Specifying Additional Filtering for Listings for a Calculated Measure," later in this chapter.

13.2 MDX Recipes for Calculated Measures

For a calculated measure, the MDX expression that you use as its definition should be a numeric expression. The section "Numeric Expressions" in *DeepSee MDX Reference* provides information on all the ways to create numeric expressions in MDX.

For an introduction to DeepSee MDX, see *Using MDX with DeepSee*.

This section discusses recipes for the following scenarios:

- Measures based on other measures

- Measures that use a pivot variable as a multiplier

- Measures that show percentages of aggregate values

- Measures that count distinct members

- Semi-additive measures

- Filtered measures, which are null in some cases

- Measures that use KPIs or plugins, which are discussed in the *Advanced DeepSee Modeling Guide*

For samples, see the HoleFoods and Patients cubes.

Note: Do not define a calculated measure that is based on *another* calculated measure that is based on a plugin. (For information on plugins, see the *Advanced DeepSee Modeling Guide*).

13.2.1 Measures Based on Other Measures

It is common to base one measure on other measures via an **Expression** like the following:

```
([MEASURES].[my measure 1] + [MEASURES].[my measure 2]) / [MEASURES].[my measure 3]
```

More formally, **Expression** is a numeric-valued MDX expression and can include the following elements:

- References to measures. The syntax is as follows:

  ```
  [MEASURES].[measure name]
  ```

  Or:

  ```
  MEASURES.[measure name]
  ```

  You can omit the square brackets around the measure name, if the measure name contains only alphanumeric characters, does not start with a number, and is not an MDX reserved word.

  The expression is not case-sensitive.

- Numeric literals. For example: `37`

- Percentage literals. For example: `10%`

  Note that there cannot be any space between the number and the percent sign.

- Pivot variables. See "Defining and Using Pivot Variables" in *Using the DeepSee Analyzer*.

  To refer to a pivot variable, use the syntax `$variable.`*variablename* where *variablename* is the logical variable name. This syntax is not case-sensitive; nor is the pivot variable name.

- Mathematical operators. DeepSee supports the standard mathematical operators: + (addition), – (subtraction), / (division), and * (multiplication). It also supports the standard unary operators: + (positive) and – (negative).

  You can also use parentheses to control precedence.

  For example: `MEASURES.[%COUNT] / 100`

- MDX functions that return numeric values. Many MDX functions return numeric values, including AVG, MAX, COUNT, and others. See the *DeepSee MDX Reference* for details. The function names are not case-sensitive.

Tip: The MDX function IIF is often useful in such expressions, for example, to prevent dividing by zero. It evaluates a condition and returns one of two values, depending on the condition.

13.2.2 Measure That Uses a Pivot Variable As a Multiplier

To define a measure that uses a pivot variable as a multiplier, use an **Expression** like the following:

```
measures.[measure A]*$variable.myQueryVariable
```

Where *myQueryVariable* is the logical name of a pivot variable. In this scenario, use a literal pivot variable that provides numeric values. See "Defining and Using Pivot Variables" in *Using the DeepSee Analyzer*. The syntax is not case-sensitive; nor is the pivot variable name.

13.2.3 Percentages of Aggregate Values

It is often necessary to calculate percentages of the total record count or percentages of other aggregate values. In such cases, you can use the %MDX function, which is an InterSystems extension. This function executes an MDX subquery, which should return a single value, and returns that value, which is unaffected by the context in which you execute the function. This means that you can calculate percentages with an **Expression** like the following:

```
100 * MEASURES.[measure A] / %MDX("SELECT MEASURES.[measure A] ON 0 FROM mycube")
```

The subquery `SELECT MEASURES.[measure A] ON 0 FROM mycube` selects the given measure from the cube and aggregates it across all records.

For example:

```
100 * MEASURES.[%COUNT]/%MDX("SELECT MEASURES.[%COUNT] ON 0 FROM patients")
```

In the case of the Count measure, you can use a simpler subquery:

```
100 * MEASURES.[%COUNT]/%MDX("SELECT FROM patients")
```

The following shows an example that uses the `Percent of All Patients` calculated measure, which is defined by the preceding **Expression**:

Diagnoses	Patient Count	Percent of All Patients
None	843	84.30
asthma	55	5.50
CHD	42	4.20
diabetes	46	4.60
osteoporosis	37	3.70

13.2.4 Distinct Member Count

In some cases, for a given cell, you want to count the number of distinct members of some particular level. For example, the `DocD` dimension includes the levels `Doctor Group` and `Doctor`. The calculated measure `Unique Doctor Count` uses the following **Expression**, which uses the `Doctor` level:

```
COUNT([docd].[h1].[doctor].MEMBERS,EXCLUDEEMPTY)
```

We can use this measure in a pivot table as follows:

Decade	Patient Count	Unique Doctor Count
None	206	143
1910s	78	63
1920s	215	156
1930s	515	269
1940s	709	321
1950s	1,034	348
1960s	1,414	381
1970s	1,530	387
1980s	1,347	378
1990s	1,421	381
2000s	1,399	382
2010s	132	90

13.2.5 Semi-Additive Measures

A *semi-additive measure* is a measure that is aggregated across most but not all dimensions. For example, customers' bank balances cannot be added across time, because a bank balance is a snapshot in time. To create such measures, you can use the %LAST function, an InterSystems extension to MDX.

Consider the following measures:

- `Balance` is based on the source property `CurrentBalance` and is aggregated by summing.

  You would avoid aggregating this measure over time, because it would give incorrect results; that is, you should use this measure only in pivot tables that include a time level for rows or columns.

- `Transactions` is based on the source property `TxCount` and is aggregated by summing.

You can define a calculated measure called `LastBalance` and use the following expression for **Expression**:

```
%LAST(Date.Day.Members,Measures.Balance)
```

The %LAST function returns the last non-missing value for a measure evaluated for each member of the given set. In this case, it finds the last day that has a value and returns that value.

13.2.6 Filtered Measures

A normal measure considers all records in the fact table for which the source value is not null. In some cases, you may want to define a filtered measure, which has the following behavior:

- The measure is null for certain records.

- For the other records, the measure has a value.

For a filtered measure, use an **Expression** like the following:

```
AGGREGATE([DIMD].[HIER].[LEVEL].[member name],[MEASURES].[my measure])
```

In this case, the AGGREGATE function aggregates the given value across all the records that belong to the given member.

For example, the Patients sample has the `Avg Test Score` measure, which is the average test score considering all patients who have a non-null value for the test. Suppose that in addition to the `Avg Test Score` measure, your customers would like to see another column that just shows the average test scores for patients with coronary heart disease (the CHD diagnosis). That is, the customers would like to have the measure `Avg Test Score - CHD`. In this case, you can create a calculated measure that has the following **Expression**:

```
AGGREGATE(diagd.h1.diagnoses.chd,MEASURES.[avg test score])
```

13.2.7 Measures That Use KPIs or Plugins

For any KPI or plugin, (all discussed in the *Advanced DeepSee Modeling Guide*), you can create a calculated measure that retrieves values from it. Then users can drag and drop this measure within the Analyzer. To create such a calculated measure, use an MDX expression of the following form for **Expression**:

```
%KPI(kpiname,propertyname,seriesname,"%CONTEXT")
```

Where *kpiname* is the name of the KPI or plugin, *propertyname* is the name of the property or column, and *seriesname* is the name of the series. You can omit *seriesname*; if you do, this function accesses the first series in the KPI or plugin.

For MDX-based KPIs and plugins, you can provide a parameter that carries context information. "%CONTEXT" is a special parameter that provides row, column, and filter context to the KPI or plugin; this information is passed to the base MDX query used by the KPI or plugin. The default for this parameter is "all", which uses the row, column, and filter context in combination. For additional options, see the %KPI function in the *DeepSee MDX Reference*.

For example (for a KPI or plugin with only 1 series):

```
%KPI("PluginDemo2","Count",,"%CONTEXT")
```

For another example, you can define a calculated measure that uses the sample median plugin (%DeepSee.PlugIn.Median). To do so, use the following **Expression**:

```
%KPI("%DeepSee.Median","MEDIAN",1,"%measure","Amount Sold","%CONTEXT")
```

13.3 Defining a Calculated Member (Non-Measure)

To add a calculated member that is not a measure:

1. Click **Add Element**.

The system displays a dialog box.

2. For **Enter New Item Name**, type the name of the member.

   See "Names for Model Elements," earlier in this book.

3. Click **Calculated Member (Dimension)**.

4. Click **OK**.

5. Select the calculated member in the Model Viewer (in the section **Calculated Members**).

6. For **Dimension**, type the name of the dimension to which this member belongs.

   You can specify any dimension, including an existing dimension that includes non-calculated members or a new dimension.

7. Specify the MDX expression that defines the new member. To do so, you can do either of the following:

   • Type the expression directly into **Value Expression**.

   • Use the Expression Builder. This tool is intended to make it easy to obtain the correct MDX identifier for any part of the cube. To access this tool, click the magnifying glass next to **Expression**. The left area lists the contents of the cube, including all measures and levels. The right area displays the expression that you are creating. To add an item to the expression, drag and drop it from the left area to the expression. The item is added to the end of the expression, and you might need to move it to a different part of the expression.

   The next section provides some recipes.

   For details and examples, see "WITH Clause" in the *DeepSee MDX Reference*.

   The Patients cube defines some samples; see the `ColorD` dimension, which includes two calculated members in addition to the standard members.

13.4 MDX Recipes for Non-Measure Calculated Members

This section provides recipes for non-measure calculated members for some common scenarios.

For general syntax, see "WITH Clause" in the *DeepSee MDX Reference*.

For an introduction to DeepSee MDX, see *Using MDX with DeepSee*.

This section discusses recipes for the following scenarios:

• Defining age members

• Aggregating multiple members together

• Aggregating ranges of dates

• Aggregating members defined by a term list

• Combining members of different levels, especially for filtering

For samples, see the Patients cube. The `ColorD` dimension, which includes two calculated members in addition to the standard members.

13.4.1 Defining Age Members

It is often useful to have members that group records by age. To define such members, use an existing time level and the special NOW member. For example, consider the `MonthSold` level in the HoleFoods sample. You could define a calculated member named `3 Months Ago` with the following **Expression**:

```
[dateofsale].[actual].[monthsold].[now-3]
```

You can define a set of age members, to create groups by age. For example, you could define the following members:

- **Dimension**: `AgeGroups`

    Member name: `1 to 2 year(s)`

    Expression: `%OR(DateOfSale.DaySold.[NOW-2y-1d]:[NOW-1y])`

- **Dimension**: `AgeGroups`

    Member name: `2 to 3 year(s)`

    Expression: `%OR(DateOfSale.DaySold.[NOW-3y-1d]:[NOW-2y])`

For more details and options, see "NOW Member for Date/Time Levels" in the *DeepSee MDX Reference*.

13.4.2 Aggregating Members

In many cases, it is useful to define a coarser grouping that combines multiple members. To do so, create a non-measure calculated member that has an **Expression** of the following form:

```
%OR({member_expression, member_expression,...})
```

For example:

```
%OR({[colord].[h1].[favorite color].[red],
[colord].[h1].[favorite color].[blue],
[colord].[h1].[favorite color].[yellow]})
```

In any case, each non-measure member refers to a set of records. When you create a member that uses the %OR function, you create a new member that refers to all the records that its component members use.

13.4.3 Aggregating Ranges of Dates

Another useful form uses a range of members aggregated by %OR:

```
%OR(member_expression_1:member_expression_n)
```

The expression `member_expression_1:member_expression_n` returns all members from `member_expression_1` to `member_expression_n`, inclusive. This form is particularly useful with time levels, because you can use it to express a range of dates in a compact form.

For time levels, you can also use the special NOW member. The following expression aggregates sales records from 90 days ago through today:

```
%OR(DateOfSale.DaySold.[NOW-90]:DateOfSale.DaySold.[NOW])
```

Or use the following equivalent form:

```
%OR(DateOfSale.DaySold.[NOW-90]:[NOW])
```

You can also use the %TIMERANGE function, which enables you to define a member that consists of all the members in an open-ended range. For example, the following expression defines a range that starts *after* the 2009 member:

```
%TIMERANGE(DateOfSale.YearSold.&[2009],,EXCLUSIVE)
```

The %TIMERANGE function is supported only with time levels and is not supported with relationships.

You can also use the PERIODSTODATE function to get a range of dates. For example, the following expression gets the range of days from the start of the current year to today and aggregates these days together:

```
%OR(PERIODSTODATE(DateOfSale.YearSold,DateOfSale.DaySold.[NOW]))
```

13.4.4 Defining an Aggregation of Members Defined by a Term List

Term lists provide a way to customize a DeepSee model without programming. A term list is a simple (but extendable) list of key and value pairs. (See "Defining Term Lists" in the *Advanced DeepSee Modeling Guide*.)

You can use term lists in the multiple ways; one is to build a set of members, typically for use in a filter. In this case, you use the %TERMLIST function and the %OR function; create a non-measure calculated member that has a **Expression** of the following form:

```
%OR(%TERMLIST(term_list_name))
```

Where *term_list_name* is a string that evaluates to the name of a term list.

For example:

```
%OR(%TERMLIST("My Term List"))
```

This expression refers to all records that belong to any of the members indicated by the term list (recall that %OR combines the members into a single member).

The %TERMLIST function has an optional second argument; if you specify "EXCLUDE" for this argument, the function returns the set of all members of the level that are not in the term list.

13.4.5 Defining a Member for Filtering on Multiple Dimensions

Member-based filters are so useful that it is worthwhile to create members whose sole purpose is for use in filters. Suppose that you need a filter like the following (which does not show literal syntax):

```
Status = "discharged" and ERvisit = "yes" and PatientClass="infant"
```

Also suppose that you need to use this filter in many places.

Rather than defining the filter expression repeatedly, you could define and use a calculated member. For this calculated member, specify **Expression** as follows:

```
%OR({member_expression,member_expression,...})
```

For example:

```
%OR({[BIRTHD].[H1].[YEAR].[NOW],[ALLERSEVD].[H1].[ALLERGY SEVERITIES].[003 LIFE-THREATENING]}
```

The expression ([BIRTHD].[H1].[YEAR].[NOW],[ALLERSEVD].[H1].[ALLERGY SEVERITIES].[003 LIFE-THREATENING]) is a tuple expression, which is the intersection of the member [BIRTHD].[H1].[YEAR].[NOW] and the member [ALLERSEVD].[H1].[ALLERGY SEVERITIES].[003 LIFE-THREATENING] — that is, all patients who were born in the current year and who have a life-threatening allergy.

Or more generally, use an expression of the following form:

```
%OR({set_expression})
```

13.5 Specifying Additional Filtering for Listings for a Calculated Measure

By default, when a user displays a detail listing, DeepSee displays one row for each source record that is used in the current context (that is, the context in which the listing was requested). For a given calculated measure, you can specify an additional filter for DeepSee to use when displaying the detail listing.

To specify an additional filter for listings, for a specific calculated measure:

1. Select the calculated measure in the Model Viewer.

2. In Details Area, specify **Listing Filter** as an MDX filter expression. To create the expression, type directly into this field or click the magnifying glass and create the expression via the editor.

   For information on and examples of MDX filter expressions, see "Filtering a Subject Area," earlier in this book.

Regular measures can also include an additional filter for use when a listing is displayed. The syntax is different but the net effect is the same. For details and an example, see "Specifying Additional Filtering for Listings," earlier in this book.

14

Defining a Named Set

To add a named set:

1. Click **Add Element**.

   The system displays a dialog box.

2. For **Enter New Item Name**, type the name of the set.

   See "Names for Model Elements," earlier in this book.

3. Click **Named Set**.

4. Click **OK**.

5. Select the named set in the Model Viewer (in the section **Named Sets**).

6. For **Set expression**, specify an MDX set expression.

   For details, see the reference section "Set Expressions" in the *DeepSee MDX Reference*.

   For samples, see the HoleFoods and Patients cubes.

15

Defining Subject Areas

This chapter describes how to define subject areas. It includes the following topics:

- Introduction to the parts of the Architect when displaying a subject area

- How to define a new subject area

- How to filter a subject area

- How to specify other options for a subject area

- How to add items to a subject area

- How to add overrides for measures

- How to add overrides for dimensions, hierarchies, and levels

- How to redefine an existing listing or add a new listing

- How to compile a subject area

Note: In the current release, you cannot hide relationships in a subject area. To hide, add, or override named sets and calculated members (including calculated measures), use Studio. See "Reference Information for Subject Area Classes."

Also see "Defining Compound Cubes" in the *Advanced DeepSee Modeling Guide*.

15.1 Introduction to Subject Areas in the Architect

This section describes what the Architect displays when you are viewing a subject area rather than a cube.

1. Click **Open**, click **Subject Areas**, click **AsthmaPatients**, and then click **OK**.

    Now the system displays the page similar to what you see for a cube, except that it does not have the Class Viewer on the left:
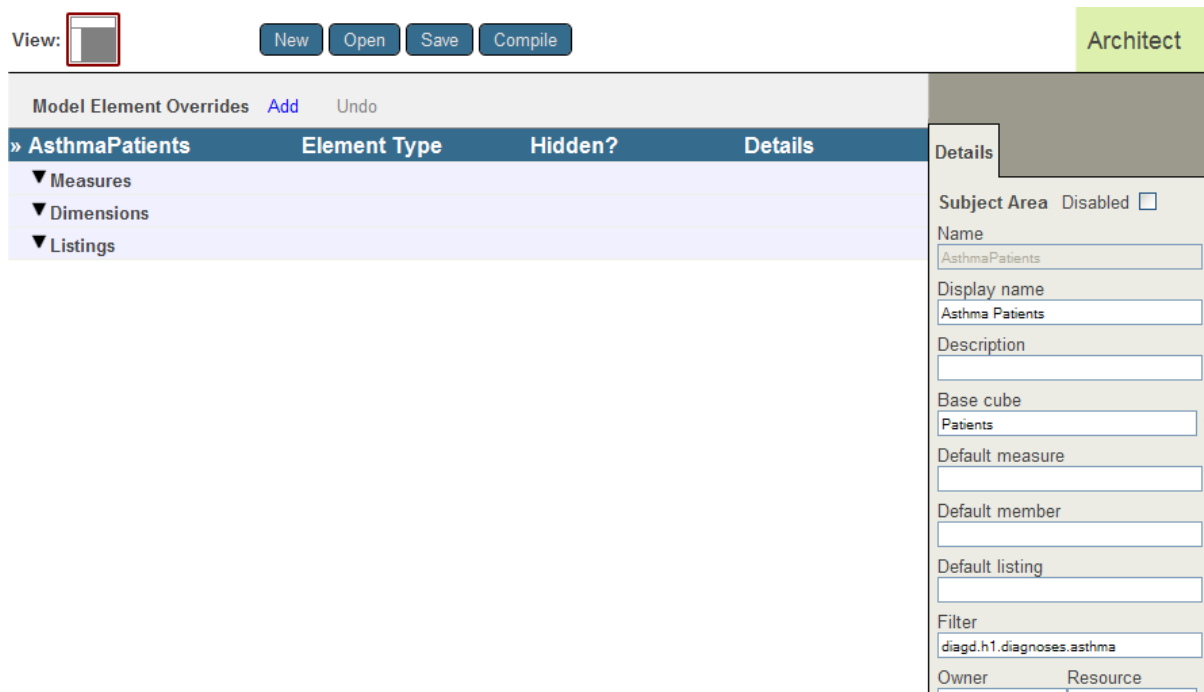
15.1.1 Model Viewer

The left area is the *Model Viewer*, and it shows the overrides defined in this subject area. For example:

This subject area does not define any overrides or any listings. (Many subject areas include only a filter.)

Here you can add overrides, select existing overrides for editing, and delete overrides.

You can also define new listings.

15.1.2 Details Area

The right area is the *Details Area*, and it shows details for the element that is currently selected in Model Viewer (if any), or for the subject area (if nothing is selected).

For example:

15.2 Defining a Subject Area

To define a subject area:

1. In the Architect, click **New**.

2. Click **Subject Area**.

3. Enter the following information at a minimum:

    - **Subject Area Name** — Name of the subject area to use as the default caption and to use in queries.

    - **Class name for the Subject Area** — Complete package and class name for the subject area class.

    - **Base Cube** — Logical name of the cube on which to base this subject area.

        You can either type the cube name or click **Browse** and select the cube.

        You can also use a comma-separated list of cubes; see "Defining Compound Cubes" in the *Advanced DeepSee Modeling Guide*.

    The other options are discussed later in this chapter.

    Apart from the class name, you can edit all options after creating the subject area.

4. Click **OK**.

5. Click the subject definition in the Architect to select it. Then edit the **Depends On** option in the Details Area. For the value, specify the full package and class name of the cube class on which this subject area is based.

    The subject area class should always be compiled after the cube class. The **Depends On** setting helps control this.

6. Optionally save the subject area. To do so:

    a. Click **Save**.

    b. Click **OK**.

    The system creates the class.

Or manually create the class as described in the appendix "Reference Information for Subject Area Classes."

15.3 Filtering a Subject Area

In most cases, you use subject areas to control access to data. In these cases, you specify the filter for the subject area. You can do this in either or both of the following ways:

- You can specify a hardcoded filter. To do so, specify a value for the **Filter** option in the Architect. For this option, specify an MDX filter expression to use as the filter for this subject area.

  This section discusses how to create hardcoded filters.

- You can specify a filter whose value is determined at runtime. To do so, customize the **%OnGetFilterSpec()** method of the subject area class. This method has access to the hardcoded filter, if any, and can modify that filter.

  For information, see "Filtering a Cube or Subject Area Dynamically" in the *Advanced DeepSee Modeling Guide*.

Note: The following subsections apply to MDX filters in general, not just to filters of subject areas.

15.3.1 Building a Filter in the Analyzer and Copying It into Your Model

If you are not familiar with MDX, the easiest way to specify a filter expression is as follows:

1. Create a filter by drag and drop as described in *Using the DeepSee Analyzer*.

2. Click the Query Text button .

   The Analyzer displays a dialog box with text like this:

   ```
   SELECT FROM [PATIENTS] WHERE [AGED].[H1].[AGE GROUP].&[0 TO 29]
   ```

3. Ignore all text before `WHERE`, which might be considerably more complicated depending on what you have done in the Analyzer. Copy all the text that follows `WHERE` into the system clipboard or other temporary location.

4. Paste the copied text into the **Filter** option for the subject area.

   For example:

   ```
   [AGED].[H1].[AGE GROUP].&[0 TO 29]
   ```

15.3.2 Writing Filter Expressions

This section discusses common forms of filter expressions.

Also the *DeepSee MDX Reference*.

15.3.2.1 Simple Filter

A simple filter refers to a single member:

```
[AGED].[H1].[AGE GROUP].&[0 TO 29]
```

This filter accesses only patients in the age range 0 to 29.

The preceding expression is an MDX member expression. Note the following details:

- `[AGED]` is a dimension in this sample.

- `[H1]` is the hierarchy in this dimension.

- [AGE GROUP] is a level in this hierarchy.

- &[0 TO 29] is the key for a member of that level.

  The preceding member expression refers to the &[0 TO 29] member of the [AGE GROUP] level of the [H1] hierarchy of the [AGED] dimension.

Also note the following possible timesaving variations:

- You can specify any of these identifiers in upper case, lower case, or mixed case at your convenience. MDX is not case-sensitive.

- You can omit the square brackets for any identifier that does not include space characters.

- You can omit the hierarchy and level names if the resulting expression is not ambiguous.

  You must always include the dimension name.

- You can use the member name instead of the member key. In most cases, this means that you can simply omit the ampersand (&) from the member identifier.

15.3.2.2 Filter with Set of Members

In other cases, you might need multiple members. For example:

```
{[aged].[h1].[age group].[0 to 29],[aged].[h1].[age group].[30 to 59]}
```

The subject area is filtered to include all records of the fact table that belong to *any* of the elements of the set. This subject area sees patients in the age group 0 to 29 and patients in the age group 30 to 59 — that is, it sees all patients under 60 years of age.

Important: Notice that the list is enclosed by braces; this indicates that the list is an MDX *set*. A set is a union of elements.

15.3.2.3 Filter with a Tuple

In some cases, you might need to find only records that meet multiple simultaneous criteria. In this case, you use a tuple expression. For example:

```
([aged].[h1].[age group].[60+],[diagd].[h1].[diagnoses].[diabetes])
```

This subject area sees patients who are 60 or older and who also have diabetes.

Important: Notice that the list is enclosed by parentheses; this indicates that the list is an MDX *tuple*. A tuple is an intersection of elements.

15.3.2.4 Filter with Multiple Tuples

You can list multiple tuple expressions within a set. For example:

```
{([aged].[h1].[age group].[60+],[diagd].[h1].[diagnoses].[diabetes]),
([colord.].[h1].[color].[red],[allergd].[h1].[allergy].[soy])}
```

This subject area sees all patients who meet at least one of the following criteria:

- They are at least 60 years old and who also have diabetes.

- They are allergic to soy and their favorite color is red.

15.4 Specifying Other Subject Area Options

In addition to the previously described options, you can specify the following additional options for a subject area:

- **Disabled** — (Optional) If you select this check box, the override is disabled. When you recompile the subject area, this override is ignored and DeepSee uses the definition given in the cube.

- **Display name** — Localizable name of the subject area.

- **Description** — (Optional) Comments to add to the subject area class definition. Each line is saved as a separate comment line at the start of the class definition.

- **Default member** — (Optional) Default member to use when a query skips an axis. Specify an MDX expression that returns a member that is accessible in this subject area. For information on member expressions, see the *DeepSee MDX Reference*. If this is not specified, the system uses the default member as defined in the cube.

- **Default listing** — (Optional) Logical name of the default listing for this subject area. This listing must be defined in the cube or in the subject area. If this is not specified, the system uses the default listing as defined in the cube.

- **Owner** — (Optional) Specify the owner of the cube. Specify a Caché username.

- **Count measure caption** — (Optional) Specify an alternative name for the Count measure.

- **Resource** — (Optional) Specify the Caché resource that secures the subject area.

  For information on how this is used, see "Setting Up Security" in the *DeepSee Implementation Guide*.

- **Caption**— (Optional) Specify the caption to display in the Analyzer and other utilities when working with this cube.

- **Domain**— (Optional) Specify the name of the domain to contain the localized strings of this subject area. You might find it convenient to use a single domain for all your cubes and subject areas; in other cases, it might be appropriate to have a separate domain for each cube and subject area. See the chapter "Performing Localization" in the *DeepSee Implementation Guide*.

15.5 Adding Items to a Subject Area

To add an override or a listing to a subject area, use the following general procedure:

1. Click **Add**.

   The system displays a dialog box on which you can choose an item from the cube:

If the subject area already defines an override for an item, this dialog box displays a check mark for that item.

2. Click **Measure**, **Dimension**, or **Listing**.

    With **Measure**, you can override any measure defined in the cube.

    With **Dimension**, you can override any dimension, hierarchy, or level defined in the cube.

    With **Listing**, you can override any listing defined in the cube. You can also add listings.

3. Click the item or items to override and then click **OK**.

    The system adds the items, displays them in the Model Viewer, and shows their details in the Details Area.

4. Edit the details in the Details Area.

15.6 Defining an Override for a Measure

To define an override for one or more measures:

1. Optionally click a measure name in the Model Viewer, to indicate where the new override is to be added.

    If you do so, the new override is added before the measure you clicked.

2. Click **Add**.

3. Click **Measure**.

4. Click the measure or measures to override and then click **OK**.

    The system adds the measures and displays them in the Model Viewer.

5. To define the override for a measure, click the measure name in Model Viewer and edit the following options in the Details Area. These are all optional.

    *   **Hidden** — Select this to hide this measure in this subject area. Or clear it to use this measure in this subject area.

    *   **Display name** — Specify a new display name to replace the one defined in the cube.

    *   **Description** — Specify a new description to replace the one defined in the cube.

- **Format string** — Specify a new format string to replace the one defined in the cube. See "Specifying a Format String," earlier in this book.
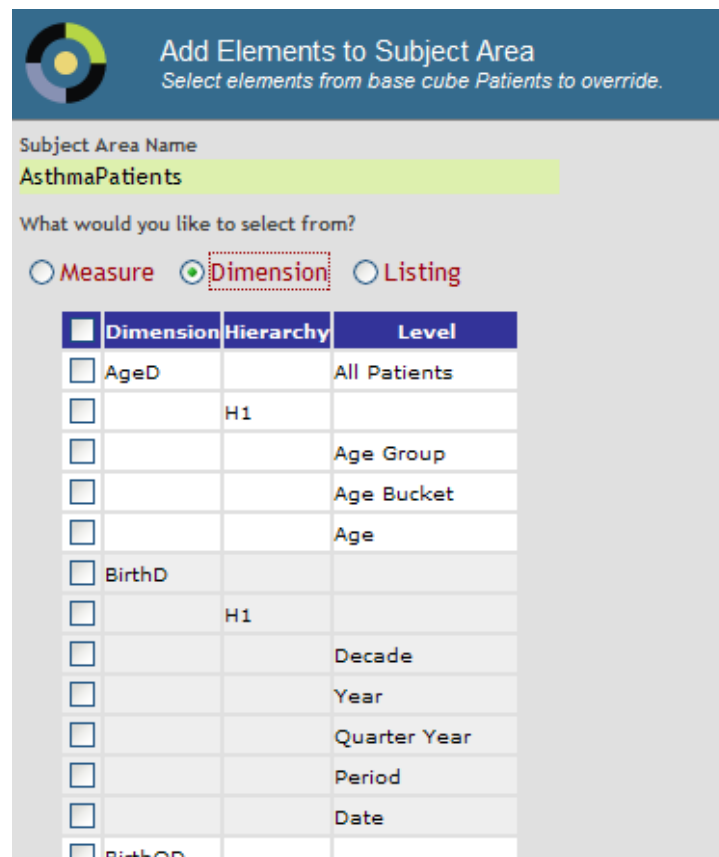
Note: You cannot define overrides for calculated measures this way. A calculated measure is actually a calculated member; to define an override for it, you must use Studio.

15.7 Defining an Override for a Dimension, Hierarchy, or Level

To define an override for one or more dimensions, hierarchies, or levels:

1. Click **Add**.

2. Click **Dimension**.

   The system displays the dimensions, hierarchies, and levels of the cube as follows:

3. Click the item or items to override and then click **OK**.

   The system adds the items and displays them in the Model Viewer.

   You can click any combination of dimensions, hierarchies, and levels. If you click a hierarchy but not its containing dimension, the system always adds the dimension as well (because the dimension is part of the hierarchy definition). Similarly, if you click a level but not its containing hierarchy or dimension, the system always adds the hierarchy and dimension as well.

4. To define the override for an item, click the item name in Model Viewer and edit the following options in the Details Area. These are all optional.

   - **Hidden** — Select this to hide this item in this subject area. Or clear it to use this item in this subject area.

   - **Display name** — Specify a new display name to replace the one defined in the cube.

   - **Description** — Specify a new description to replace the one defined in the cube.

   - **Sort** (Levels only) — Specify a different sort order for the members of this level.

      Note: If you change the sort attribute for a level in a subject area, that change is not in effect until you rebuild the cube.

15.8 Redefining a Listing or Adding a New Listing

To redefine a listing or define a new listing for the subject area:

1. Optionally click a listing name in the Model Viewer, to indicate where the new override is to be added.

   If you do so, the new override is added before the listing you clicked.

2. Click **Add**.

3. Click **Listing**.

   The system displays the listings of the cube and of the subject area as follows:

4. Optionally click the listing or listings to redefine.

5. Optionally type the name of a new listing in the box at the bottom.

6. Click **OK**.

   The system adds the listings (with no definitions) and displays them in the Model Viewer.

7. To define a listing, click the listing name in Model Viewer and edit the options in the Details Area.

    For details, see the chapter "Defining Listings."

15.9 Compiling a Subject Area

To compile a subject area class in the Architect:

1. Click **Compile**.

    The system displays a dialog box.

2. Click **Compile**.

    DeepSee starts to compile the class and displays progress as it does so.

3. Click **OK**.

Or open the subject area class in Studio and compile it in the same way that you compile other classes.

Tip: Before you compile a subject area class, you must compile the cube class on which it depends. It is useful, but not required, to use Studio to edit the subject area class and add the DependsOn keyword to force the classes to be compiled in the correct order. See "Requirements for a Subject Area Class" in the appendix "Reference Information for Subject Area Classes."

16

Defining Listing Groups

This chapter describes how to use the Listing Group Manager, which enables you to define listings that are not contained in any cube definition. It discusses the following topics:

- Introduction to listing groups
- How to access the Listing Group Manager
- How to add a new listing group
- How to display an existing listing group
- How to add a listing to a listing group
- How to remove a listing from a listing group
- How to compile a listing group
- How to delete a listing group

16.1 Introduction to Listing Groups

A *listing group* is a special kind of class that defines multiple listings, which are supplementary in the sense that they are defined outside of the cube definition. These can be any kind of listing supported by DeepSee. You can give users access to the Listing Group Manager so that they can define additional listings without modifying the cube definitions.

A listing group is meant to be a convenient container; it also specifies the following pieces of information that apply to all listings in the group:

- The cube or cubes in which the listings can be used.
- The resource that secures each listing (by default).

A listing group cannot override listings defined elsewhere.

A listing group is either disabled or enabled. If a listing group is disabled, its listings are not available to any cubes. If a listing group is enabled, and if it has been compiled, its listings are available to the specified cubes.

Important: The listings in a listing group are not available until the listing group has been compiled.

The listing group can have an associated resource, so that only users with access to that resource can edit the listing group.

Note that the Analyzer and the Dashboard Designer do not display the listing groups themselves, but rather simply display all available listings. End users do not have any information about where or how any given listing is defined.

16.2 Accessing the Listing Group Manager

The Listing Group Manager enables you to define listing groups and the listings in them. To access the Listing Group Manager:

1. Click the InterSystems Launcher and then click **Management Portal**.

   Depending on your security, you may be prompted to log in with a Caché username and password.

2. Switch to the appropriate namespace as follows:

   a. Click **Switch**.

   b. Click the namespace.

   c. Click **OK**.

3. Click **DeepSee**, click **Tools**, and then click **Listing Group Manager**.

   If there is a listing group in this namespace, the Listing Group Manager displays the last listing group that you looked at.

   If there is no listing group in this namespace, the Listing Group Manager prompts you to create one. Enter the following details and then press **OK**:

   • **Listing Group Name** — Logical name of the listing group. You can edit this later.

   • **Listing Group Class Name** — Class name of the listing group, including the package.

   • **Listing Group Description** — Optional description of the listing group. You can edit this later.

   Or click the X in the upper right of the dialog box.

The Listing Group Manager displays a listing group as shown in the following picture:

The left area displays the logical name of the listing group (Sample Listing Group in this example), followed by the names of the listings in this listing group. You can collapse and expand this part of the display.

The right area displays details for the item that you have selected on the left, either the listing group or one of the listings defined in it.

16.3 Adding a Listing Group

In the Listing Group Manager, to add a listing group:

1. Click **New**.

   The system then displays a dialog box.

2. Enter the following details:

   • **Listing Group Name** — Logical name of the listing group. You can edit this later.

   • **Listing Group Class Name** — Class name of the listing group, including the package.

   • **Listing Group Description** — Optional description of the listing group. You can edit this later.

3. Press **OK**.

The Listing Group Manager then creates and saves the class and displays the new listing group. The left area now displays something like the following:
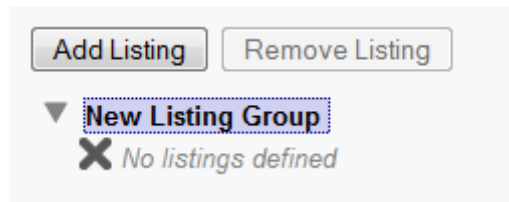
Now you can modify details of the listing group (which affect all the listings it contains), as described in the following subsection. Or you can add listings, as described in the later in this chapter.

16.3.1 Modifying Details of the Listing Group

In the Listing Group Manager, when you select the listing group in the left area (rather than selecting any listing contained in it), the right area displays the following details, which you can edit:

- **Disabled** — Optionally click this to disable the listing group. If you do so, the listings in this listing group are not available to any cubes.

- **Listing Group Name** — Specify the logical name of the listing group.

- **Listing Group Display Name** — Optionally specify the display name of the listing group.

- **Listing Group Target Cubes** — Select the cubes and subject areas that should be able to use the listings defined in this listing group. Notes:

  - If you select a cube, the listings will be available in that cube and in any subject areas based on that cube.

  - If you select a subject area but do not select the cube on which the subject area is based, then the listings are available *only* in that subject area.

  The listings are available only if the listing group is enabled and compiled.

- **Listing Group Resource** — Optionally select the resource that secures this listing group. Only users with access to this resource can edit this listing group.

  For information on how resources are used, see "Setting Up Security" in the *DeepSee Implementation Guide*.

- **Default Resource for Listings** — Optionally select the resource that, by default, secures each listing in this listing group.

- **Group Description** — Optionally specify the description of the listing group.

After making changes, click either **Save** or **Save As**.

If you click **Save As**, the system prompts you for a new logical name, a new class name, and an optional new description.

To make the changes available to users, be sure to also compile the listing group.

16.4 Displaying a Listing Group

In the Listing Group Manager, to display a listing group:

1. Click **Open**.

2. Click the name of the listing group.

3. Click **OK**.

Now you can modify details of the listing group (which affect all the listings it contains), as described in the previous section. Or you can add, modify, or remove listings, as described later in this chapter.

16.5 Adding a Listing to a Listing Group

In the Listing Group Manager, to add a listing to a listing group:

1. Display the listing group.

2. Click **Add Listing**.

    When you do, the system adds a new listing (with no details) and automatically saves the listing group at this point. The left area displays the name of the new listing; it has a name such as New Listing or New Listing1 and so on.

3. Click the name of the newly added listing.

4. Modify the name and other details shown in the right area. For information on these details, see the chapter "Defining Listings."

    Note that you cannot specify a listing name that already exists in the target cube or cubes; a listing group cannot override listings defined elsewhere.

5. Click **Save** to save the listing group. Or click **Save As** to create a new listing group.

6. To make the changes available to users, be sure to also compile the listing group.

16.6 Modifying a Listing

In the Listing Group Manager, to modify a listing:

1. Display the listing group that contains the listing.

2. In the left area, click the name of the listing.

3. Modify the name and other details shown in the right area. For information on these details, see the chapter "Defining Listings."

4. Click **Save** to save the listing group. Or click **Save As** to save the listing group with a new name.

5. To make the changes available to users, be sure to also compile the listing group.

16.7 Removing a Listing from a Listing Group

In the Listing Group Manager, to remove a listing:

1. Display the listing group that contains the listing.

2. In the left area, click the name of the listing.

3. Click **Remove Listing** and then click **OK**.

4. To make the changes available to users, be sure to also compile the listing group.

16.8 Compiling a Listing Group

In the Listing Group Manager, to compile a listing group:

1. Display the listing group.

2. If **Disabled** is cleared, make sure that **Listing Group Target Cubes** specifies at least one cube.

3. Click **Compile**.

The system then saves the class (if needed), compiles the class, and displays a message to indicate that it compiled the class.

After a listing group is compiled, its listings are available to the target cubes and subject areas; for details; see "Modifying Details of the Listing Group."

16.8.1 Possible Compile Errors

The Listing Group Manager issues a compilation error in any of the following situations:

- If there is no target cube specified. In this case, make sure that **Listing Group Target Cubes** specifies at least one cube and then recompile.

- If this listing group defines listings that have the same names as existing listings in the target cubes or subject areas. In that case, modify the names of the listings in the listing group, and then recompile. (Or disable the applicable listings within the listing group and then recompile.)

- If a target cube or subject area specifies `disableListingGroups="true"`. The compilation error includes the text `Target cube does not accept listing groups:` . In this case, remove that target cube, leaving any others, and recompile. Or modify the cube or subject area so that it can accept listing groups and recompile the cube or subject area.

  For information on `disableListingGroups`, see "Reference Information for Cube Classes" and "Reference Information for Subject Area Classes." You cannot specify this option via the Architect.

16.9 Deleting a Listing Group

In the Listing Group Manager, to delete a listing group:

1. Display the listing group.

2. Click **Delete**.

3. Click **OK** to confirm this action.

The system then deletes the class and displays a message to indicate that it did so.

The listings in this listing group are no longer available.

A

Reference Information for Cube Classes

The Architect creates and modifies cube classes, which you can also create and edit directly in Studio. This appendix provides reference information on these classes. It discusses the following topics:

- Basic requirements for a cube class

- Common attributes in a cube

- <cube>

- <measure>

- <dimension>

- <hierarchy>

- <member>

- <property>

- <listing>

- <listingField>

- <calculatedMember>

- <namedSet>

- <relationship>

- <expression>

- <index>

Also see "When to Recompile and Rebuild," earlier in this book, as well as "Using Advanced Features of Cubes and Subject Areas" in the *Advanced DeepSee Modeling Guide*.

A.1 Requirements for a Cube Class

To define a cube, create a class that meets the following requirements:

- It must extend %DeepSee.CubeDefinition.

- It must contain an XData block named `Cube`

- For this XData block, XMLNamespace must be specified as follows:

  ```
  XMLNamespace = "http://www.intersystems.com/deepsee"
  ```

- The root element within the XData block must be `<cube>` and this element must follow the requirements described in the rest of this appendix.

- It is useful, but not required, for the class to specify the DependsOn compiler keyword so that this class is compiled only after the source class of the cube is compiled and can be used.

- The class can define the *DOMAIN* parameter, which specifies the domain to which any localized strings belong. For example:

  ```
  Parameter DOMAIN = "PATIENTSAMPLE";
  ```

  For details, see the chapter "Performing Localization" in the *DeepSee Implementation Guide*.

For example:

```
Class DeepSee.Model.PatientsCube Extends %DeepSee.CubeDefinition [DependsOn=DeepSee.Study.Patient]
{

XData Cube [ XMLNamespace = "http://www.intersystems.com/deepsee" ]
{
<cube
name="Patients"
 owner="_SYSTEM"
 caption="Patients"
 sourceClass="DeepSee.Study.Patient"
 other_cube_options...
>

<measure measure_definition/>
...

<dimension dimension_definition/>
...

</cube>
}
```
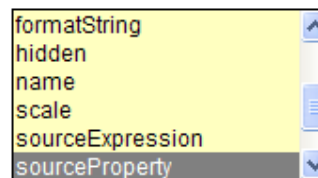
Studio provides assistance as you type. For example:

A.2 Common Attributes in a Cube

Most of the elements in the cube have the following attributes, which are listed here for brevity:

Attribute	Purpose
name	Logical name of the element for use in MDX queries against this cube. See "Names for Model Elements," earlier in this book.

Attribute	Purpose
displayName	(Optional) Localized name of this element for use in user interfaces. If you do not specify this attribute, the user interface instead displays the value specified by the name attribute. For details, see the chapter "Performing Localization" in the *DeepSee Implementation Guide*.
description	(Optional) Description of this element.
disabled	(Optional) Controls whether the compiler uses this element. If this attribute is "true" then the compiler ignores the element; this is equivalent to commenting out the element. By default, this attribute is "false"
additionalDescription	(Optional) Additional notes about this element, for display only within the Architect and Studio.

A.3 <cube>

The <cube> element is the root element in the XData block. This element contains the following contents:

Attribute or Element	Purpose
name, displayName, description, disabled	See "Common Attributes in a Cube" earlier in this appendix.
sourceClass	Complete package and class name of the base class used by this cube. See "Possible Source Classes for a Cube" earlier in this book.
owner	(Optional) Name of owner of cube.
resource	(Optional) Name of the resource used to control access to this cube, when it is accessed via the DeepSee Architect. See the *DeepSee Implementation Guide*.
caption	(Optional) Caption to display in the Analyzer and other utilities when working with this cube.
countMeasureCaption	(Optional) Caption to use for the default measure, which counts records. The default caption is Count. Internally, the name of the measure is %Count.
countMeasureName	(Optional) Name to use for the default measure, which counts records. The default name of this measure is %Count.
defaultListing	(Optional) Specify the logical name of the <listing> to use as the default in this cube. See "<listing>," later in this appendix.
nullReplacement	(Optional) Specifies the string to use as the member name if the source data for a level is null. For example, specify nullReplacement="None" This option is overridden by the nullReplacement attribute, if specified, for a given <level>.

Attribute or Element	Purpose
bucketSize	(Optional) Specifies the size of the caching buckets used for this cube. A *group* is 64,000 contiguous rows, and a *bucket* is an integer multiple of groups. The default bucket size is 8. That is, by default, the first 512,000 rows are bucket number 1, the next 512,000 rows are bucket number 2, and so on. In some cases, you may want to increase this value if you expect to have a large number of facts and do not expect many updates to older values within the cube. If you change this value, be sure to delete all cached results for this cube. To do so, call the **%KillCache()** method of the cube definition class.
actionClass	(Optional) Specifies an associated KPI class that defines actions that are available for pivot tables based on this cube. Specify the full package and class name of a subclass of %DeepSee.KPI.
maxFacts	(Optional) Specifies the maximum number of rows in the fact table. This determines the number of rows of the base table that the system uses when building the cube. Use this attribute to assist in debugging when you do not want to process the entire base table. By default, all rows of the base table are processed.
bitmapChunkInMemory	(Optional) Controls how DeepSee builds the indices: using a process-private global (slower but does not lead to <STORE> errors) or entirely in memory (faster). The default is "false", in which case DeepSee uses a process-private global for temporary storage while it builds in the indices. If your cube does not contain a large number of indices (dimensions and measures) and does not contain dimensions with very large numbers of members, you can use "true" for faster performance. It is worthwhile to try this setting; if you encounter <STORE> errors when you build the cube, then change the value back to the default.
initialBuildOrder	(Optional) Used to construct an optional ORDER BY clause for use when building the entire cube; does not affect incremental updates. Specify a comma-separated list of fields in the source table. You can use the SQL keywords ASC and DESC. For example: "Age DESC,Gender"
buildRestriction	(Optional) Specifies an optional WHERE clause to use when building or updating the cube; this causes the cube to use a subset of the records. Specify an SQL comparison expression that uses fields in the source table. For example: Gender='F'. Has no effect on a cube based on a data connector.
abstract inheritsFrom	(Optional) See "Using Cube Inheritance to Define Reusable Elements," in the *Advanced DeepSee Modeling Guide*.
disableListingGroups	(Optional) Specify this attribute as "true" if you want to prevent anyone from defining a listing group that uses this cube as a target. The default is "false". See "Compiling a Listing Group," earlier in this book.
version	(Optional) See the appendix "Using Cube Versions" in the *DeepSee Implementation Guide*.

Attribute or Element	Purpose
`defaultMeasure,` `defaultMember, precompute`	Do not use.
`<measure>`	(Optional) You can include zero or more `<measure>` elements, each of which defines a measure.
`<dimension>`	(Optional) You can include zero or more `<dimension>` elements, each of which defines a dimension.
`<listing>`	(Optional) You can include zero or more `<listing>` elements, each of which defines a listing, to be available in the Analyzer.
`<listingField>`	(Optional) You can include zero or more `<listingField>` elements, each of which defines a listing, to be available in the Analyzer.
`<namedSet>`	(Optional) You can include zero or more `<namedSet>` elements, each of which defines a named set, which is an alias for a member or for a set of members.
`<calculatedMember>`	(Optional) You can include zero or more `<calculatedMember>` elements, each of which defines a member that is calculated in terms of other members. This is typically used to define measures that are calculated based on other measures.
`<relationship>`	(Optional) You can include zero or more `<relationship>` elements. A `<relationship>` element connects another cube to your cube so that you can use levels of that other cube when you query your own cube.
`<index>`	(Optional) You can include zero or more `<index>` elements, each of which defines an optional, custom index on the fact table. This can be useful if you plan to run SQL queries on the fact table. For information on the fact table, see "Details for the Fact and Dimension Tables."
`<expression>`	(Optional) You can include zero or more `<expression>` elements, each of which defines an expression (typically an intermediate value), for use in the definition of one or more measures or dimensions. Values of expressions are not stored but are available during cube build time.

For example:

```
<cube
name="Patients"
 owner="_SYSTEM"
 caption="Patients"
 sourceClass="DeepSee.Study.Patient"
 >
...
```

A.4 <measure>

The `<measure>` element defines a measure; your cube can have any number of measures. This element has the following attributes:

Attribute	Purpose
`name`, `displayName`, `description`, `disabled`	See "Common Attributes in a Cube" earlier in this appendix.

`sourceProperty` `sourceExpression`	Specify one of these attributes in almost the same way that you would specify **Property** or **Expression** in the Architect; see "Specifying the Source Values for a Dimension or Level" and "Details for Source Expressions," earlier in this book. Notes:
	
	• For `sourceProperty`, use the same value you would enter for **Property**, enclosed within single quotes.
	
	  For example: `sourceProperty='Age'`
	
	• For `sourceExpression`, use the same value you would enter for **Expression**, enclosed within single quotes.
	
	  For example: `sourceExpression='%source.property/100'`
	
	You can enclose the value in double quotes instead if the value itself does not contain any double quotes. For example:
	`sourceExpression="%source.property/100"`

`type`	(Optional) Specifies the data type for the measure. Most measures are numeric, which is the default. See "Specifying the Measure Type."
`iKnowSource`, `iKnowDomain`, `iKnowDictionaries`, `iKnowParameters`	(Optional) See "Using Unstructured Data in Cubes (iKnow)" in the *Advanced DeepSee Modeling Guide*.
`aggregate`	(Optional) Specifies how to aggregate values for this measure, whenever combining multiple records. If you specify this, use one of the following values: • `"SUM"` • `"COUNT"` — Counts the records for which the source data has a non-null (and nonzero) value. • `"MAX"` — Uses the largest value in the set. • `"MIN"` — Uses the smallest value in the set. • `"AVG"` — Calculates the average value for the set. For a boolean or a string measure, this attribute must be `"COUNT"`.
`hidden`	(Optional) If `hidden="true"` then the measure is defined and can be used in queries, but is not listed as an available measure in the Analyzer. This lets you define measures that serve as intermediate calculations.
`scale`	(Optional) Specifies the number of decimal places to keep. By default, `scale` is 0, and DeepSee rounds each measure value to a whole number before writing it to the fact table.
`linkClass`, `linkProperty`	(Optional) See "Linking to Another Table."

Attribute	Purpose
searchable	(Optional) If searchable="true" then DeepSee displays this measure as an option in advanced filters, where you can create filter expressions that refer to the measure value (typically to compare the measure value to a constant).
	For a searchable measure, DeepSee adds an additional index, if appropriate, to support these expressions. See "Measure Search Expressions" in the *DeepSee MDX Reference*. A searchable measure cannot include square brackets or commas ([],) in its name.
	Note that it may be possible to use a measure as a searchable measure in a manual MDX query even if it is not marked as searchable="true". In all cases, the measure is also available for use in the same way as any other measure.
factName	(Optional) Name, in the generated fact table, for the column that corresponds to this measure. If this attribute is null, the system generates a name. This option does not apply to iKnow measures.
	Be sure not to use an SQL reserved word. For a list of the SQL reserved words, see "Reserved Words" in the *Caché SQL Reference*. The name must start with either a letter or a percent sign (%). If the first character is %, the second character must be Z or z. For more details on restrictions, see "Class Members" in the appendix "Rules and Guidelines for Identifiers" in the *Caché Programming Orientation Guide*. Also, do not use fact or listing, in any combination of lowercase and uppercase characters.
factSelectivity	(Optional, not used by DeepSee) Value to override the generated selectivity of the property in the fact table class. DeepSee queries do not use this parameter. This option is intended for cases where SQL is used directly against the generated fact table and the generated selectivity needs to be overridden. Specify a positive value less than or equal to 1. For details, see **SetFieldSelectivity()** in %SYSTEM.SQL.
formatString	(Optional) Controls how the values are displayed. See "formatString Details."
units	(Optional) Indicates the units in which the measure value is expressed. Currently this attribute is provided only for general information.
listingFilterOperator and listingFilterValue	(Optional) These attributes specify an optional additional filter that is applied when a user selects this measure in a pivot table and then displays a listing. For details, see "Specifying Additional Filtering for Listings," in the chapter "Defining Measures."

For example:

```
<measure name="Test Score" sourceProperty="TestScore" aggregate="SUM"/>
<measure name="Avg Test Score" sourceProperty="TestScore" aggregate="AVG"/>
<measure name="Allergy Count"
    sourceExpression="##class(Cubes.StudyPatients).GetAllergyCount(%source.%ID)"/>
```

Notice that the Allergy Count measure uses a utility method that is defined within the cube class.

A.4.1 %COUNT Measure

DeepSee includes a predefined measure named %COUNT, which returns the number of facts in a query.

A.4.2 Measures Dimension

DeepSee automatically creates the Measures dimension, and places all measures into it.

A.4.3 formatString Details

The `formatString` attribute is a pieced string as follows:

```
Format^Color
```

- For the *Format* piece, specify a string that consists of one to four subpieces as follows:

```
positive_format;negative_format;zero_format;missing_format;
```

  Where *positive_format* controls how a positive value is displayed, *negative_format* controls how a negative value is displayed, *zero_format* controls how zero is displayed, and *missing_format* controls how a missing value is displayed.

  For details, see "Specifying a Format String," earlier in this book.

- For the *Color* piece, specify a string that consists of one to four subpieces as follows:

```
positive_color;negative_color;zero_color;missing_color;
```

  Where *positive_color* controls the color for a positive value, *negative_color* controls the color for a negative value, *zero_color* controls the color for zero, and *missing_color* controls the color for a missing value.

  For each of these, specify a CSS color name or a hex color code.

For date measures, use the following variation:

```
%date%^Color
```

Where *Color* is as given above. The `%date%` piece formats the date according to the default date format for the current process.

A.5 <dimension>

The `<dimension>` element defines a dimension, which contains one or more hierarchies, each of which contains one or more levels. Your cube can have any number of dimensions.

The `<dimension>` element has the following contents:

Attribute or Element	Purpose
`name`, `displayName`, `description`, `disabled`	See "Common Attributes in a Cube" earlier in this appendix.

Attribute or Element	Purpose

sourceProperty
 	
sourceExpression
	Specify one of these attributes in almost the same way that you would specify **Property** or **Expression** in the Architect; see "Defining the Source Values for a Dimension or Level." Notes:
	
	• For sourceProperty, use the same value you would enter for **Property**, enclosed within single quotes.
	
	  For example: sourceProperty='Age'
	
	• For soureExpression, use the same value you would enter for **Expression**, enclosed within single quotes.
	
	  For example: sourceExpression='%source.property_"ABC"'
	
	You can enclose the value in double quotes instead if the value itself does not contain any double quotes. For example:
	sourceExpression="%source.property"

type	Specify one of the following: • "data" (the default) — Creates an ordinary dimension. • "time" — Creates a time dimension. See "Defining a Time Level." • "age" — Creates an age dimension. See "Defining an Age Level." • "computed" — Creates a computed dimension. See "Defining Computed Dimensions" in the *Advanced DeepSee Modeling Guide*. • "iKnow" — Creates a dimension uses text processed by iKnow. See "Using Unstructured Data in Cubes (iKnow)" in the *Advanced DeepSee Modeling Guide*.
calendar	(Optional) Specify this only if type is "time". The calendar attribute specifies the calendar to use when assigning records into members of time levels. Use "gregorian" (the default) for the Gregorian calendar. Or use "hijriTabular" or "hijriObserved" for a Hijri calendar.
iKnowMeasure and iKnowType	(Optional) See "Using Unstructured Data in Cubes (iKnow)" in the *Advanced DeepSee Modeling Guide*.
hasAll	(Optional) Indicates whether this dimension has an All Level. The default is "true".
allCaption	(Optional) Name used for the All Level and All Member for this dimension. The default name is All dimension, where *dimension* is the dimension name.
allDisplayName	(Optional) Specifies the localized name used for the All member. If you do not specify this attribute, the user interface instead displays the value specified by the allCaption attribute.
hidden	(Optional) If hidden="true" then the dimension is defined and can be used in queries, but the Analyzer does not list the dimension as available for use. The default is "false".

Attribute or Element	Purpose
`showHierarchies`	(Optional) Controls whether the Analyzer displays the hierarchy names within this dimension. Specify one of the following: • `"default"` — Display the hierarchy names only if there is more than one hierarchy. • `"true"` — Always display the hierarchy names. • `"false"` — Never display the hierarchy names. This attribute has no effect on the queries themselves.
`sharesFrom`	(Optional) See "Defining a Shared Dimension," in the *Advanced DeepSee Modeling Guide*. Do not specify this option for date dimensions, which are automatically shared.
`dimensionClass`	(Optional) See "Defining Computed Dimensions" in the *Advanced DeepSee Modeling Guide*.
`<hierarchy>`	Specifies a hierarchy within this dimension. You must include at least one `<hierarchy>` element and can include multiple `<hierarchy>` elements.

A.6 <hierarchy>

The `<hierarchy>` element specifies a hierarchy within the given dimension. This element has the following contents:

Attribute or Element	Purpose

`name,`
`displayName,`
`description,`
`disabled`
	See "Common Attributes in a Cube," earlier in this appendix.

`hidden`	(Optional) If `hidden="true"` then the hierarchy is defined and can be used in queries, but the Analyzer does not list the hierarchy and its levels as available for use. The default is `"false"`.
`<level>`	Specifies a level within this hierarchy. You must include at least one `<level>` element and you can include multiple `<level>` elements.

A.7 <level>

The `<level>` element specifies a level within the given hierarchy. It has the following contents:

Attribute	Purpose

`name, displayName,`
`description,`
`disabled`
	See "Common Attributes in a Cube," earlier in this appendix.

Attribute	Purpose

sourceProperty, sourceExpression	Specify *one* of these attributes. The syntax is almost the same as you would use for **Property** or **Expression** in the Architect; see "Defining the Source Values for a Dimension or Level." Notes:
	
	• For sourceProperty, use the same value you would enter for **Property**, enclosed within single quotes.
	
	    For example: sourceProperty='Age'
	
	• For sourceExpression, use the same value you would enter for **Expression**, enclosed within single quotes.
	
	    For example: sourceExpression='%source.property_"ABC"'
	
	You can enclose the value in double quotes instead if the value itself does not contain any double quotes. For example:
	sourceExpression="%source.property"
	For the requirements for levels within a time or age dimension, see "Defining a Time Level" and "Defining an Age Level."
	
	For levels in an iKnow dimension, sourceProperty and sourceExpression are ignored, because these levels use a different mechanism to specify the source values. See "Using Unstructured Data in Cubes (iKnow)" in the *Advanced DeepSee Modeling Guide*.

timeFunction	Only for levels within a time or age dimension. See "Defining a Time Level" and "Defining an Age Level."
timeOffset	(Optional) Only for levels within a time dimension. See "Specifying a Date Offset."
timeFormat	(Optional) Only for levels within a time dimension. See "Specifying a Time Format" in the chapter "Details of Defining Levels."
list	(Optional) If this attribute is "true" then the source value is expected to be a list, and each item in the list becomes a member of this level. The default is "false". By default, the list is expected to be in **$LIST** format. To use a string consisting of a character-delimited list, specify listDelimiter. A list-based level cannot have parent levels or child levels.
listDelimiter	(Optional) Specifies the delimiter used to separate items in the list that is used as the source data for the level. Use this if the list is a character-separated list.
nullReplacement	(Optional) Specifies the string to use as the member name if the source data for this level is null. For example, specify nullReplacement="No City".
rangeExpression	(Optional) For numeric data, this specifies how to assign numeric values to bins (each bin is a member of this level). For other data, this specifies replacement values. This attribute also controls the default order of the members of this level. See the subsections "Defining a Basic Range Expression" and "Defining a Compact Multi-Range Expression."

Attribute	Purpose
useDisplayValue	(Optional) For properties that have values for the *DISPLAYLIST* and *VALUELIST* parameters, this attribute specifies which value to include in the index. If this attribute is `"false"` (the default), the system uses the value given by *VALUELIST*; if this attribute is `"true"`, the system uses the value given by *DISPLAYLIST*. This option is ignored if you specify `linkClass` and `linkProperty`.
sort	(Optional) Specifies how to sort members of this level by default. For a level in a time dimension, specify `"asc"` or `"desc"` For a level in a data dimension, specify `"asc"`, `"asc numeric"`, `"desc"`, or `"desc numeric"`
linkClass, linkProperty	(Optional) See "Linking to Another Table."
factName	(Optional) Name, in the generated fact table, for the column that corresponds to this level. If this attribute is null, DeepSee generates a name. This option does not apply to time levels or iKnow levels. See the comments for `factName` for <measure>.
dependsOn	(Optional) Specifies the level (or relationship) on which this level has a dependency. Specify the full MDX identifier of the level (or relationship) . Or specify a comma-separated list of MDX level (or relationship) identifiers. See "Defining Dependencies Between Levels in Different Hierarchies." This attribute is completely unrelated to the DependsOn compiler keyword.
useAsFilter	(Optional) Specifies whether the level can be used as a filter control in the dashboard. If this attribute is `"true"` (the default), users can select this level when adding a filter control; if this attribute is `"false"`, the level is not listed as an option. The purpose of this option is only to reduce the number of choices seen in the Dashboard Editor, and there is no effect on the Analyzer or on the engine.
factSelectivity	(Optional, not used by DeepSee) Value to override the generated selectivity of the property in the generated level table. DeepSee queries do not use this parameter. This option is intended for cases where SQL is used directly against the level table and the generated selectivity needs to be overridden. Specify a positive value less than or equal to 1. For details, see **SetFieldSelectivity()** in %SYSTEM.SQL.
hidden	(Optional) If `hidden="true"` then the level is defined and can be used in queries, but the Analyzer does not list the level as available for use. The default is `"false"`.
<member>	(Optional) You can include any number of `<member>` elements, each of which defines a member of the level.
<property>	(Optional) You can include any number of `<property>` elements, each of which defines a property of the level.

For example:

```
<level name="ZIP" sourceProperty="HomeCity.PostalCode" />
```

For another example, the following level definition also defines a property for that level:

```
<level name="City" sourceProperty="HomeCity.Name">
  <property name="Population" sourceProperty="HomeCity.Population" />
```

A.7.1 Linking to Another Table

Within a `<level>`, `<measure>`, or `<property>` element, the `linkClass` and `linkProperty` attributes enable you to use a property in another class, a class that you cannot access via Caché dot syntax. To use the link feature, you must be able to look up a record in the other class by the ID of the record.

The feature works as follows:

- Specify `sourceProperty` or `sourceExpression` as a value that gives the ID of the desired record in the other class.

- Specify `linkClass` as the complete package and class name for the other class.

- Specify `linkProperty` as the property in the other class on which you want to base the level that you are defining.

For example, you could add the following to the Hole Foods sample:

```
<level name="Product" sourceProperty ="Product" linkClass="HoleFoods.Product" linkProperty="Name" />
```

If the external class `linkClass` and a property `linkProperty` are defined, DeepSee runs a query against the external class to fetch the value of the given property from that class, for the record whose ID equals the given `sourceProperty` or `sourceExpression`.

A.7.2 Specifying a Date Offset

In some cases, you may need a time level to match a corporate financial calendar that does not start on 1 January. For example, in many companies, the financial year starts 1 Oct. Consider the following pivot table:

Year	Amount Sold
FY 2005	7,131.04
FY 2006	12,892.19
FY 2007	14,653.86
FY 2008	16,540.95
FY 2009	18,987.04
FY 2010	16,666.45

In this case, the member `FY 2005` consists of records of sales between 1 October 2004 and 30 September 2005, inclusive.

To create a level like this, specify the `timeOffset` and `timeFormat` attributes of the level (which must be within a time-type dimension). This section discusses the `timeOffset` attribute; the next section discusses `timeFormat`.

The `timeOffset` attribute specifies an amount of time to add to the source values used in this level; this amount of time can be negative or positive. The system uses this at cube build time.

For `timeOffset`, specify an amount of time by specifying a string of the following form:

```
#y#m#d
```

Where # is a number, #y represents an amount of time in years, #m represents an amount of time in months, and #d represents an amount of time in days. If you omit an element, DeepSee uses zero in its place. For example, the string `3m15d` represents three months and 15 days.

The most common value for `timeOffset` is `-3m`, which you use if the fiscal year starts in 1 October of the previous year. If `timeOffset="-3m"`, DeepSee subtracts three months to each time value used in this level. For example, for this level, the date 1 Jan 2010 is converted to 1 Oct 2009.

Other levels are unaffected, even within the same dimension. This means that you can also define more granular levels that display the actual dates. For an example, see "Handling a Calendar That Has a Date Offset."

A.7.3 Defining a Basic Range Expression

For `rangeExpression`, the basic syntax is as follows:

```
value_or_range:new_value;value_or_range:new_value; ... ;
```

Here *new_value* is a single value, which DeepSee treats as a string and uses as the name of a level member. Also, *value_or_range* is any of the following:

- A single value such as `5` or `Louisiana`

- A range in the form `(value1,value2)`

- A range in the form `[value1,value2]`

- A range in the form `(value1,value2]`

- A range in the form `[value1,value2)`

In these expressions, *value1* and *value2* are numeric values or null (that is, omitted from the expression). The left and right parentheses indicate that the range is not inclusive at the given endpoint. The left and right square brackets indicate that the range is inclusive at the given endpoint. For example, `(45,49]` represents all values greater than 45 and less than or equal to 49.

A.7.4 Defining a Compact Multi-Range Expression

In some cases, it is necessary to define many ranges for a given level, which can be a tedious process. Also the resulting range expression (when viewed in Studio) is difficult to read. In such cases, you may want to use the following alternative syntax for the value of `rangeExpression`:

```
rangeExpression="[start:increment:end]:replacement;"
```

Or use the appropriate parenthesis in place of either square bracket (details below).

This syntax generates a series of ranges.

Here, *start* is the numeric start value of the first range, *end* is the numeric end value of the last range, and *increment* is the numeric value by which the ranges are defined. The first range extends from *start* to *start* plus *increment*, and so on.

Also, *replacement* is an expression that is used as the replacement value. Within *replacement*, you can use the following elements:

- `%1` — is replaced with the start value of the range

- `%2` — is replaced with the end value of the range

- `$$$eval(`*expression*`)` — causes the system to evaluate the contained expression. For example: `$$$eval(%2-1)` returns the end value of the range minus one

The square brackets or parentheses affect how the *start* and *end* values are treated, in the generated start and end ranges:

- A square bracket causes the value to be included in the range.

- A parenthesis causes the value to be excluded from the range.

At the intermediate range boundaries, the boundary value is always assigned to the upper range, rather than the lower range. That is, for intermediate ranges, the opening bracket is always [and the closing bracket is always).

Consider the following range expression:

```
rangeExpression="[0:30:90]:%1 to $$$eval(%2-1);"
```

This generates the same members as the following longer form:

```
rangeExpression="[0,30):0 to 29;[30,60):30 to 59;[60,90]:60 to 90;"
```

If the value *end - start* is not an integer multiple of *increment* , the last range extends beyond *end*. For example, consider the following range expression:

```
rangeExpression="[0:30:100]:%1 to $$$eval(%2-1);"
```

This generates the same members as the following longer form:

```
rangeExpression="[0,30):0 to 29;[30,60):30 to 59;[60,90):60 to 90;[90,119]:90 to 119;"
```

A.8 <member>

The `<member>` element is primarily for advanced use. See "Manually Specifying the Members of a Level," in the chapter "Using Additional Advanced Features of Cubes and Subject Areas" in the *Advanced DeepSee Modeling Guide*.

The `<member>` element has the following contents:

Attribute	Purpose

`name,`
`displayName,`
`description,`
`disabled`
	See "Common Attributes in a Cube," earlier in this appendix.

`spec`	If the level is contained in a dimension that has `type="computed"`, specify the SQL query that returns the IDs of the fact table rows (or the source table rows) that this member uses. In other cases, optionally specify the value to use as the member key. If you omit this, `name` becomes the member key.

A.9 <property>

A level may contain zero or more custom level properties. These are properties whose value is derived from the source data and is associated with a specific member of a level. For example, the city level could include properties such as population or ZIP code. For each city, there would be one value of each of these properties.

The `<property>` element has the following contents:

Attribute	Purpose

`name, displayName,`
`description,`
`disabled`
	See "Common Attributes in a Cube," earlier in this appendix.

Attribute	Purpose

sourceProperty
 	
sourceExpression
	Specify one of these attributes in almost the same way that you would specify **Property** or **Expression** in the Architect; see "Specifying the Source Values for a Dimension or Level" and "Details for Source Expressions." Notes:
	
	• For sourceProperty, use the same value you would enter for **Property**, enclosed within single quotes.
	
	    For example: sourceProperty='MyProp'
	
	• For sourceExpression, use the same value you would enter for **Expression**, enclosed within single quotes.
	
	    For example: sourceExpression='%source.MyProp_"ABC"'
	
	You can enclose the value in double quotes instead if the value itself does not contain any double quotes. For example: sourceExpression="%source.MyProp"

sort	(Optional) Specifies how to use this property to sort members of the level that contains this property. You can sort a level by multiple properties; if you do, the sorting is applied in the order in which you define the <property> elements; the first property controls the primary sort, the second property controls the secondary sort, and so on. Specify "asc", "asc numeric", "desc", or "desc numeric" By default, the property does not affect the sort order of the members.
isName	(Optional) If "true", this attribute specifies that, for a given level member, DeepSee should use the value of this property to specify the names of that member. Specify either "true" or "false" (the default).
isDescription	(Optional) If "true", this attribute specifies that, for a given member, DeepSee should use the value of this property as the tooltip for the member. Specify either "true" or "false" (the default).
isReference	(Optional) If "true", this attribute specifies that DeepSee should not store the property value, but should instead define the property as an SQL computed field that refers to the original source tables. If you specify isReference="true", the level must be defined so that the member keys are the IDs of the records on which the level (and its properties) are based.
useDisplayValue	(Optional) For class properties that have values for the *DISPLAYLIST* and *VALUELIST* parameters, this attribute specifies which value to use for the property. If this attribute is "true" (the default), the system uses the value given by *DISPLAYLIST*; if this attribute is "false", the system uses the value given by *VALUELIST*.
linkClass, linkProperty	(Optional) See "Linking to Another Table."

Attribute	Purpose
`factName`	(Optional) Name used (in the generated dimension table) for the column that corresponds to this property. If this attribute is null, DeepSee generates a name. You do not need to worry about this unless you plan to issue SQL queries directly against the generated dimension table. Despite the name of this attribute, properties are not contained in the fact table. They are contained in the table that corresponds to the level to which the property belongs. See the comments for `factName` for <measure>.
`formatString`	(Optional) Controls how the values are displayed. See "formatString Details," earlier in this appendix. In the format string, you can also use the special character `0`, which serves as a placeholder for leading zeros. For example, `00000` would show a five-digit number padded with leading zeros.
`hidden`	(Optional) If `hidden="true"` then the property is defined and can be used in queries, but is not listed as an available property in the Analyzer.

For example:

```
<property name="Population" sourceProperty="City.Population"/>
```

A.9.1 Intrinsic Properties

For each level, DeepSee also automatically defines a set of intrinsic properties. These are listed in the *DeepSee MDX Reference*.

A.10 <listing>

A cube can contain zero or more named listings. These are available in the Analyzer.

The *default listing* for a cube is either the listing specified in the `defaultListing` attribute for the `<cube>` element (if specified) or the first `<listing>` element contained in the `<cube>`.

The `<listing>` element has the following contents:

Attribute	Purpose

`name,`
`displayName,`
`description,`
`disabled`
	See "Common Attributes in a Cube," earlier in this appendix.

`fieldList`	Comma-separated list of fields to display. For options, see "Specifying a Simple Listing," earlier in this book. For options in a data connector listing, see "Specifying a Data Connector Listing."

Attribute	Purpose
formatList	List (delimited by carets) of CSS formatting instructions for the listing. Each piece of the list is applied to the corresponding column in the listing. This list must have the same number of elements as there are columns, and each list element must have one of the following forms: • {*CSS formatting instruction*}, for example, {text-align:center;} — Formats the corresponding column. You can use {text-align:left;} or {text-align:center;}. Other instructions are ignored. • display:none; — Hides the corresponding column. • Null — Displays the corresponding column in the default style.
orderBy	Applies only to simple listings. This attribute is a comma-separated list of fields by which to sort the listing. The overall sort is controlled by the first field in the list, the secondary sort is controlled by the second field, and so on. After a field name, you can include the ASC or DESC keyword to sort in ascending or descending order, respectively. For additional options, see "Specifying a Simple Listing," earlier in this book. This attribute is ignored if the source is a data connector.
sourceClass	Specifies the data connector, if any, on which this listing is based. If you specify this attribute, specify the name of a class that extends %DeepSee.DataConnector. See the *DeepSee Implementation Guide*.
sql	Specifies the custom SQL query, if any, for this listing. See "Specifying a Custom Listing," earlier in this book.
listingType	(Optional) Specifies the format of the listing. The default is "table". If you instead specify "map" the system displays a map-type listing, which displays a map with points marked by latitude and longitude. In this case, the listing query must contain the fields Latitude and Longitude (case-sensitive). See "Defining a Map-type Listing," earlier in this book.
resource	(Optional) Name of the resource used to control access to this listing. See the *DeepSee Implementation Guide*.

There are three kinds of listings, considering their sources:

• Basic listings, which directly use the source table used by the cube

• Data connector listings (which use data connectors)

• Custom listings (which use custom queries)

In all cases, the system creates and uses an SQL query.

The following table indicates which attributes of <listing> you specify for these kinds of listings:

Kind of Listing	`fieldList`	`orderBy`	`sourceClass`	`sql`
Basic	Required	Optional	Do not specify; this attribute would take precedence.	Do not specify; this attribute would take precedence.
Data connector	Optional	Ignored	Required	Ignored
Custom	Ignored	Ignored	Do not specify; this attribute would take precedence.	Required

A.11 <listingField>

A cube can define zero or more listing fields that enable end users to create custom listings in the Analyzer (see "Creating a Custom Listing" in *Using the DeepSee Analyzer*).

Note: DeepSee supports another kind of custom listing — a listing that does not necessarily use the source table of the cube. See "Defining a Custom Listing" in the chapter "Defining Listings."

The `<listingField>` element has the following contents:

Attribute	Purpose
`name`, `displayName`, `description`, `disabled`	See "Common Attributes in a Cube," earlier in this appendix.
`fieldExpression`	SQL expression that refers to a field in the source table or (via arrow syntax, for example) to a field in a related table. If you specify `displayName`, do not specify an alias within the SQL expression. Conversely, if you specify an alias within the SQL expression, `displayName` is ignored and the field name cannot be localized. For details, see "Creating a Listing Field," earlier in this book.
`resource`	(Optional) Name of the resource used to control access to this listing field. See the *DeepSee Implementation Guide*.

For example:

```
<listingField name="PatientID" displayName="PatientID" fieldExpression="PatientID" />
<listingField name="Age" displayName="Age" fieldExpression="Age" />
<listingField name="Gender" displayName="Gender" fieldExpression="Gender" />
<listingField name="Test Score" displayName="Test Score" fieldExpression="TestScore" />
<listingField name="City" displayName="City" fieldExpression="HomeCity->Name" />
<listingField name="Doctor" displayName="Doctor" fieldExpression="PrimaryCarePhysician->LastName" />
```

A.12 <calculatedMember>

A cube can contain zero or more *calculated members*. A calculated member is a member that is defined in terms of other members. You can add two kinds of calculated member:

- You can define a new measure that is based on other measures. For example, you can define a measure via a formula like the following:

  ```
  Measure 3 = (Measure 1 + Measure 2) / Measure 2
  ```

  This is not the exact syntax.

- You can define a new member that is based on other non-measure members. For example, you could create a `Primary Colors` member that combines the `red`, `yellow`, and `blue` members of the `Favorite Color` dimension.

  The new `Primary Colors` member refers to all the records of the fact table that correspond to the `red`, `yellow`, or `blue` members.

In MDX, a measure is considered to be a member, which is why both kinds of calculated elements are considered to be *calculated members*.

Note: When you use this element to define a measure, the `cube` command in the MDX shell does not currently list this measure. You can, however, use the set in the MDX shell or in the query API.

The `<calculatedMember>` element has the following contents:

Attribute	Purpose
`name`, `displayName`, `description`, `disabled`	See "Common Attributes in a Cube," earlier in this appendix. For the `name` attribute, DeepSee does not check whether the name is already in use. If you use a name that was previously used by another member of this dimension, you are overriding that member.
`dimension`	Dimension to which this member belongs.
`valueExpression`	MDX expression that defines the values for this member in terms of references to other members. For simple but common scenarios, see "Defining Calculated Elements" in *Using the DeepSee Analyzer*. For details and examples, see the *DeepSee MDX Reference*.
`formatString`	(Optional) Controls how the values are displayed. See "Specifying a Format String."
`units`	(Optional) Indicates the units in which the measure value is expressed. Currently this attribute is provided only for general information.
`listingFilter`	(Optional, applies only if `dimension` is `"measures"`) Specifies an additional filter that is used when a user displays this calculated measure in a pivot table and then requests a detail listing. See "Specifying Additional Filtering for Listings for a Calculated Measure," earlier in this book.

For example:

```
<calculatedMember name="Avg Age" dimension="MEASURES"
valueExpression="[MEASURES].[Age]/[MEASURES].[%COUNT]"/>
```

In any context where you use this calculated member, DeepSee first evaluates the `Age` and `Patient Count` measures in that context and then performs the division.

Note: You can define a calculated member in two other ways:

- Within an MDX query by using the WITH clause.

- Within the CREATE MEMBER statement.

  This is useful only within the MDX Shell.

  See the *DeepSee MDX Reference*.

A.13 <namedSet>

A cube can contain zero or more named sets. A named set is an alias for a member or for a set of members. You can use named sets for rows or columns in your queries, and DeepSee substitutes the member or the set when you run the query.

Note: When you define a named set, the `cube` command in the MDX shell does not list this set. You can, however, use the set in the MDX shell or in the query API.

The `<namedSet>` element has the following contents:

Attribute	Purpose

`name,`
`displayName,`
`description,`
`disabled`
	See "Common Attributes in a Cube," earlier in this appendix. For the `name` attribute, DeepSee does not check whether the name is already in use. If you use a name that was previously used by another named set, you are overriding that named set.

`setExpression`	MDX expression that returns a member or a set of members. For details and examples, see the *DeepSee MDX Reference*.

For example:

```
<namedSet name="SampleSet" setExpression="[homed].[h1].[city].MEMBERS" />
```

Note: You can define a named set in two other ways:

- Within an MDX query by using the WITH clause.

- Within the CREATE SET statement.

See the *DeepSee MDX Reference*.

A.14 <relationship>

To define a one-way *relationship* from one cube to another cube, you define a `<relationship>` element in the first cube.

To define a two-way *relationship*, you define two complementary `<relationship>` elements, one in each cube.

The `<relationship>` element has the following attributes:

Attribute	Purpose
`name`	Name of the relationship. You use this logical name in MDX queries to use levels of the other cube. Typically this is the name of the other cube.

`displayName,`
`description,`
`disabled`
	See "Common Attributes in a Cube," earlier in this appendix.

`relatedCube`	Logical name of the other cube.
`inverse`	Value of the `name` attribute in the `<relationship>` element in the other cube.
`cardinality`	Cardinality of the relationship.

Attribute	Purpose
sourceProperty sourceExpression	See "Defining a One-Way Relationship" and "Defining a Two-Way Relationship" in "Defining Cube-Cube Relationships" in the *Advanced DeepSee Modeling Guide*.
nullReplacement	(Optional) Specifies the string to use as the member name if the source data for this relationship is null. For example, specify nullReplacement="No City". Specify this attribute within a `<relationship>` that specifies sourceProperty or sourceExpression
factName	(Optional) Name used (in the fact table) for the column that corresponds to this relationship. If this attribute is null, the system generates a name. See the comments for factName for `<measure>`.
linkClass, linkProperty	Do not use. These are ignored.
dependsOn	(Optional) Specifies the relationship on which this relationship has a dependency. Specify the logical name of the relationship. See "Defining Dependencies Between Levels in Different Hierarchies." (Or, if the relationship depends upon a level, specify the MDX identifier of that level.) This attribute is completely unrelated to the DependsOn compiler keyword.

Important: When you compile the cube classes, first compile the independent cube, which is the one that does *not* define a source property or source expression for the relationship. To control the compilation order, specify the DependsOn keyword in the class definition of the dependent cube.

Similarly, you must build the independent cube first. The DependsOn keyword has no effect on the order in which cubes are built.

The following table summarizes which keywords to specify in each of scenarios:

Attribute	In a One-way Relationship...	In a Two-way Relationship...
inverse	Omit this	Specify this in both cubes
cardinality	Use "one"	• In the dependent cube, use "one" • In the independent cube, use "many"
sourceProperty or sourceExpression	Specify as usual	Applicable only in the dependent cube
factName	Specify as usual	Applicable only in the dependent cube; ignored in the other cube

For examples, see in "Defining Cube-Cube Relationships" in the *Advanced DeepSee Modeling Guide*.

A.15 <expression>

The `<expression>` element defines an optional expression that has a value for each row in the source table. Your cube can have any number of expressions.

<index>

The `<expression>` element has the following contents:

Attribute	Purpose
`name`, `description`, `disabled`	See "Common Attributes in a Cube," earlier in this appendix.
`sourceProperty`, `sourceExpression`	See the *Advanced DeepSee Modeling Guide*.
`displayName`, `factName`, `linkClass`, `linkProperty`	Do not specify these attributes.

A.16 <index>

The `<index>` element specifies an optional, custom index that you add to the fact table. DeepSee does not use this index (because it automatically adds the indices that it needs).

You can add a custom index if you plan to access the fact table via SQL. The `<index>` element has the following contents:

Attribute	Purpose
`name`, `displayName`, `description`, `disabled`	See "Common Attributes in a Cube," earlier in this appendix.
`type`	Specifies the type of index. Use `"bitmap"`, `"bitslice"`, `"index"`, or `"key"`.
`properties`	Specifies the fields of the fact table on which this index is based. Specify a comma-separated list of properties of the fact table class.

For example:

```
<index name="IndexName" type="bitmap"  properties="MxAge,DxGender"/>
```

Tip: Remember that you can specify the `factName` attribute for most levels and measures, to control the names of the properties in the generated fact table class.

B
Reference Information for Subject Area Classes

A subject area is analogous to an SQL view of a table. A subject area is a subcube that enables users to focus on smaller sets of data without the need for multiple cubes. A subject area also enables you to customize captions and defaults of the cube.

This appendix contains reference information for subject area classes. It discusses the following topics:

- Basic requirements for a subject area class

- Common attributes in a subject area

- <subjectArea>

- <measure>

- <dimension>

- <hierarchy>

- <listing>

- <calculatedMember>

- <namedSet>

Also see "Using Advanced Features of Cubes and Subject Areas" in the *Advanced DeepSee Modeling Guide*.

You must recompile a subject area class after making any change.

B.1 Requirements for a Subject Area Class

To define a subject area, create a class that meets the following requirements:

- It must extend %DeepSee.SubjectArea.

- It must contain an XData block named `SubjectArea`

- For this XData block, XMLNamespace must be specified as follows:

    ```
    XMLNamespace = "http://www.intersystems.com/subjectarea"
    ```

- The root element within the XData block must be `<subjectArea>` and this element must follow the requirements described in the rest of this appendix.

- It is useful, but not required, for the class to specify the DependsOn keyword so that this class is compiled only after the cube class is compiled and can be used.

- The class can define the *DOMAIN* parameter, which specifies the domain to which any localized strings belong. For example:

```
Parameter DOMAIN = "PATIENTSAMPLE";
```

  For details, see the chapter "Performing Localization" in the *DeepSee Implementation Guide*.

For example:

```
Class DeepSee.Model.SubjectAreas.AsthmaPatients Extends %DeepSee.SubjectArea
[DependsOn=Cubes.StudyPatients]
{

/// This XData definition defines the SubjectArea.
XData SubjectArea [ XMLNamespace = "http://www.intersystems.com/deepsee/subjectarea" ]
{
<subjectArea name="AsthmaPatients"
    displayName="Asthma Patients"
    baseCube="Patients" filterSpec="diagd.h1.diagnoses.asthma" >
    </subjectArea>
}

}
```

Studio provides assistance as you type.

B.2 Common Attributes in a Subject Area

Most of the elements in the subject area have the following attributes, which are listed here for brevity:

Attribute	Purpose
`name`	Logical name of the element as specified in the base cube on which this subject area is based.
`displayName`	(Optional) Localized name of this element for use in user interfaces. If you do not specify this attribute, the user interface instead displays the logical name. For details, see the chapter "Performing Localization" in the *DeepSee Implementation Guide*.
`description`	(Optional) Description of this element. If you do not specify this attribute, the user interface instead displays the `description` specified in the cube definition.
`disabled`	(Optional) Controls whether the compiler uses the override defined by this element. If this attribute is `"true"` then the compiler ignores this override; this is equivalent to commenting out the override. By default, this attribute is `"false"` If the override is disabled, then DeepSee uses the definition as given in the cube.

B.3 \<subjectArea\>

The \<subjectArea\> element is the root element in the XData block in a subject area class. This element contains the following items:

Attribute or Element	Purpose
`name`, `displayName`, `description`, `disabled`	See "Common Attributes in a Subject Area" earlier in this appendix. Note that the `name` attribute must be unique within the Caché namespace.
`baseCube`	Logical name of the cube on which this subject area is based. (This can also be a comma-separated list of logical cube names; see "Defining Compound Cubes" the *Advanced DeepSee Modeling Guide*.)
`owner`	(Optional) Name of owner of subject area.
`resource`	(Optional) Name of the resource used to control access to this subject area, when it is accessed via the DeepSee Architect. See the *DeepSee Implementation Guide*.
`filterSpec`	(Optional) MDX set expression to use as a filter for this subject area. See the subsection "Filtering a Subject Area." The default is an empty string so that there is no filtering.
`caption`	(Optional) Caption for this subject area. If you do not specify this, the system uses the caption of the base cube instead.
`countMeasureCaption`	(Optional) Caption to use for the default measure, which counts records. The default caption is `Count`. Internally, the name of the measure is `%Count`.
`defaultListing`	(Optional) Specify the logical name of the `<listing>` to use as the default in this subject area; see "\<listing\>." If you do not specify this, the system uses the default listing as specified in the base cube.
`disableListingGroups`	(Optional) Specify this attribute as `"true"` if you want to prevent anyone from defining a listing group that uses this subject area as a target. The default is `"false"`. See "Compiling a Listing Group," earlier in this book.
`defaultMember`, `defaultMeasure`	Do not use.
\<measure\>	(Optional) You can include zero or more `<measure>` elements, each of which can hide or customize a measure.
\<dimension\>	(Optional) You can include zero or more `<dimension>` elements, each of which can hide or customize a dimension.
\<listing\>	(Optional) You can include zero or more `<listing>` elements, each of which can hide, customize, or add a listing for this subject area.

B.3.1 Filtering a Subject Area

The `filterSpec` attribute enables you to specify a filter that applies to the subject area. This attribute must equal a valid MDX set expression. For example:

```
{AgeD.H1.[10 to 19],AgeD.H1.[20 to 29]}
```

See "Writing Filter Expressions," earlier in this book.

Instead of (or in addition to) specifying filterSpec, you can implement the **%OnGetFilterSpec** callback; see "Filtering a Cube or Subject Area Dynamically" in the *Advanced DeepSee Modeling Guide*.

B.4 \<measure>

Within a subject area, a \<measure> element hides or customizes a measure. This element has the following attributes:

Attribute	Purpose
name, displayName, description, disabled	See "Common Attributes in a Subject Area" earlier in this appendix.
hidden	(Optional) If hidden="true" then the measure is defined and can be used in queries, but is not listed as an available measure in this subject area. The default is "false".
formatString	(Optional) If specified, this attribute overrides the formatString attribute specified for this measure in the base cube. See "Specifying a Format String."

Note: You cannot define overrides for calculated measures this way. A calculated measure is actually a calculated member; to define an override for it, you must use \<calculatedMember>.

B.5 \<dimension>

Within a subject area, a \<dimension> element hides or customizes a dimension. This element has the following contents:

Attribute or Element	Purpose
name, displayName, description, disabled	See "Common Attributes in a Subject Area" earlier in this appendix.
hidden	(Optional) If hidden="true" then the dimension is defined and can be used in queries, but is not listed as an available dimension. The default is "false".
allCaption	(Optional) If specified, this attribute overrides the allCaption attribute specified for this dimension in the base cube.
allDisplayName	(Optional) If specified, this attribute overrides the allDisplayName attribute specified for this dimension in the base cube.
\<hierarchy>	(Optional) You can include zero or more \<hierarchy> elements, each of which can hide a hierarchy (or part of a hierarchy).

B.6 \<hierarchy>

Within a subject area, a \<hierarchy> element hides or customizes a hierarchy. This element has the following contents:

Attribute or Element	Purpose
`name`, `displayName`, `description`, `disabled`	See "Common Attributes in a Subject Area" earlier in this appendix.
`<level>`	(Optional) You can include zero or more `<level>` elements, each of which can hide a level.
`hidden`	(Optional) If this attribute is `"true"` then the hierarchy is not available in this subject area. The default is `"false"`. If all hierarchies of a dimension are hidden, then the dimension itself will be hidden within the subject area. If you hide a dimension in this manner, DeepSee also hides any calculated members that you might place in the same dimension.

B.7 <level>

Within a subject area, a `<level>` element hides or customizes a level. This element has the following contents:

Attribute	Purpose
`name`, `displayName`, `description`, `disabled`	See "Common Attributes in a Subject Area" earlier in this appendix.
`hidden`	(Optional) If this attribute is `"true"` then the level is not visible in this subject area. The default is `"false"`.
`sort`	(Optional) For a level in a time dimension, this specifies how to sort members of this level by default. Specify either `"asc"` or `"desc"` This attribute has no effect for levels in data or age dimensions.

B.8 <listing>

Within a subject area, a `<listing>` element hides, customizes, or adds a listing. This element has the following contents:

Attribute	Purpose
`name`, `displayName`, `description`, `disabled`	See "Common Attributes in a Subject Area" earlier in this appendix.
`hidden`	(Optional) If this attribute is `"true"` then the listing is not visible in this subject area. The default is `"false"`.
other attributes	(Optional) See "<listing>" in the previous appendix. Specify these only if you are redefining a listing or adding a new listing.

B.9 <calculatedMember>

Within a subject area, a `<calculatedMember>` element hides or customizes a calculated member.

Attribute	Purpose
`name`, `displayName`, `description`, `disabled`	See "Common Attributes in a Subject Area" earlier in this appendix.
`dimension`	Dimension to which this member belongs.
`hidden`	(Optional) If this attribute is `"true"` then the calculated member is not visible in this subject area. The default is `"false"`.
other attributes	(Optional) See "<calculatedMember>" in the previous appendix. Specify these only if you are redefining or adding a calculated member.

B.10 <namedSet>

Within a subject area, a `<namedSet>` element hides or customizes a named set. This element has the following contents:

Attribute	Purpose
`name`, `displayName`, `description`, `disabled`	See "Common Attributes in a Subject Area" earlier in this appendix.
`hidden`	(Optional) If this attribute is `"true"` then the named set is not visible in this subject area. The default is `"false"`.
other attributes	(Optional) See "<calculatedMember>" in the previous appendix. Specify these only if you are redefining or adding a named set.

C

Details for the Fact and Dimension Tables

Internally, DeepSee uses a *star schema*, which consists of the fact table and dimension tables, which the fact table refers. Note that dimension tables are also called *star tables*. You can directly query these tables, which DeepSee populates when you build a cube, and which DeepSee updates when changes occur. This chapter discusses the following:

- Fact table

- Dimension tables

Important: Do not redefine these tables or attempt to write data to them except via the authorized interfaces; see the chapter "Keeping the Cubes Current" in the *DeepSee Implementation Guide*.

When you compile a cube definition class, DeepSee also generates a table name `Listing`, in the same package as the fact table. Do not use the `Listing` table, which is for internal use only.

C.1 Fact Table

When you compile a cube definition class, DeepSee generates the corresponding fact table with the name *Package_CubeClass*`.Fact` where *Package_CubeClass* corresponds to the package and class of the cube definition, following the usual rules for translating package and class name to table names.

DeepSee updates the fact table whenever you build the cube or update it incrementally.

It is useful to examine this table, as well as the class definition that uses it.

The fact table contains the following fields:

- `ID` — ID of this row, assigned when the row is created.

- `%dspartition` — For future use. Ignore this field.

- `%sourceID` — The ID of the record in the base class on which this record is based. This field is a pointer to the source table.

- One field for each measure, to contain this measure value for this record. (There is one exception; see "Reuse of a Source Property in the Fact Table," later in this appendix.)

  This field stores the actual measure values. The fact table class defines bitslice indices on these values. For example:

```
/// Index for measure M1.
Index MxAge On MxAge [ Type = bitslice ];


/// Measure: MxAge <br/>
/// Source: Age
Property MxAge As %Integer;
```

- One field for each level, to indicate the level member to which this record belongs. (There is one exception; see "Reuse of a Source Property in the Fact Table," later in this appendix.)

  - For levels in data dimensions, the fact table stores the ID of the level member, which is a pointer to a table that defines the members.

    The fact table class defines bitmap indices on these values. For example:

    ```
    /// Index for fact 1. Index DxGender On DxGender [ Type = bitmap ];

    /// Dimension: DxGender <br/>
    /// Source: Gender
    Property DxGender As Test.TestCube.DxGender;
    ```

    If this dimension is shared from another cube, then this pointer refers to a record in the applicable dimension table for that other cube. For information on shared dimensions, see the *Advanced DeepSee Modeling Guide*.

  - For a time- or age-type level, the fact table stores an integer.

    The fact table class defines the corresponding property as a computed value and defines a bitmap index for it. For example:

    ```
    /// Index for fact 4. Index DxBirthDateFxYear On DxBirthDateFxYear [ Type = bitmap ];

    /// Dimension: DxBirthDateFxMonthYear<br/>
    /// Source: BirthDate Property DxBirthDateFxMonthYear As %Integer [ details omitted ];
    Property DxBirthDateFxYear As %Integer [ Calculated, SqlComputeCode = ... , SqlComputed ];
    ```

  - For a cube-to-cube `<relationship>`, if this cube specifies the `sourceProperty` or `sourceExpression` for the relationship, the fact table stores the ID of the corresponding row in the other fact table. That ID is a pointer to the other fact table.

- An additional field for each time-type or age-type dimension, to contain the complete value of the time or age dimension for this record. (There is one exception; see "Reuse of a Source Property in the Fact Table," later in this appendix.)

  This value is in the form %DeepSee.Datatype.dateTime and is not indexed.

The class does not include properties for any iKnow levels or iKnow measures; those are handled differently.

If you specified the `dependsOn` attribute for any levels, the fact table contains additional indices on the level combinations:

```
Index DxPostalCodeViaHomeCityANDDx2642257510 On (DxPostalCodeViaHomeCity, Dx2642257510) [ Type = bitmap
 ];
```

If you defined any `<index>` elements in the `<cube>`, the fact table contains additional, custom indices. For example:

```
Index %UserIndexName On (MxAge, DxGender) [ Type = bitmap ];
```

These custom indices are for your own use; DeepSee does not use them.

C.1.1 Field Names

The following table summarizes how DeepSee determines the names for the measure, level, and dimension fields (in the case where you do not specify the `factName` attribute):

Item and Scenario	Field Name in Fact Table (If Not Overridden by factName attribute)	Example
Measure based on a source property	Mx*prop_name* where *prop_name* is the name of the property.	`MxAge`
Measure based on a source property in another table, via dot syntax (rare)	Mx*other_prop_name*Via*prop_name* where *other_prop_name* is the name of the property in the other class. However, if the resulting field name would be too long, DeepSee generates a unique number.	`MxAgeViaOtherTable`
Measure based on a source expression	Mx*nnnnnnnnn*T where *nnnnnnnnn* is an integer and *T* indicates the measure type. (For example, `I` represents an integer measure.)	`Mx1968652733I`
Data level based on a source property (with no range expression)	Dx*prop_name*	`DxGender`
Data level based on a source property with a range expression	Dx*prop_name*Rg*nnnnnnnnnn* where *nnnnnnnnn* is an integer.	`DxAgeRg855025875`
Data level based on a source property in another table, via dot syntax	Dx*other_prop_name*Via*prop_name* However, if the resulting field name would be too long, DeepSee generates a unique number.	`DxPostalCodeViaHomeCity`
Data level based on a source expression	Dx*nnnnnnnnnn*	`Dx2163088627`
Time- or age-type dimension	Dx*dim_name* where *dim_name* is the name of the dimension. (This field is for use by levels in this dimension.)	`DxBirthDate`
Time- or age-type level	Dx*dim_name*Fx*func_name*, where *func_name* is the name specified in the `timeFunction` attribute for this level.	`DxBirthDateFxYear`
Relationship	Rx*generated_name*, where *generated_name* is the name of the source property or a generated name based on the source expression.	`RxMainCity`

C.1.2 Reuse of a Source Property in the Fact Table

If a cube contains multiple measures that use the same property (via `sourceProperty`), the fact table contains a field for only one of those measures (the last of those measures found in the cube definition). For example, suppose that the cube contained the following measure definitions:

```
<measure name="Age" sourceProperty="Age" aggregate="SUM" factName="AgeFact"/>
<measure name="Avg Age" sourceProperty="Age" aggregate="AVG" factName="AvgAgeFact"/>
```

These two measures are different only by how they aggregate across *multiple* records; the fact table would contain the same value for these measures, for any given record. For efficiency, the fact table would include the field `AvgAgeFact`, but not the field `AgeFact`.

The same logic applies when a cube contains multiple levels that use the same property. The logic also applies when a cube contains multiple age or time dimensions that use the same property.

For a given measure, level, or dimension, if you use `sourceExpression` and access the property via `%source.`*propertyname*, DeepSee always generates a separate field for that value.

C.2 Dimension Tables

When you compile a cube definition class, DeepSee also generates a table for each level, other than the age- and time-type levels. These tables are in the same package as the fact table.

DeepSee updates the dimension tables (also called *star tables*) whenever you build the cube or update it incrementally.

The dimension table for a level contains one row for each member of that level. The dimension tables are created dynamically as DeepSee processes records in the base table. For a given level, each time a new unique value is discovered, that value is added as a new row to the appropriate dimension table. This means that DeepSee automatically adds rows to the dimension tables when needed; no intervention is required.

C.2.1 Name of Dimension Table

If the cube definition specifies the `factName` attribute for the corresponding level, the applicable dimension table uses that value as its name.

Otherwise, the name of the dimension table has the following form:

```
Stargenerated_name
```

Where `generated_name` is the corresponding field name in the fact table, without the `Dx` at the start. For example, suppose that in the fact table, the field name for the `Home City` level is `DxPostalCodeViaHomeCity`. In this case, the corresponding dimension table is named `StarPostalCodeViaHomeCity`.

C.2.2 Columns in a Dimension Table

The columns in this row are as follows:

- `ID` — ID of this row, assigned when the row is created.

- One column to contain the key for this member. The field name is the same as the column name field in the fact table that corresponds to this level; see the previous section.

- One column for each property of this level, to contain the actual property value for this member.

  The field name starts with `Dx` and is based either on the source property name or is generated as a unique number, as described earlier.

  If you use `linkClass` and `linkProperty` to define both the property and the level to which it belongs, the field names for the property and the level would be identical. In this scenario, DeepSee appends `_Link` to the end of the field name for the property.

- One column for the parent level of this level, to contain the ID of the parent of this member.

  The field name is the same as the column name in the fact table that corresponds to the parent level.

Depending on how the level is defined, you can find the member names as follows:

- By default, the key is used as the name, and the name is not stored separately.

- If the level includes a property that is defined with `isName="true"`, then the member names are stored in the column that contains that property (with one exception). The exception is when the property is also defined with `isReference="true"`; in this case, the field is computed at runtime.