Ariba Analysis
Customization Guide

Version 3.0
May 2004

Table of Contents

Preface

The *Ariba Analysis Customization Guide* describes how you can tailor Ariba Analysis to meet the specific analytical needs of your company to best manage the effectiveness of its spend.

This preface includes the following topics:

Audience and Prerequisites

The intended audience of this book is anyone who customizes Ariba Analysis:

- Data warehouse designers
- Database or Java programmers

This document assumes that you have customized Ariba Buyer. You should at least be familiar with Ariba Buyer or be able to confer with those who customized it. Customizing Ariba Analysis requires skills similar to those needed for customizing Ariba Buyer:

- Familiarity with XML
- Knowledge of Java programming
- Understanding of database concepts, such as indexes and lookup keys

You also need an aptitude for understanding the concepts underlying Online Analytical Processing (OLAP).

This book is **not** a replacement for the documentation from the supplier of your Web, application, or database server. The book assumes that you already have such documentation or know how to obtain it.

Understanding Ariba Analysis

Implementing Ariba Analysis may involve many people:

- System administrators to maintain the Ariba Analysis computers and software
- Database administrators to make sure the databases powering Ariba Analysis run smoothly
- Data warehouse designers, whose task is to make the components of your company's analytical reports useful
- Business analysts and managers, the end-users of Ariba Analysis

Before customizing Ariba Analysis, you must understand the users of this software, their goals, and their needs. One prerequisite to this understanding is becoming familiar with how Ariba Analysis itself works: how to create an analytical report and use it to investigate spend data. At a minimum, you should look over the end-user documentation to gain this familiarity. This book assumes that you know the definitions and concepts presented in the following:

- The Ariba Analysis Quick Tours
- The printed book *Ariba Analysis Advanced User Guide*

Ariba Analysis Documentation

The Ariba Analysis documentation set contains the following books.

Title	Audience	Purpose and contents
Ariba Analysis Installation Guide	System and database administrators	**Planning and installing Ariba Analysis** • Architectural overview, software components, deployment configurations • Step-by-step installation
Ariba Analysis Configuration Guide	System and database administrators	**Configuring and administering Ariba Analysis** • Configuration files, command reference, parameters, administration console • System security • Managing users and reports
Ariba Analysis Customization Guide	Systems integrators and data warehouse designers	**Customizing Ariba Analysis** • Design goals, methodologies, and best practices • Working with and extending the data model. Adding facts, measures, dimensions, and materialized views. • Controlling data visibility • Tailoring the user interface
Ariba Analysis Data Load Guide	Data warehouse designers and database administrators	**Loading data from Ariba applications and external systems into Ariba Analysis** • Configuration files • Data-loading metadata XML • command reference
Ariba Analysis Advanced User Guide	Procurement and sourcing business analysts, systems integrators	**Setting-up Ariba Analysis for daily use** • Ariba Analysis prepackaged report models • Designing compound reports, multi-source reports, and template dashboards by role • Customizing Microsoft Excel templates for use with Ariba Analysis • Overview to Ariba integrated Supplier Performance Management

Typographic Conventions

The following table describes the typographic conventions used in this document:

Typeface or Symbol	Meaning	Example
AaBbCc123	Text you need to change is italicized.	`http://server:port/app/inspector`
AaBbCc123	The names of user interface controls, menus, and menu items.	Choose **Edit** from the **File** menu.
`AaBbCc123`	Files and directory names, parameters, fields in CSV files, command lines, and code examples.	There is one line in `ReportMeta.csv` for each report in the system.
AaBbCc123	The names of books.	For more information, see the *Ariba Analysis Customization Guide*.

Ariba Technical Support

For assistance with Ariba products, Ariba Technical Support is available by phone, email, or over the Web. For information on how to contact Ariba Technical Support, refer to the following page on the Ariba Technical Support Website:

http://connect.ariba.com/TechSupport_Contacting.htm

Chapter 1
Overview to Customizing Ariba Analysis

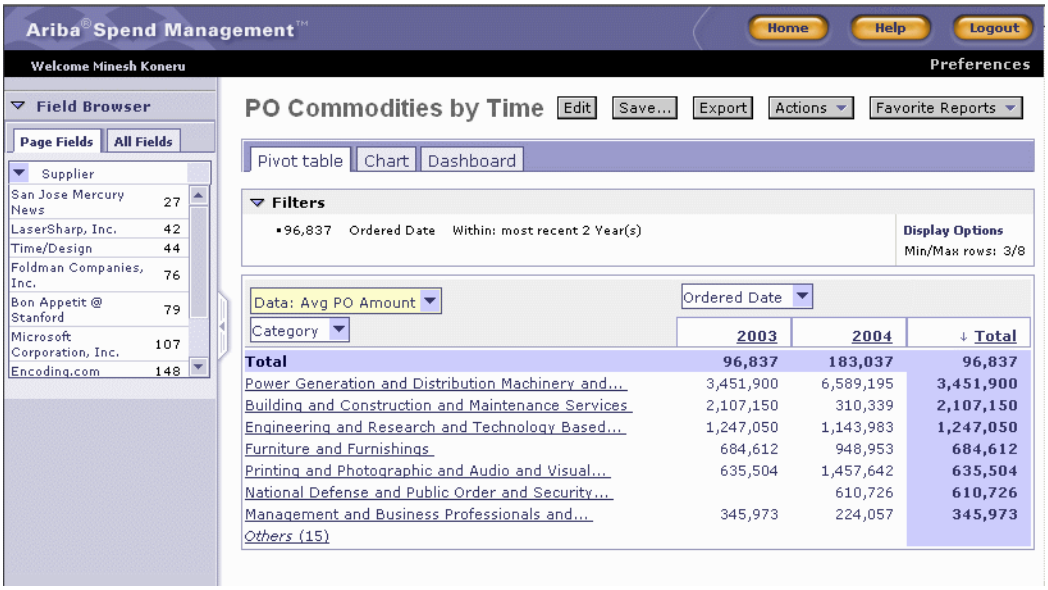
This chapter provides a high-level overview of Ariba Analysis as a background for how to customize it:

Ariba Analysis Basics

Ariba Analysis gives you the power to base buying decisions on your own intelligent, self-customized analyses of data from other Ariba products and Enterprise Resource Planning (ERP) systems. The goal of Ariba Analysis is to improve your ability to see spending and other business patterns to identify opportunities to decrease costs. It is also a tool for examining purchasing processes themselves for possible areas of improvement to save money.

If you are a data warehouse designer or implementer, you create special purpose data objects in Ariba Analysis (called *facts)* that represent business-specific spend data, such as purchase orders, contracts, invoices, and expense reports. Facts are aggregated, summed, rolled-up, and specialized in many ways.

If you are a manager or business analyst, you use Ariba Analysis facts to create analytical reports and graphs, such as pie charts, bar charts, and pivot tables, with which you can investigate business patterns.



You can manipulate your reports in many ways to reveal or hide data or view it at different levels. You can export data to spreadsheet programs, save reports on a *dashboard* for instant viewing, share them with other users, and much more.

OLAP Background

Ariba Analysis is an Online Analytical Processing (OLAP) application for decision support. This section discusses key concepts about OLAP in relation to the databases of both Ariba Analysis and Ariba Buyer. It also introduces some common OLAP terminology in the context of a user's view of Ariba Analysis.

Transactional vs. Analytical Data

This discussion of the nature of transactional and analytical data contrasts the differing goals of Ariba Buyer and Ariba Analysis.

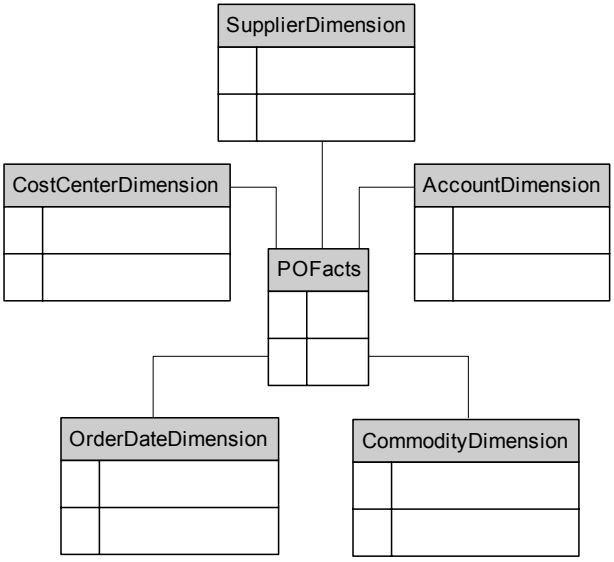
Ariba Buyer is a transactional system for recording all factual aspects of a company's spend: requisitions, purchase orders, contracts, invoices, and their associated line items. In database terminology, these facts are arranged in a *schema*. A schema is a collection of database objects that include records (or *rows*), tables, views, indexes, and synonyms. The records for a particular type of information about your spend are stored in discrete tables. For example, purchase orders are in the PO table, and expense reports in the ER table.

In a transactional schema, the individuality of the data is most important. The **discreteness** of the transactional details is essential for conducting critical business functions and for the purposes of auditing. This very discreteness, however, makes it difficult to generalize the data to achieve a broader understanding of trends and relationships. The sheer number of details hides the patterns in those details, because relationships among the data are not of paramount concern.

In Ariba Analysis, however, **interrelations** among the data are the primary focus. For this reason, the traditional schema for transactional data is not adequate for investigating the data in the many ways it can be viewed. For such investigation, a different kind of schema is needed.

Star Schema Basics

Ariba Analysis is built on *star schemas*. A star schema a simple data warehousing schema. It is called a "star schema" because a diagram of it resembles a star, with points radiating from a center. At the center of the star is a *fact table*, and the points of the star are *dimension tables*.



One or more fact tables contain the primary information in the data warehouse. The dimension tables store information about particular attributes in a fact table. A typical fact table contains *dimension references* (or *keys*) and *measures*. For example, a simple fact table such as **PO Facts** might contain the measure **Amount** and references to dimensions **Time**, **Supplier**, **Commodity**, **Cost Center**, and **Account**. In this example, the keys correspond to dimension tables for **OrderDate**, **Supplier**, **Commodity**, **Cost Center**, and **Account**.

Almost all database queries in Ariba Analysis are star queries. A *star query* is a join between a fact table and dimension tables. Each dimension table is joined to the fact table via a primary-key-to-foreign-key join called a *lookup key.*

In Ariba Analysis, *materialized views* pre-compute and store aggregated data, which are referred to as *measures* or *roll-ups,* such as the sum of sales. Materialized views eliminate the overhead associated with the expensive joins or aggregations characteristic of star queries and so are a key factor in performance. Ariba Analysis

implements its underlying database's architectural feature that most closely approximates materialized views. For example, in the case of IBM DB2, Ariba Analysis creates summary tables. In the case of Oracle, Ariba Analysis creates true materialized views.

User View of Ariba Analysis

This section restates the OLAP terminology from the preceding section in the context of how a user of Ariba Analysis investigates spend data.

An *analytical report* is the intelligent arrangement of *facts*, *fields*, *dimensions*, and *hierarchies* to reveal aspects of the data that might otherwise be hidden.

The people implementing Ariba Analysis have organized your company's spend data into useful information groupings. These groupings are called *facts*, *fields, dimensions,* and *hierarchies*. Although your company may have specific custom spend data not described here, these groupings are common to almost all implementations of Ariba Analysis.

Facts

Facts are the basis for all data investigation. They are the basic transactions you are investigating. Ariba Analysis has the following predefined facts:

- Purchase orders and PO delivery information: commodity purchasing patterns
- Expense reports, expense report lines and violations: travel expense spending patterns
- Invoices, invoice lines and exceptions: supplier billing patterns
- Contracts and contract clauses: agreements established with suppliers from Ariba Contract Workbench
- Requests: a general term for employee requisitions, travel requests, and similar data, including collaborative requisitions
- User activity: Ariba Buyer usage and information
- Projects: Ariba Category Management- and Ariba Contract Workbench-related facts
  - Contract projects
  - Sourcing projects
  - Supplier Performance Management projects and tasks:
- RFX summaries and awards: Ariba Enterprise Sourcing requests for proposal or information

- Surveys and scorecards Ariba Enterprise Sourcing supplier evaluation data
- Temporary labor and time sheets: services spend information
- Proposals

Data fields (or measures)

Data fields (also called *measures*) are numerical data or calculations or aggregations of numerical data. In using Ariba Analysis, you place fields into the *data fields* of a *pivot table*. There are two main kinds of data fields:

1. *Pre-defined fields*: these come with Ariba Analysis, having been created in advance for you.

The following are examples of some common pre-defined fields:

- Amounts
- Quantities
- Counts of number of lines of purchase orders, expense reports, invoices, or requests

2. *User-defined fields* you create yourself.

In addition, you can create your own *user-defined fields*, which are calculations based on other fields.

Dimensions and hierarchies

A *dimension* is an aspect of a fact that is of some interest and has some useful purpose in an analytical report. For example, **Supplier** and **Commodity** are dimensions of purchase orders and are predefined in Ariba Analysis. Similarly, there are predefined dimensions for expense reports, invoices, requests, contracts, projects, and so forth.

A *hierarchy* imposes structure on an entire collection of *levels* of data of a dimension. For example, within the **Management** hierarchy, there are many levels: the first level (top executives), the second (middle managers), the third (line managers), and so on. These hierarchical levels in the **Management** dimension are referred to as *L1*, *L2*, *L3*, and so forth.

Pivot table

The act of creating an analytical report is to arrange facts, fields, dimensions, and hierarchies into a *pivot table*, which is a spreadsheet-like structure of rows and columns that allows you to operate on the data to view it in different ways, such as:

- Drill-down to the next level
- Expand the view
- Hide some of the data to reveal others

A pivot table has something that a common spreadsheet does not: *page fields*. Page fields act like filters on all of the other fields in a report.

Charting in Ariba Analysis

The pivot table, although informative in itself, is also the basis for the usually desired end-products of an analytical report: graphs in the form of pie, bar, other charts.

Note: The purpose of charting in Ariba Analysis is to preview the graphics you want to prepare for presentations, not to produce those graphics. Once you have an idea of the general appearance, export your data to a spreadsheet application for the actual preparation of the presentation graphics.

Common OLAP Verbs

A useful approach to your design work with Ariba Analysis is to think in terms of *verbs*, or common actions in the use of an Ariba Analysis pie chart, bar chart, and pivot table. Verbs can help you define what users want to do with the data structures you create in Ariba Analysis. Employing a verb to describe a customer requirement can characterize the data structures you need to create. For instance, if your users say they want to “expand the view” or “drill down,” the need for a hierarchy is apparent.

Pivot table operations on data include the following.

Operation	Explanation
Expand	View more data by exposing the next lower level of a hierarchy. Opposite of *collapse*.
Collapse	View less data by hiding a level in a hierarchy. Opposite of *expand*.
Drill down	Show finer detail by constraining the data to the next level of a hierarchy.
Move left or right or up or down	Change the column or row position of a hierarchy.
Show Field On	Move the hierarchy to a row, column, or page field.
Hide Value or Column or Row	Eliminate a hierarchy from the current view of the pivot table.
Select Level	Select a specific level in a hierarchy. This is also sometimes called *slicing*. Thus, a level in a hierarchy is a *slice*. Slicing is also sometimes called *constraining*.
Select Others	Select a set of values to display. This is also sometimes called *slicing* and *dicing* or *constraining*.
Sort	Sequence the values in ascending or descending order.
Search	Find matching values in the data.

Ariba Analysis Data Model

The Ariba Analysis data model includes basic definitions of fact tables and the dimension tables associated with them. Ariba Analysis metadata XML is the mechanism to express this data model.

Ariba Analysis metadata XML is similar in concept to metadata XML in Ariba Buyer and in fact reuses it extensively. Ariba Analysis comes with metadata XML files that define the basic star schemas provided with the product.

Using extensions to metadata XML, you can modify these star schemas to accommodate specific business needs. You can also create new schemas.

The following sections of this book discuss metadata XML:

- For approaches to working with metadata XML, see “Ariba Analysis Metadata XML” on page 27.

- For details of the formal syntax of metadata XML, see “Ariba Analysis Metadata XML” on page 77
- For a description of the sample configuration of Ariba Analysis that comes with the product for you to study, see “Practical Design Approaches and Tools” on page 22.
- For a comprehensive example of how to add a new dimension to Ariba Analysis, see “Extended Example: Supplier Risk” on page 61.

Data Loading

Describing your data with metadata XML is only one step in implementing Ariba Analysis. You must also load the actual data. Working with data loading can be divided into two general tasks:

1. Defining the data to be loaded, its sources, its transformations, and the mechanism to load it.
2. Actually loading the data, which is a recurring activity that must be monitored and administered.

You define data loading structures and processes by using XML expressly for this purpose, which is referred to as *data-loading XML*.

For practical considerations in determining the extent and characteristics of the data loading task, how to create or extend data loading definitions, data-loading configuration files, how to execute data loading events, and the formal definition of all elements of data-loading XML, see “Data Sources” on page 21, “Sample Customer Data” on page 21, and the *Ariba Analysis Data Load Guide*.

- For an example of data loading in conjunction with adding a new dimension, see “Loading Supplier Risk Data” on page 63.

Approaches to Customization

This section presents some general considerations for approaches to customizing Ariba Analysis and some practical background for understanding how to proceed:

- “Design Considerations” on page 20
- “Practical Design Approaches and Tools” on page 22

Design Considerations

Implementing Ariba Analysis involves several determinations you must make:

User Requirements

The single most important activity before you begin your design is to determine what your users need and how they want to see their data. This task is not necessarily a technical challenge but calls for communication skills.

As stated in "Common OLAP Verbs" on page 17, a fruitful exercise can be to work with your customers to help them express in terms of **verbs** (common operations to manipulate the data) exactly what they want to do with the data.

Organizational Hierarchies

One obvious source of design information is the hierarchies of the organization for whom you are implementing Ariba Analysis. These include:

- "Org chart" hierarchies, like managerial reporting relationships
- Accounting hierarchies, such as cost centers, operating companies, and general ledger structures, such as granularity on cost of goods sold
- Geographical distributions

Ariba Buyer Implementation

If Ariba Buyer has been customized to a great extent, you will need to mirror those customizations in Ariba Analysis.

For example, if you have made extensive modifications of or additions to the structure of purchase orders or related information, such as requisitions, you need to make those same structural changes in Ariba Analysis.

Data Sources

A helpful way to plan your data loading is to list the sources of data that need to be analyzed:

- Types of data
- Systems on which the data resides: whether Ariba Buyer or an ERP system

Sample Customer Data

Gathering sample data like the following from your customers can help you make design decisions.

Data	Description	Example
Source of Data	Identifier assigned to your systems and locations to recognize the data source	`ariba3`
Business Unit	Code representing your organizational structure	Employer Services
Division	Code representing your organizational structure	EBS
Buying Location	Code representing the location that actually made the purchase	San Diego
Sourcing Family	Definition of the category or commodity family being purchased	Facilities
Sourcing Group	Definition of the sub-category/commodity being purchased	Maintenance
Sourcing Group Description	Detailed description of the sub-category/commodity being purchased	General contractors
Vendor Number	Code identifying vendor	8972
Vendor Name	Name of vendor	Safety Kleen Corporation
Annual Spend Last Year in Millions	Total dollar spend last year with this particular supplier by this location/organization	2.45
Annual Spend Current Year in Millions	Total annualized dollar spend this year with this particular supplier by this location/organization (based on actual spend or estimate for the entire year)	1.75

Practical Design Approaches and Tools

This section describes practical considerations for customizing Ariba Analysis:

Designing for Performance

Critical to the performance of Ariba Analysis is how you design your dimensions, hierarchies, and materialized views. Aggregating the lowest level of detail not only affects performance but also impacts users' view of the data. For an example, see "Concepts: Designing for Performance" on page 27.

Tools for Visualizing the Data Model

This section describes tools to help design or test your work in Ariba Analysis, such as software that lets you see the Ariba Analysis data model.

The Administration Console Schema Browser

Ariba Analysis has an administration console that includes a useful tool called the *Schema Browser* for looking at relationships among facts, dimensions, and their related tasks. It provides a direct view into these relationships as represented in Ariba Analysis metadata XML.

The Schema Browser is similar to the Inspector but pertains only to the interrelationships among Ariba Analysis OLAP structures, whereas the Inspector deals with objects instantiated in the Ariba Analysis database.

See the *Ariba Analysis Configuration Guide* for details about this tool.

The Inspector

With the Inspector you can examine values in the database, reset those values, and see relationships among data objects.

By default, you access the Inspector with the following URL. The Inspector may be protected. See the *Ariba Analysis Configuration Guide* for more information.

`http://<yourServer>:<yourPort>/Analysis/inspector`

To view relationships among the metadata definitions you create, under the heading **Metadata**, click **Submit Query**. The Inspector displays various aspects of Ariba Analysis under different headings, known as *modules*. Of primary interest to you as a designer are the following modules:

```
ariba.analytics.core.Analytics
ariba.analytics.core.Core
```

For instance, in designing a custom access control manager (discussed in "Custom Java for Access Control" on page 70), you might find it helpful to study the cluster root for the `User` object, which is viewable under `ariba.analytics.core.Core`.

Graphical Modeling Tools

This section introduces the Ariba Buyer commands `printinheritance` and `printORdiagram`, which read the object model and describe the inheritance graph and relationships among classes. These are optional tools that can help you to understand the classes and relationships in the Ariba Buyer.

Viewing the Output

The `printinheritance` and `printORdiagram` commands produce ASCII files. To produce a graphical overview of metadata XML relationships, you must use an appropriate graphing program to convert the ASCII output of `printinheritance` and `printORdiagram` into a tree graph.

One graphing program you can use is `VCG` (Visualization for Compiler Graphs), which is freeware. To download `VCG`, go to:

`http://sal.kachinatech.com/E/2/VCG.html`

The printinheritance command

The `printinheritance` command generates an inheritance tree diagram for the Ariba object model in GDL (Graph Description Language) format. It takes arguments to specify whether you are generating the tree from the metadata in the file system or the metadata in the database.

The following instructions describe using printinheritance with VCG.

▼ To use printinheritance:

1. At the top level of your Ariba installation, run printinheritance.

   The command generates the file InheritanceTree.gdl.

2. Run VCG and load InheritanceTree.gdl.

   VCG displays the inheritance tree. To pan around the tree, you can use the a, b, c, and d keys.

The printORdiagram Command

You use printORdiagram to generate an object-relationship (OR) diagram for the Ariba Buyer object model. The syntax of the command is as follows:

```
printORdiagram -classes <classlist> -variants <varName>
```

The -classes switch specifies a colon-separated list of classes to be included in the output. For example, to display a diagram for the Requisition and ExpenseReport classes, you would type the following command (all on one line):

```
printORdiagram -classes
ariba.procure.core.Requisition:ariba.expense.core.ExpenseReport
```

If you omit the -classes switch, the graph displays the entire object model, which can be difficult to read.

To display objects for specific variants only, use the -variants switch. You specify variants with a colon-delimited list of names, such as:

```
-variants Plain:var1
```

The following instructions describe using printORdiagram with VCG.

▼ To use printORdiagram:

1. At the top level of your Ariba installation, run printORdiagram.

   The command generates the file ORDiagram.gdl.

2. Run VCG and load ORDiagram.gdl.

In the graph, each node displays the class name and attribute list. Attributes are displayed according to whether they are intrinsics, indicated by a trailing (I), or extrinsics, indicated by a trailing (E).

Complete Design Example

Chapter 3, "Extended Example: Supplier Risk," is a representative example of a design for customizing Ariba Analysis and implementing that design. If you are familiar with how Ariba products work, you might read that chapter as a quick tutorial.

Report and Excel Templates Models

The *Ariba Analysis Advanced User Guide* describes many useful arrangements of data fields and dimensions that are delivered as models with Ariba Analysis. The analytical reports and Excel templates are delivered with Ariba Analysis as a basis for your own customization or adaptation of them for your particular customer's needs. They can be helpful in designing your customers' spend investigations.

Chapter 2

Ariba Analysis Metadata XML

With metadata XML you can extend the Ariba Analysis data model to meet customer needs. You can also use metadata XML to control the visibility of data. This chapter contains the following sections:

Appendix A, "Ariba Analysis Metadata XML Reference," contains formal definitions of all metadata XML elements.

"Extended Example: Supplier Risk" on page 61 is a comprehensive example of using metadata XML to achieve a specific design goal in customizing Ariba Analysis.

Concepts: Designing for Performance

The key to acceptable performance in Ariba Analysis is to organize your data into hierarchies and materialized views.

Consider the sheer amount of data in the following example. Each fact, such as Purchase Order line items, has multiple measures and values: Amount and Line Item Count. In addition, POs have these dimensions:

- Ordered Date
- Supplier
- Commodity

- Department (Cost Center)
- Account Code (GL entry)
- Requester (User)

To meaningfully analyze hundreds of thousands of transactions, users need to see aggregated data so they can subtotal (dice) across these dimensions, for example, PO volume by department over time, or spend by commodity and supplier.

However, rolling up data by the lowest level of these dimensions—individual days, suppliers, or cost centers—obscures the big picture: there are too many distinct dimension values for each of these dimensions. For example:

- Thousands of days – Ordered Date
- Tens of thousands of Suppliers
- Hundreds of commodities
- Hundreds of thousands of Departments
- Hundreds of Account Codes (GL entries)
- Hundreds of thousands of Requesters (Users)

A report of hundreds of commodities diced by five hundred cost centers is not a manageable report.

So in your design, strive for a proper set of hierarchies for these dimensions. Otherwise, materialized views for the lowest levels of these dimensions will likely lead to poor performance.

If you find that you are adjusting Ariba Analysis performance-related parameters in Parameters.table, such as the maximum number of rows returned to a pivot table, your hierarchies are almost certainly badly classified.

Working with Metadata XML: The Mechanics

The best way to become familiar with metadata XML is to look at the Ariba Analysis sample files.

This section describes the mechanical aspects of working with metadata XML:

Warning: Do **not** directly modify either the core metadata XML files, such as `SpendAnalysis.aml`, or the sample extension file `SpendExt.aml`, which are reserved for Ariba only. You must extend the data model. See "Extending the Data Model" on page 30.

Core Metadata XML files: SpendAnalysis.aml and Others

Predefined facts, measures, dimensions, and hierarchies are in the basic metadata XML files located in the following directory:

AnalysisServerRoot`/ariba/variants/Plain/extensions/`

These are the core metadata files, one for general structures, such as users, cities, suppliers, and one each fact, such as POs, ERs, contracts, and so forth. Segregation of these structures into separate metadata XML files makes them easier to study and extend or hide.

Note: Do not modify any of these core files. Instead, create extensions to them. See "Extending the Data Model" on page 30.

File Name	Function
`ACMAnalysis.aml`	For analyzing Ariba Category Management data
`ContractAnalysis.aml`	For Ariba Contracts analysis
`ExpenseReportAnalysis.aml`	For Ariba Travel & Expense analysis
`InvoiceAnalysis.aml`	For Ariba Invoice analysis, including invoice exceptions
`PurchaseOrderAnalysis.aml`	For PO analysis
`RequestAnalysis.aml`	For analyzing requisitions

File Name	Function
RFXAnalysis	For analyzing RFXs from Ariba Enterprise Sourcing
SpendAnalysis.aml	Basic analyses: contains user, city, supplier, account, and other common dimensions
SurveyScorcardAnalysis.aml	For analyzing supplier surveys and scorecards from Ariba Enterprise Sourcing
UserActivityAnalysis.aml	For analyzing Ariba Buyer user activity

Sample Extension File: SpendExt.aml

Ariba Analysis comes with a sample extension file, which you can use as a model for your work:

AnalysisServerRoot/config/variants/Plain/extensions/SpendExt.aml

Note: Do not modify SpendExt.aml. Instead, create and extension. See "Extending the Data Model" on page 30

Extending the Data Model

As in Ariba Buyer, you must not modify the core metadata XML definition files directly. Instead you must set up a metadata XML *extension*. Physically, an extension is a file with a name ending in .aml that conventionally resides in the following directory.

AnalysisServerRoot/config/variants/Plain/extensions

In your extension, you must use the <extension> element to state the name of your extension and the path to its .aml file relative to *AnalysisServerRoot*. For example, consider the following <extension> element.

```
<extension name="config.variants.Plain.extensions.MyExtensions">
```

This states that Ariba Analysis can find the file MyExtensions.aml in the directory *AnalysisServerRoot*/config/variants/Plain/extensions.

To define a new object, you must create a module that consists of elements such as <fact>, <dimension>, <hierarchy>, or <level> and their related elements, depending on what kind of object you are creating.

When you want to extend a predefined object, in addition to the <extension> element, the extension must use the <import> and <inModule> elements to specify **which** module it extends. For example:

```
<extension name="config.variants.Plain.extensions.MyExtensions">
<!-- We are extending the basic extension file SpendAnalysis.aml -->
<import extension="ariba.variants.Plain.extensions.SpendAnalysis">
<inModule name="ariba.variants.Plain.extensions.SpendAnalysis">
```

The remainder of the extension module contains elements such as <inFact>, <inDimension>, <inMaterializedView>, and related elements to supersede the definitions that come with Ariba Analysis.

BigDecimal Data Type in Extensions

For performance reasons, Ariba recommends that you use `BigDecimal` as the type for numerical data in your extensions, rather than `double` or `float`.

Server Restart

When you alter the Ariba Analysis data model, you must update the database with your extensions by restarting Ariba Analysis, as described in the *Ariba Analysis Configuration Guide*.

Facts and Measures

Fact tables hold the raw data that you want to analyze. This section includes the following topics:

Ariba-supplied Facts

Ariba Analysis has predefined fact tables for the following:

- Purchase orders and PO delivery information: commodity purchasing patterns
- Expense reports, expense report lines and violations: travel expense spending patterns
- Invoices, invoice lines and exceptions: supplier billing patterns
- Contracts and contract clauses: agreements established with suppliers from Ariba Contract Workbench
- Requests: a general term for employee requisitions, travel requests, and similar data, including collaborative requisitions
- User activity: Ariba Buyer usage and information
- Projects: Ariba Category Management- and Ariba Contract Workbench-related facts
  - Contract projects
  - Sourcing projects
  - Supplier Performance Management projects and tasks:
- RFX summaries and awards: Ariba Enterprise Sourcing requests for proposal or information
- Surveys and scorecards Ariba Enterprise Sourcing supplier evaluation data
- Temporary labor and time sheets: services spend information
- Proposals

Distinctions between Line and Header Facts

Ariba Analysis needs different fact tables at different granularities for different types of investigation. For example, consider invoice exceptions, which can be thought of as "bad" invoices. Ariba Analysis includes facts for Invoice, Invoice Exception Lines, and Invoice Exception Headers. With regards to the data structures, the Invoice fact is at the split accounting level with one record per each split line item. The Invoice Exception facts are at the header level and the line item level.

Ariba Analysis cannot count the number of invoices if the fact table is not at the header level. The invoice count measure on the Invoice fact table (which is at split line level) is a fractional count. For example, with four lines from four invoices that have two lines each, the count of invoices is two: 0.5 + 0.5 + 0.5 +0.5. For a "pure" count of

invoice exceptions, Ariba Analysis needs to store information about the invoice header level. Without modeling the data this way, a customer cannot see a a crucial number: how many invoice lines were bad.

However, with multi-fact analytical reports, you can reconcile these distinctions among facts, when necessary. For more information about multi-fact reports, see the *Ariba Analysis Advanced User Guide*

Location of Predefined Facts and Measures

The Ariba Analysis fact tables are defined in the base metadata XML file `SpendAnalysis.aml` or the other core metadata XML files listed on page 29. Each of the predefined facts comes with a set of useful measures defined in these same files.

Steps for Defining a New Fact

▼ **Defining a new fact consists of three main steps:**

1. You must know how your users want to use the fact table data. What do they want to investigate?
2. Create a metadata XML extension file as described in "Extending the Data Model" on page 30. Use the `SpendAnalysis.aml` and other metadata XML files as templates. They contain definitions of all Ariba-supplied fact tables.
3. In the extension file, define the structure of the fact table with `<fact>`, `<measure>`, and other elements.

   If any of the fact fields are dimension objects, you must make sure to create a dimension class so that field elements in the fact table can refer to it. Creating dimensions is discussed in "Dimensions and Hierarchies" on page 37.

Extending Predefined Facts

▼ **To modify a fact predefined in Ariba Analysis, follow these two general steps:**

1. Create a metadata XML extension file as described in "Extending the Data Model" on page 30. Use the `SpendExt.aml` file as a template. It contains examples of extensions to facts.
2. Use the `<inFact>` element to modify fields or add more fields to the fact table.

For example, to add the definition of a measure to the expense report fact (ERLineItem), use the following metadata XML:

```
<extension name="config.variants.Plain.extensions.MyExtensions">
<!-- We are extending the basic extension file SpendAnalysis.aml -->
<import extension="ariba.variants.Plain.extensions.SpendAnalysis">
<inModule name="ariba.variants.Plain.extensions.SpendAnalysis">
<!-- We are adding a measure to the expense report fact -->
<inFact name="ariba.analytics.fact.ERLineItem">
  <measure name="myMeasure">
    <type class="BigDecimal"/>
    <measureOperator type="count"/>
    <properties label="My Measure"
       description="A count of things"/>
  </measure>
</inFact>
```

Common Elements of a <fact>

In metadata XML, a fact is an object you define with the <fact> element. The value of the <fact> element's name attribute is a fact class name in the ariba.analytics.fact package. For example:

```
<fact name="ariba.analytics.fact.POLineItem">
  ...
</fact>
```

The following sections describe some of the more common metadata XML subelements of a <fact>:

- "<lookupKey>" on page 34
- "<field>" on page 35
- "<measure>" on page 35

<lookupKey>

A <lookupKey> defines a unique key or keys in the fact table and is a way to identify <fact> data for later updating. For example, in the predefined POLineItem fact table, there are several lookup keys. These lookup keys are described with metadata XML as follows:

```
<fact name="ariba.analytics.fact.POLineItem">
  <lookupKey fields="POId,POLineNumber,SplitAccountingNumber,SourceSystem"/>
  ...
```

```
</fact>
```

<field>

A <fact> usually contains at least one <field>. There are two major types of <field>, as defined by their <type> elements:

1. Information fields
2. Dimension objects

Information fields are similar to labels or descriptions: they are to inform users. Each field usually has a simple data type, which is expressed with the <type> element, such as int, String, or Boolean. Example information fields include Description, ReqId (requisition identification number), or POId (purchase order identification number). The following example clearly indicates what the fact in question is: a purchase order identification number.

```
<field name="POId" nullAllowed="false">
  <type class="java.lang.String"/>
  <properties label="PO Id"
    description=”Purchase order identification number”/>
</field>
```

Note: POId is also a <lookupKey>.

The second kind of <field> of a <fact> is a dimension object, which indicates how the data will be analyzed. The <type> of this field points to a dimension class. Because the field depends on its pointed-to dimension, when you define the <field>, you must make to sure to also define the dimension.

Examples of dimension object fields include CostCenter, Commodity, or Supplier, as in the following metadata XML example:

```
<field name="Supplier" nullAllowed="false">
  <type class="ariba.analytics.dimension.Supplier"/>
</field>
```

<measure>

A <measure> of a fact is a field that is aggregated. For example, Amount is a measure. The amount column from the database, when included in various dimensions, is summed along those dimensions.

```
<measure name="Amount" nullAllowed="false">
  <type class="MultiCurrency"/>
  <measureOperator type="sum"/>
</measure>
```

Use the <type> element to specify a measure's data type, which is expressed in terms of a Java data type, such as int, BigDecimal, or MultiCurrency.

Use the <measureOperator> element to define a measure's purpose. The <measureOperator> element specifies the measure's mathematical function: average, sum, or count.

Multiple Currency Measures

Measures of money should have a <type> of MultiCurrency to indicate that their values represent different currencies. For example, all Amount measures in the default configuration are of type MultiCurrency. Ariba Analysis stores multiple values for the field, corresponding to the value in different currencies.

Note: Do not declare money measures as type int. This prevents Ariba Analysis from storing multiple values and makes the measure only single-valued.

Inline Dimensions

You can make an informational fact field act like a dimension or hierarchy, so that it can be placed anywhere in the pivot table: data, row, column, or page fields. *Inline dimensions* are best used when the dimension has low cardinality, such as catalog vs. non-catalog or requisition status.

The metadata XML attribute inlineDimension for the <field> element of a <fact> defines a inline dimension field. For example, the following field definition defines the line type field as an inline dimension:

```
<field name="LineType" nullAllowed="false" inlineDimension="true">
```

Including Dimensions

You can indicate that a dimension must be included with a particular measure when that measure is added to the pivot table. Use the includeDimension attribute on the <measure> element. For example, the following snippet of metadata XML declares that the Part dimension must be included when Quantity measure is exported:

```
<measure name="Quantity" nullAllowed="false"
   includeDimension="Part"
</measure>
```

Excluding Data from Export

You can prevent data from being exported to Excel with the `excludeFromExport` attribute on the `<properties>` element. See "excludeFromExport Attribute" on page 105.

Dimensions and Hierarchies

You need a solid understanding of OLAP dimensions and hierarchies to successfully implement Ariba Analysis. As introduced in "Star Schema Basics" on page 14, a dimension is a generalization of a fact that is of some interest or usefulness in an analytical report. For example, a purchase order has many dimensions: its supplier, its commodities, its requester, its approvers, and so forth.

Similarly, a hierarchy reveals levels of a dimension in some larger organizational scheme that may be useful to investigate. One simple hierarchy, for example, is a management structure: line managers, mid-managers, and executives. A dimension can have an infinite number of hierarchies, which can be characterized by how you want to view the data.

This section includes the following topics:

Location of Predefined Dimensions and Hierarchies

The dimensions and hierarchies that come with Ariba Analysis are defined in the base metadata XML file SpendAnalysis.aml or the other core metadata XML files listed on page 29. Each of the predefined dimensions comes with a set of useful hierarchies defined in these same files.

Common Elements of a <dimension>

In metadata XML, a dimension is an object you define with the <dimension> element. The value of the <dimension> element's name attribute is a dimension class name in the ariba.analytics.dimension package. For example:

```
<dimension name="ariba.analytics.dimension.CostCenter">
   ...
</dimension>
```

Some of the more common metadata XML subelements of a <dimension> are as follows:

- <level>
- <hierarchy>
- <groupLevel>
- <field>

Steps for Defining a New Dimension

▼ **To define a new dimension, follow these general steps:**

1. Develop the characteristics of the dimension.

   A helpful way to determine what dimensions you need is to ask questions about how users want to view the facts. Think in terms of **verbs**, as discussed in "Common OLAP Verbs" on page 17. For example:

   - How do you want to slice and dice the data?
   - Which data have hierarchical levels that should be expandable?

2. Create a metadata XML extension file as described in "Extending the Data Model" on page 30. Use the SpendAnalysis.aml and other metadata XML files as templates. They contain definitions of all Ariba-supplied dimensions.

3. In the extension file, define the structure of the dimension with <dimension>, <field>, <hierarchy>, <level>, <groupLevel>, and other elements.

The following example illustrates some approaches to designing and creating a hierarchy.

Example Dimension with Hierarchies

This section first presents the requirements of an example dimension and then illustrates two methods of creating a hierarchy:

1. With an explicit definition of each <level> of the hierarchy
2. With parent-child relationships specified with <groupLevel>

In this simple example, the designer contemplates a single hierarchy for the CostCenter dimension. She knows that the company wants to explore its purchase order spend and has decided to base her design on the following data:

Fact	POLineItem, the purchase order line item <fact>
Measure	The Amount <measure>
Dimension	The CostCenter <dimension>

She has also learned that the company rolls up cost centers by geographical grouping:

Hierarchy	Country	Plant	Cost Center
Example	USA	Georgia Plant	CC12345

Having understood the characteristics of the dimension and its hierarchy, she proceeds to specify the hierarchy with metadata XML.

Explicit Level Definitions

In the first approach to creating a hierarchy, the designer explicitly defines each of three levels of the geographical hierarchy of the CostCenter dimension. This is correct but somewhat labor intensive, because the designer must add a <level> element for each level.

As shown in bold in the following metadata XML, she uses the <level> element to define the levels: Country, Plant, and CostCenter. She also uses the <hierarchy> element to define the Geography hierarchy that comprises those levels.

```
<dimension name="ariba.analytics.dimension.CostCenter">
  <level name="Country">
    <lookupKey name="CountryName"/>
    <field name="CountryName">
      <type class="java.lang.String"/>
      <properties label="Country"/>
    </field>
  </level>
  <level name="Plant">
    <lookupKey name="PlantName"/>
    <field name="PlantName">
      <type class="java.lang.String"/>
      <properties label="Plant"/>
    </field>
  </level>
  <level name="CostCenter">
    <index fields="CostCenterName"/>
    <lookupKey fields="CostCenterId"/>
    <field name="CostCenterName">
      <type class="java.lang.String"/>
      <properties label="Cost Center"/>
    </field>
    <field name="CostCenterId">
      <type class="java.lang.String" length="50"/>
    </field>
  </level>
  <hierarchy name="Geography" levels="Country,Plant,CostCenter">
    <properties label="Geography"/>
  </hierarchy>
  <overrides>
    <override name="ClassProperties">
    <properties label="Cost Center"/>
    </override>
  </overrides>
</dimension>
```

Parent-Child Relationships with <groupLevel>

With the second approach to creating a hierarchy, the designer's labor is reduced and the metadata XML somewhat simplified. Instead of a <level> element for each level, she uses the <hierarchy> and <groupLevel> elements.

The <groupLevel> element is a way to group levels together by indicating parent-child relationships. Using <groupLevel> is the best approach if the data are already structured in parent-child relationships in the data source.

```
<dimension name="ariba.analytics.dimension.CostCenter">
  <level name="CostCenter">
```

```
      <index fields="CostCenterName"/>
      <lookupKey fields="CostCenterId"/>
      <field name="CostCenterName">
         <type class="java.lang.String"/>
         <properties label="Cost Center"/>
      </field>
      <field name="CostCenterId">
         <type class="java.lang.String" length="50"/>
      </field>
      <field name="GeographyParent">
         <type class="java.lang.String"
   </level>
   <groupLevel name="Geography" maxLevels="3"
      sourceLevel="CostCenterOwnerLevel"
      sourceLevelLookup="CostCenterOwnerId"
      sourceLevelParentLookup="SupervisorId">
      <lookupKey name="GeographyId"/>
      <field name="GeographyName">
         <type class="java.lang.String"/>
         <properties label="Geography"/>
      </field>
      <field name="GeographyId">
         <type class="java.lang.String"/>
      </field>
   </groupLevel>
   <hierarchy name="Geography" levels="Geography,CostCenter">
      <properties label="Geography"/>
   </hierarchy>
   <overrides>
      <override name="ClassProperties">
         <properties label="Cost Center"/>
      </override>
   </overrides>
</dimension>
```

Extending Predefined Dimensions

▼ **To modify a dimension predefined in Ariba Analysis::**

1. Create a metadata XML extension file as described in "Extending the Data Model" on page 30. Use the `SpendExt.aml` file as a template. It contains examples of extensions to dimensions.
2. Use the `<inDimension>`, `<hierarchy>`, and `<level>` elements to modify levels and hierarchies or add more levels and hierarchies to the dimension.

In this example, suppose you want to create a new level called Executive in the existing CostCenter dimension and create a new hierarchy based on that level. Use the <inDimension>, <level>, and <hierarchy> tags to effect this change, as shown in bold.

```
<extension name="config.variants.Plain.extensions.MyExtensions">
<!-- We are extending the basic extension file SpendAnalysis.aml -->
<import extension="ariba.variants.Plain.extensions.SpendAnalysis">
<inModule name="ariba.variants.Plain.extensions.SpendAnalysis">
<inDimension name="ariba.analytics.dimension.CostCenter">
  <level name="Executive">
    <lookupKey fields="Executive"/>
    <field name="Executive">
       <type class="java.lang.String"/>
    </field>
  </level>
  <hierarchy name="Executive" levels="Executive,CostCenter"/>
</inDimension>
```

Another example of extending a dimension is in “Defining the Supplier Risk Hierarchy” on page 62.

Bucketing a Dimension

Having clear, distinct groups or ranges of data can make some data easier to analyze and investigate. You can create a dimension in which a single record holds a range of values, and you can categorize these data into “buckets” or groups you define dynamically with the data themselves. Such a dimension is called a *bucketed dimension*. For example, invoices grouped by Accounts Payable aging ranges is a bucketed dimension.

With Ariba Analysis dimension bucketing, you do not have to manually categorize the data into individual buckets. You only have to define which of the data **represent** the buckets you want to establish, with attributes of the <dimension> element specifically for bucketing. Then, when the data are loaded, they are grouped into the desired buckets automatically.

Note: Bucketed dimensions cannot be versioned. For more information about versioning, see “Slowly Changing Dimensions: Versioning Data Values” on page 46.

▼ To create a bucketed dimension:

1 Decide what buckets you want to create by selecting a field that represents them and specifying that field name with the `bucketSourceValue` attribute.

Note: These records must specify the upper and lower bounds of a bucket (that is, the range of the bucket). Avoid ambiguity in range starts and stops:

Bucket	Ambiguous	Unambiguous
A	0 to 100	0 to 100
B	**100** to 1,000	**101** to 1,000
C	**1,000** to 10,000	**1,001** to 10,000

The starting point for the next range is offset by one from end of the previous range. This leaves no confusion about which bucket a 10,000 or a 1,000 item belongs to.

2 Use the `<dimension>` element's attributes `bucketFlagField`, `bucketRangeMin`, and `bucketRangeMax` to indicate which records represent the buckets and which fields in those records define the ranges of the buckets.

3 Load the data to create the bucketed dimension.

Example of bucketed dimension

A concrete example is the best illustration of a bucketed dimension. This example, which is from the file `SpendAnalysis.aml`, creates a dimension representing a buckets of amount ranges. The fact class `ariba.analytics.fact.POLineItem` contains a field defined as follows:

```
<field name="AmountRange"
  nullAllowed="false"
  bucketSourceValue="Amount"
  bucketName="Request Amount (USD)">
  <type class="ariba.analytics.dimension.AmountRange"/>
</field>
```

The value of the `bucketSourceValue` attribute is the name of another field that defines the buckets.

The following metadata XML defines the AmountRange dimension to represent these data. The dimension has been declared as “bucketable” with the attribute bucketValues set to true. The value of the bucketNameField indicates a field in the dimension that defines the buckets and which in turn point to the actual bucketed data. The value of the bucketFlagField attribute states that the field named IsBucket in the data indicates whether or not a record is a bucket. Those records with isBucket set to true contain fields that define the buckets’ ranges: the RangeMinValue and RangeMaxValue fields.

```
<dimension name="ariba.analytics.dimension.AmountRange"
     bucketValues="true"
     bucketMinField="RangeMinValue"
     bucketMaxField="RangeMaxValue"
     bucketNameField="BucketName"
     bucketFlagField="IsBucket">
  <level name="Interval">
     <lookupKey fields="BucketName,RangeDescription"/>
     <field name="RangeDescription">
        <type class="java.lang.String" length="50"/>
        <properties label="@aml.analysis.SpendAnalysis/a390"/>
     </field>
     <field name="RangeMinValue">
        <type class="java.math.BigDecimal"/>
     </field>
     <field name="RangeMaxValue">
        <type class="java.math.BigDecimal"/>
     </field>
     <field name="BucketName">
        <type class="java.lang.String" length="20"/>
     </field>
     <field name="IsBucket">
        <type class="boolean"/>
     </field>
  </level>
  .
  .
  .
</dimension>
```

Amount Ranges for Non-USD Currencies

The Amount Range dimensions are examples bucketed dimensions. Ariba Analysis has amount ranges for USD and EUR currencies.

For other currencies, you must define new amount ranges and load them into the same AmountRange dimension with a different value for the BucketName, such as PO Amount (JPY). On the fact table you need to add a new dimension of type AmountRange for each foreign currency. An examples of bucketed AmountRange dimensions for the euro are included in the default metadata XML extension file SpendExt.aml.

Non-Rollup Hierarchies

Certain kinds of data cannot be rolled-up (totalled or aggregated) in a meaningful way without introducing errors in calculation of totals or other unwanted effects. Such data can be defined as a *non-rollup hierarchy*. For example, consider a scorecard. The individual lines of a scorecard might be calculated with one formula (say, a 100 point scale), but the totals of a scorecard might be calculated with another formula (including some non-numeric or qualitative factors) to result in a "grade" of either "Pass" or "Fail." In the case of such data, to include the individual line items in the total is not only inaccurate but misleading. The value for the total is usually supplied in the source data itself, not through aggregation.

To define a hierarchy as non-rollup, use the rollupType attribute on the <hierarchy> element. For example, in the default configuration the metadata XML file SurveyScorcard.aml defines the following:

```
<hierarchy name="Question"
  levels="Question"
  rollupType="false">
```

Setting rollupType to false means that the hierarchy cannot be rolled-up in the normal way. Setting it to true (or not specifying the attribute, which is the default) indicates that the hierarchy can be rolled-up correctly.

Dimensions with Multi-Valued Fields

Some facts contain fields that do not have a one-to-one relationship with a dimension. That is, the field relates to more than one dimension field. For example, a particular project might deal with more than one category or more than one organization. Such a dimension is called a *multi-valued dimension*, or a dimension with multi-valued fields.

Note: Use multi-valued fields only when it makes sense for the data you are modeling. Declaring a field as a vector unnecessary can cause performance degradation.

To declare a multi-valued field, set the attribute `vector=true` on the `<type>` element element of the `<field>`.

```
<field name="Region" nullAllowed="fals">
  <type class="ariba.analytics.dimension.Region" vector="true"/>
</field>
```

To load data into multi-valued fields, after the initial data load, simply run more data loading events. However, you must be sure to order the records to be loaded by their lookup keys, because you thus will need to load fewer records and achieve better performance.

Fore information about how to disable an unneeded reference to a multi-valued dimension field, see "<disableVectorDim/>" on page 83.

Slowly Changing Dimensions: Versioning Data Values

A *slowly changing dimension* is one whose field values change over time. For example, consider the Organization hierarchy. At one point, an employee might belong to a certain division. The next year, that same employee might transfer to a different division. With Ariba Analysis support for slowly changing dimensions (also called *versioning*), this employee organizational transfer is recorded in the data.

During data loading, versioned records are stored in staging tables. These staged records must be transferred to the actual dimensions when loading is complete. See "Related Tasks" on page 48.

versioning Attribute

You can version dimensions case-by-case. By default, versioning is not enabled (`false`), except in the demonstration configurations. To enable it, set the following:

```
<properties versioning="true">
```

The `versioning` attribute on the `<properties>` element associated with a `<dimension>` indicates that Ariba Analysis must maintain a historical record of all versions of the dimension's field values for auditing.

Data-loading a Slowly Change Dimension

In `sample/demoConfig/Global/config/sourceTypes/dataLoads/DemoBuyerLoads.xml` the PO load refers to the Contract dimension, which is an example of a slowly changing dimensio. To key the versioning during data loading, an effective date field is specified in the data-load definition:

```
<field name="Contract.ContractId">
  <csvMapping selectField="Contract"/>
</field>
<field name="Contract.VersionEffectiveDate">
  <csvMapping selectField="OrderedDate"/>
</field>
```

The `VersionEffectiveDate` is not a customizable field. This field name is required. However, you can specify either the OrderedDate or ApprovedDate or whatever date you desire to indicate the time that this fact is valid in respect to this slowly chanmging dimension.

Interval between Versioning: Partial Data Loading

The `NoNewSCDRowInterval` parameter in `config/Parameters.table` specifies a time interval in days during which multiple data loading to the same spend dimension does **not** result in versioned data. This allows you to load partial data without creating new versions. For more information about parameters, see the *Ariba Analysis Configuration Guide*.

Bucketed Dimensions: No Versioning

You cannot version a bucketed dimension. For more information about bucketed dimensions, see “Bucketing a Dimension” on page 42.

User View of Versioned Dimensions

Slowly changing dimensions are implicit. You do not need to do anything to see earlier or later versions of the same data field. When you constrain an analytical report to a certain time period, the data values at that time are automatically displayed.

Related Tasks

See the following related tasks in the *Ariba Analysis Data Load Guide*:

- `LoadFromStaging` moves records from staging tables to the actual versioned dimension.
- `PopulateStagingTables` creates the staging tables needced to convert an existing dimension into a versioned dimension.

Materialized Views

This section describes how to work with materialized views. Materialized views are aggregations of measures of a fact table, sometimes called *roll-ups*. A materialized view pre-computes and summarizes these aggregations for maximum speed.

You should create a materialized view for any new fact you define. Make sure that your materialized view contains all measures of the fact and that the dimensions and levels of the data included in a materialized view represent how the data are used in an analytical report.

Database Distinctions

Ariba Analysis implements its underlying database's architectural feature that most closely approximates materialized views. For example, in the case of IBM DB2, Ariba Analysis creates summary tables. In the case of Oracle, Ariba Analysis creates true materialized views.

Common Element of a <materializedView>

In metadata XML, a materialized view is an object you define with the `<materializedView>` element. The value of the `<materializedView>` element's `fact` attribute is the name of a fact class. For example:

```
<materializedView name=”POLineItem” fact="ariba.analytics.fact=”POLineItem">
   ...
</materializedView>
```

The only metadata XML subelement of a `<materializedView>` is `<mvField>`, which indicates the name of a field in the materialized view and the level of its data (if any).

Selecting Levels for the View

Proper selection of levels to include in the materialized view depends on the data. Here are some recommendations about selecting levels.

- Make sure that the number of unique values in a level is not large.

  In general, do **not** include the lowest level of a hierarchy if it has many unique values. For example, if the lowest level of the `Commodity` hierarchy contains thousands of items, it should not be included in the materialized view. On the other hand, if it contains only tens or hundreds, including it in the view may not adversely impact performance.

- Include a sufficient number of levels of a dimension to ensure adequate coverage of hierarchies.

  For example, your `CostCenter` dimension might include two different hierarchies: `Region` and `CompanyCode`. You might think to include the lowest level of each in your view because it is common to both hierarchies. However, that lowest level has too many unique values. Instead, you should include two other higher levels for these hierarchies that are still representative of the uniqueness of the data.

- Prefer the "middle way."

  Assume that for any given hierarchy, if you have more than three levels:

  1. In your materialized view, you probably should not include the lowest level because it most likely contains too many unique values.
  2. You also should probably not include the highest level because it could be too sparse. If you include the highest level and the user drills down to the next lowest level in the hierarchy, your materialized view is no longer used, resulting in poor performance.

  Select a level that is in the middle. A good example is the `Calendar` hierarchy. The highest level is the year, but the predefined materialized view aggregates by month.

After you observe how users actually use Ariba Analysis, such as which levels in the data they access most frequently, you may find it necessary to modify the materialized views you have created. Changing the materialized view may in turn require that you alter the definitions of your dimensions and hierarchies to match those usage patterns.

Performance of Ariba Analysis Queries

The performance of Ariba Analysis depends on materialized views.

Queries that can successfully locate desired data in a materialized view are significantly faster than queries that must retrieve unaggregated data via joins of the various tables of the database or aggregations of the lowest level details of a fact.

A scheduled task that relates to materialized view is `ComputeMatView`, which rebuilds your materialized views and is discussed in the *Ariba Analysis Configuration Guide*.

Location of Predefined Materialized Views

Each fact delivered with Ariba Analysis has at least one corresponding materialized view defined in the base metadata XML file `SpendAnalysis.aml`.

Steps for Defining a New Materialized View

▼ **To define a materialized view.**

1. Determine the materialized views you need, based on an examination of the fact table and usage patterns.
2. Create a metadata XML extension file as described in "Extending the Data Model" on page 30. Use the `SpendAnalysis.aml` and other metadata XML files as templates. They contain definitions of all Ariba-supplied materialized views.
3. In the extension file, define the structure of the materialized view with `<materializedView>` and `<mvField>` elements.
4. After your materialized view is defined, run the scheduled task `ComputeMatView` to construct the materialized view in the database. See the *Ariba Analysis Configuration Guide*.

The following is an example of a materialized view of the `InvoicePO` fact table.

```
<materializedView name="POLineItem" fact="ariba.analytics.fact.POLineItem">
  <!-- These specify the measures to include in the materialized view. -->
  <mvField name="Amount"/>
  <mvField name="LineItemCount"/>
  <mvField name="SplitAccountCount"/>
  <mvField name="POCount"/>
  <!-- These specify the dimensions to include in the view. -->
  <mvField name="OrderedDate"level="Month"/>
  <mvField name="OrderedDate"level="FiscalMonth"
  columnName="OrderedDateFiscal"/>
  <mvField name="CostCenter"level="CostCenter"/>
  <mvField name="Supplier"level="SupplierRank"/>
```

```
  <mvField name="Supplier"level="SupplierType"
  columnName="SupplierType"/>
</materializedView>
```

Extending Predefined Materialized Views

▼ **To extend a materialized view:**

1. Create a metadata XML extension file as described in "Extending the Data Model" on page 30. Use the `SpendExt.aml` file as a template. It contains the examples of extensions to materialized views.
2. Use the `<inMaterializedView>`, `<deleteMVField>`, and `<mvField>` elements to add more aggregations about the fact table in question or delete existing roll-up names so that they can be modified.

For example, the following metadata XML removes the predefined `CostCenter` roll-up from `POLineItem`, so that aggregations can be based on the `Geography` hierarchy described in "Example Dimension with Hierarchies" on page 39.

```
<inMaterializedView name="POLineItem">
  <!-- Remove the originally defined CostCenter roll-up -->
  <deleteMVField name="CostCenter" level="CostCenter"/>
  <!-- roll-up by geographic region -->
  <mvField name="CostCenter" level="Geography" levelNumber="2"
  columnName=GeographicCostCenter/>
</inMaterializedView>
```

Changing User Data Views with Metadata XML

This section discusses how to alter the user view of data and control its visibility with metadata XML under these general headings:

- "Display Group for Line Details" on page 52
- "Changing the Line Details" on page 53
- "Hiding Facts, Measures, or Dimensions" on page 54

This section relies on your familiarity with some concepts and metadata XML tags in Ariba Buyer:

- Displaying fields with `<group>`, `<inGroup>`, and `<groupField>`
- Conditional display of fields

- Display controllers
- The <visibility> tag

Information about these concepts and metadata is in the *Ariba Buyer Customization Guide*.

Display Group for Line Details

One of the most useful changes to the user interface is to alter what the user sees in the detail page, which is the data underlying an analytical report. Ariba Analysis displays the detail page when a user does one of the following on the pivot table view of a report:

- Changes the **Data** field button to **Line-level Details**
- Selects a data field and clicks **Show line-level details**

The metadata XML file SpendAnalysis.aml defines the display group LineLevelDetails for the displaying of detailed line item data in the Ariba Analysis user interface. This section discusses how you can change the display of those details.

Changing the Rank of Hierarchies in the Analysis Wizard

You can use the rank attribute of the <properties> metadata XML element to control the sequence, or rank, of the hierarchies displayed in Step 2 of the Analysis Wizard.

For example, the **Supplier Part** hierarchy of the purchase order fact has a rank value of 5, as shown in this snippet from SpendAnalysis.aml file:

```
<field name="Part" nullAllowed="false">
  <type class="ariba.analytics.dimension.Part"/>
  <properties rank="5"/>
</field>
```

The value of 5 for rank places **Supplier Part** in a position below the top of the list of hierarchies in Step 2 of the Analysis Wizard. Changing the rank of Supplier Part to 0, for example, raises that hierarchy and related hierarchies to the top of the column.

Changing the Line Details

▼ **To alter the display of line-level detail:**

1. Create an extension file (`.aml`) that uses the `<inGroup>` tag and specify the `<inGroupClass>` of the fact you want to alter.
2. Use the `<remove>` and `<properties>` elements to control the appearance of the fields.

The following example changes the line level details for purchase orders:

- The `<remove>` element eliminates the account name from the view.
- A `<groupField>` adds the requisition identification number.

```
<inGroup name="LineLevelDetails_POLineItem">
  <inGroupClass name="ariba.analytics.fact.POLineItem">
    <remove>
       <!-- get rid of account name -->
       <groupField name="Account.AccountName"/>
    </remove>
    <!-- add requisition id -->
    <groupField name="RequisitionId">
       <properties rank="30" label="Req Number/>
    </groupField>
  </inGroupClass>
</inGroup>
```

Controlling Search Levels in Parameterized Reports

Ariba Analysis users can create and use parameterized reports. The key feature of parameterized reports is the ability to pre-constrain an analytical report to specific values or specific levels of a hierarchy. For details about creating or using parameterized reports, see the online help.

With the `searchSubLevels` attribute on a `<hierarchy>` element, you can control how deeply into a hierarchy a user can search to select particular values for a parameterized report. You may want to restrict the user's ability to search only the top level of a hierarchy.

By default, the value of the `searchSubLevels` attribute is `false`. You must specify `searchSubLevels` only if you want to set it to `true`, as shown in the following example of the Management hierarchy:

```
<hierarchy name="Management"
   levels="Management,UserId"
   searchSubLevels="false">
```

For parameterized reports, the initial default value of `searchSubLevels` for a hierarchy are taken from the `SpendAnalysis.aml` and other metadata XML files. The user can override this initial setting, however, in the creation or use of a parameterized report.

At initial installation, the attribute is set true for the following hierarchies:

- Region
- Organization
- Management
- Category

Hiding Facts, Measures, or Dimensions

By default, without customization, a user of the Analysis Wizard can see all facts, measures, and dimensions. With the appropriate metadata XML, however, you can control exactly which facts, measures, and dimensions are visible.

Metadata XML vs. Data Access Control?

You can control the visibility of data in Ariba Analysis in two ways:

1. Basic data visibility with metadata XML, which this section discusses.
2. User-profile-based data access control with the Ariba-supplied access control example or your own custom Java. This type of access control is discussed in Chapter 4, "Examples of Data Access Control".

Metadata XML is a general approach to controlling the basic visibility of data, which you can refine in development with the Ariba-supplied example access control manager or by using your own custom Java, as discussed in Chapter 4, "Examples of Data Access Control.". For example, if there are data fields that you want nobody to see, consider hiding them with metadata XML controls. On the other hand, if some data should be visible to only a specific organization or user (for example), and you are working in development, the example user-profile-based access control would be more appropriate and easier to implement.

Writing Your Own Conditions

With metadata XML, you can control the display of data with the conditions provided with Ariba Analysis or with your own custom conditions. A *condition* is a Java class you write that equates to either `true` or `false`. You specify the class as the value of the `implementation` attribute of the `<condition>` or `<notCondition>` element in the `<visibility>` element of a fact or dimension. See the *Ariba Buyer Customization Guide* for more information. Here are some basic considerations:

- You must import the following packages:

  ```
  ariba.util.core.PropertyTable
  ```

- Your condition must extend from `ariba.base.fields.Condition`.
- You need to implement an `evaluate` method that returns a Boolean value, like the following:

  ```
  public Boolean evaluate (Object value, PropertyTable params)
  ```

- If you need to access the particular user object for which the condition is being evaluated, import these packages:

  ```
  ariba.base.core.Base
  ariba.analytics.core.User
  ```

  To get the details about the user, call the following method:

  ```
  User effectiveUser = (User)Base.getSession().getEffectiveUser();
  ```

Visibility of Facts and Measures

Control the visibility of a fact table with an `<override>` to specify the `<visibility>` condition of the `<fact>` element. Conditions are specified with the `<condition>` or `<notCondition>` elements. If the specified condition evaluates to `false`, the fact table is not displayed in the user interface. To control visibility in the original fact table declaration, use an `<override>` such as the following. This example assumes that you have written the class `config.condition.MyCondition` that evaluates to `false`.

```
<fact name="ariba.analytics.fact.MyNewFact">
  …
  <overrides>
    <override name="ClassProperties">
       <visibility>
          <condition implementation="config.condition.MyCondition"/>
       </visibility>
    </override>
  </overrides>
</fact>
```

Similarly, if you want to apply a condition to a fact in a different extension file from where that fact is originally declared, use <inFact> and <inField>:

```
<inFact name="ariba.analytics.fact.MyNewFact">
  <inField name="ClassProperties">
    <visibility>
       <condition implementation="config.condition.MyCondition"/>
    </visibility>
  </inField>
</inFact>
```

Control the visibility of measures with a <visibility> condition on the measure field itself. For example, suppose you declare a new measure called POCount. You can add a visibility condition like the following in the initial definition of the fact:

```
<measure name="POCount">
…
  <visibility>
    <condition implementation="config.condition.MyCondition"/>
  </visibility>
</measure>
```

The same can be accomplished in a separate extension file with the <inField> tag:

```
<inField name="POCount">
  …
  <visibility>
    <condition implementation="config.condition.MyCondition"/>
  </visibility>
</inField>
```

Visibility of Dimension Fields

For any fact table, you can control the visibility of its associated dimension fields with the a <visibility> condition on the field in the fact table that refers to the dimension in question. For example, suppose you declare a new field called OrderedDate. You can add a visibility condition like the following:

```
<dimension name="config.variants.extensions.MyClasses">
…
  <field name="OrderedDate">
    …
    <visibility>
       <condition implementation="config.condition.MyCondition"/>
    </visibility>
  </field>
…
```

```
</dimension>
```

The <inField> tag allows you to use a separate extension file to control the visibility of field in a dimension:

```
<inDimension name="config.variants.extensions.MyClasses">
…
   <inField name="OrderedDate">
   …
      <visibility>
         <condition implementation="config.condition.MyCondition"/>
      </visibility>
   </inField>
<inDimension>
```

Deleting Entire Hierarchies

If you do not need any fields from a dimension, you can delete the entire dimension with the <deleteHierarchy> element. For example, the following metadata XML removes the Supplier Type hierarchy:

```
<inDimension name="ariba.analytics.dimension.Supplier">
   <deleteHierarchy name="SupplierType"/>
</inDimension>
```

Note: If you delete hierarchies, modify any data-loading definitions that might attempt to load data into them and remove any references to them in the Ariba-supplied task definitions.

Descriptions and Labels in Resource Files

In Ariba's metadata XML files, values that start with the character @ are interpreted as indirect references to strings in resource files. Ariba Analysis reads values from resource files. The XML file specifies the name of the resource file and the name of the key within that resource file. For example:

```
<properties label="@aml.analysis.SpendExt/a001"/>
```

To expand this string, Ariba Analysis appends the extension .csv to the specified resource file name and looks for a resource file with that name in the appropriate locale. In this example, the value of the label attribute is specified with the key a001 in the resource file named aml.analysis.SpendExt.

Note: You must place your own resource files in the appropriate subdirectory in *AnalysisServerRoot*/config/resource/*locale*/strings, depending on the locale of your resources. In addition, you must restart Ariba Analysis for your additions to take effect.

Substituting Values in the Analysis Wizard

You can substitute the field values displayed in the Analysis Wizard with the MapLabelController class for the <properties> element and an associated string resource CSV file. MapLabelController works like a decode method, similar to the Ariba Analysis data loading mapValue's function decode (for example). A value from the database for the specified <field> name is replaced with the value from second column of the resource string file.

Invoke the MapLabelController class with the exact controller attribute shown in the following metadata XML snippet. The name of the file containing the string is a parameter to the class specified with the controllerFile attribute.

```
<properties controller="ariba.analytics.fields.MapLabelController"
     controllerFile="data.RFxType"/>
```

Resource string files must be located in the standard resource directories, in *AnalysisServerRoot*/ariba/resources/*locale*/strings, where *locale* is the locale specified in the Ariba Analysis config/Parameters.table.

Note: Do not specify the .csv extension in the controllerFile attribute.

In this example from RFXAnalysis.aml (one of the default metadata XML files), a record from the database whose Type field contains the value negotiationParallel displays the string Individual Negotiation in the Ariba Analysis user interface.

```
<field name="Type" nullAllowed="false" inlineDimension="true">
  <type class="java.lang.String" length="30"/>
    <properties rank="70"
    label="@aml.analysis.RFXAnalysis/a039"
    description="@aml.analysis.RFXAnalysis/a040"
    controller="ariba.analytics.fields.MapLabelController"
    controllerFile="data.RFxType"/>
```

The corresponding resource string file looks like this (partially) (*AnalysisServerRoot*/ariba/resources/en_US/strings/data.RFxAnalysis.csv) :

```
8859_1,,
english,English
negotiationParallel,Individual Negotiation
negotiationParallelCollective,Collective Negotiation
openRFQ,Open RFQ
```

Chapter 3
Extended Example: Supplier Risk

This chapter is a comprehensive example of customizing Ariba Analysis to add a new hierarchy—supplier risk—to an existing dimension—supplier. Topics are as follows:

The example relies on four hypothetical characters at a fictitious company:

1. The users: persons who want a customization of Ariba Analysis.
2. The business analyst: a person who has a great deal of knowledge about what the users want.
3. The designer: the person who is extending Ariba Analysis to meet the users' requirements.
4. The system administrator: the person who works with the designer to help meet the users' needs.

Characteristics of Supplier Risk

The Ariba Analysis users want to add another aspect to their investigations: *supplier risk*. For example, they can use supplier risk to determine which of their departments might be purchasing goods or services from suppliers whose continued business may be in doubt.

The designer determines that supplier risk has the following characteristics:

- Risk is categorized as high, medium, or low.
- Each supplier's risk rating is reviewed only every half year.

- The risk rating for all suppliers is maintained by a business analyst, who keeps a spreadsheet detailing each approved supplier's name, supplier identification number, and risk rating.
- This same analyst has an Excel template for forecasting risk that would be ideal to store in Ariba Analysis for users to export the data for analysis.

This customization is a straightforward implementation in Ariba Analysis. The necessary tasks are as follows:

1. Define a new hierarchy for the `Supplier` dimension
2. Set up a data load event to load the analyst's spreadsheet data into Ariba Analysis
3. Run the data load event and validate the data

Defining the Supplier Risk Hierarchy

Because Ariba Analysis comes with the predefined `Supplier` dimension, the designer decides to add the hierarchy `Risk` as an extension to it. The designer makes a copy of the sample extension file to begin the definition:

```
cd <CodePlaceHolder>AnalysisServerRoot
cp config/variants/Plain/extensions/SpendExt.aml \
  config/variants/Plain/extensions/SpendExtSupplierRisk.aml
```

The designer then edits `SpendExtSupplierRisk.aml` to add the `Risk` hierarchy and change the extension name:

```
<!DOCTYPE extension SYSTEM "../../../../ariba/analytics/core/extensions.dtd">
<!-- Module name must match path to .aml file -->
<extension name="config.variants.Plain.extensions.SpendExtSupplierRisk">
<import extension="ariba.variants.Plain.extensions.SpendAnalysis"/>
<inModule name="ariba.analytics.core.Analytics">

  <!-- ********** SUPPLIER RISK HIERARCHY ********** -->

  <!-- We use inDimension because this is an extension
  to the existing Supplier dimension -->
  <inDimension name="ariba.analytics.dimension.Supplier">
    <level name="Risk">
       <lookupKey fields="Risk"/>
       <field name="Risk">
          <type class="java.lang.String"/>
       </field>
    </level>
```

```
    <hierarchy name="Risk" levels="Risk,Supplier"/>
  </inDimension>
</inModule>
</extension>
```

After the design is complete, the designer adds this hierarchy to the database by restarting Ariba Analysis.

Loading Supplier Risk Data

Referring to the *Ariba Analysis Data Load Guide*, the designer examines the data to be loaded and designs a corresponding data load event. The designer then sends the system administrator a file containing the data to be loaded and the files defining the loading event.

The system administrator executes `runTask` to actually load the data and then validates the data to make sure the loading was successful. The administrator then exports the loaded data to a file to return to the designer for validation.

Examining the Data

The business analyst's spreadsheet in which the risk rating for all suppliers is maintained looks like the following:

SupplierID	SupplierName	Risk
01002	ABC Tech	1
00010	Ariba	0
2432	FlyByNight	2
1024	Quondam Software	1
...		

The designer determines that the easiest way to load this data is to save the spreadsheet in CSV (comma-separated values) format. The designer saves the file as `RiskLoad.csv`, which must first be transferred to the system that houses Ariba Analysis. Then the designer will run a data load event with that file as the data source.

Setting up a Data Load Event and Defining the Data Load

The designer creates a `DataLoadEvent.table` entry called `SupplierRiskLoad`, as follows:

```
SupplierRiskLoad = {
  DataSourceParams = {
    Filename =
    "config/sourceTypes/Global/sourceSystems/Default/data/RiskLoad.csv";
    };
  Threads = 4;
  };
```

The designer creates a data load definition with the data-loading XML:

```
<!DOCTYPE allDataLoads SYSTEM
"../../../../ariba/analytics/core/dataLoads.dtd">
<allDataLoads>
<dataLoad name="SupplierLoadCSV">
  <loadStages>
    <csvStage/>
    <analysisStage destinationName="ariba.analytics.dimension.Supplier"/>
  </loadStages>
  <fieldMappings>
    <field name="SupplierId">
       <csvMapping selectField="SupplierId"/>
    </field>
    <field name="SupplierName">
       <csvMapping selectField="SupplierName"/>
    </field>
    <field name="Risk">
       <csvMapping selectField="Risk"/>
       <analysisMapping>
          <mapValue implementation="ariba.analytics.mapValue.Decode">
             <parameter name="mapKeys">
                <vector>
                   <entry value="0"/>
                   <entry value="1"/>
                   <entry value="2"/>
                </vector>
             </parameter>
             <parameter name="mapElements">
                <vector>
                   <entry value="Low"/>
                   <entry value="Medium"/>
                   <entry value="High"/>
                </vector>
             </parameter>
          </mapValue>
```

```
        </analysisMapping>
      </field>
    </fieldMappings>
</dataLoad>
</allDataLoads>
```

The designer's data load event and definition are ready, so the designer sends them to the Ariba Analysis system administrator, along with the data file and instructions about how to execute the data load event.

Running runTask

The Ariba Analysis system administrator saves the data file from the designer, making sure to us a location and file name to match the path defined by the `Filename` parameter of the `SupplierRiskLoad` data load event, relative to `<CodePlaceHolder>AnalysisServerRoot`:

```
config/sourceTypes/Global/sourceSystems/Default/data/RiskLoad.csv
```

To run this data load, the system administrator sees `runtask`:

```
cd <CodePlaceHolder>AnalysisServerRoot
bin/runtask -dataload SupplierRiskLoad –sourceSystem demo
```

Validating the Load

After running the `SupplierRiskLoad` data load event, the system administrator defines two scheduled tasks:

1. Validate the supplier risk hierarchy in place.
2. Dump the data that was loaded to a file to give to the designer to determine if the load was correct.

The administrator sets up the tasks in the `config/DataLoadTasks.table` file. To validate the data, the administrator decides to use the `ValidateDimension` task and defines a scheduled task called `ValidateSupplierRisk`, as follows:

```
ValidateSupplierRisk = {
   ScheduledTaskClassName = "ariba.analytics.tasks.ValidateDimension";
   DimensionClass = "ariba.analytics.dimension.Supplier";
   };
```

To create a file based on the data that was loaded, the administrator relies on the DumpDimension task to define the scheduled task DumpSupplierRisk:

```
DumpSupplierRisk = {
  ScheduledTaskClassName = "ariba.analytics.server.DumpDimension";
  DimensionClass = "ariba.analytics.dimension.Supplier";
  Hierarchy = “Risk”;
  Filename = "DumpSupplierRisk.csv";
  };
```

He then executes runTask to process the tasks:

```
cd <CodePlaceHolder>AnalysisServerRoot
bin/runTask -task ValidateSupplierRisk –sourceSystem demo
bin/runTask -task DumpSupplierRisk –sourceSystem demo
```

After runTask is complete, the system administrator examines the data load event log <CodePlaceHolder>AnalysisServerRoot/logs/AnalysisLoadDBLog.txt for errors. Seeing no errors reported during data loading, the system administrator then sends a copy of the following file to the designer for review:

```
config/sourceTypes/Global/sourceSystem/Default/DumpSupplierRisk.csv
```

Adding Supplier Risk to the Materialized View

The designer decides to add supplier risk data to the materialized view of the InvoiceLineItem fact for speed. The materialized view of the InvoiceLineItem fact and its associated dimensions is defined in the following file:

```
<CodePlaceHolder>AnalysisServerRoot/ariba/variants/Plain/extensions/SpendAnal
ysis.aml
```

Because the materialized view is predefined, the designer realizes that an extension to it is needed. The designer therefore uses the <inMaterializedView> element:

```
<inMaterializedView name="InvoiceLineItem">
  <!-- supplier risk field -->
  <mvField name="Supplier" level="Risk" columnName=”Supplier Risk”/>
</inMaterializedView>
```

The designer then recomputes the materialized view. The scheduled task ComputeMatView accomplishes this (see the *Ariba Analysis Configuration Guide* for details):

```
ComputeInvoiceMatView = {
   ScheduledTaskClassName = “ariba.analytics.tasks.ComputeMatViews";
   MatView = “InvoiceLineItem”;
   };
```

Chapter 4
Examples of Data Access Control

In addition to controlling data visibility with metadata XML as discussed in "Changing User Data Views with Metadata XML" on page 51, you can establish finer grained limitations by writing your own custom data access control classes in Java.

Ariba Analysis comes with two examples of data access control implementations:

- The User Profile Editor and XML-based rules to protect dimension data
- Rule-based access control, complete with source code

Note: Ariba intends the supplied data access control mechanisms in Ariba Analysis as examples of possible implementations of access control. In development, you can use the example access control manager to apply XML-based or file-based rules to user names to limit those users' view of data.

In production, however, such permissions and viewing limitations are established in Ariba Buyer or Ariba Enterprise Sourcing and enforced when a user logs into Ariba Analysis from one of those applications. See *Ariba Spend Management Integration Guide* for more details.

The remainder of this chapter discusses the two examples:

Example Access Control: XML-based Profile Editor

Familiarize yourself with the first example of data access control by looking at the User Profile Editor in the Ariba Analysis administrative console, which is described in the *Ariba Analysis Configuration Guide*. With the example access control, an administrator employs the User Profile Editor to specify XML-based rules to permit users to see certain facts.

This section contains the following topics:

How The XML User Profile Access Control Works

With the example access control mechanism in Ariba Analysis, each user has a profile that determines whether or not that user has rights to see the facts she wants to view. For each user whose access you want to restrict, you must use the User Access Profile Editor to add lines to the user's profile.

Only a user with the `AnalysisAdmin` permission can invoke the User Access Profile Editor. For details about how to use the User Access Profile Editor, see the *Ariba Analysis Configuration Guide*.

Custom Java for Access Control

This section describes how to implement your own access control manager class in Java using the provided API. There are two general steps:

1. Writing a Java program that uses the example Ariba Analysis access control API
2. Registering your access control manager in `Parameters.table` with the `AccessControlManager` parameter

Where to Store Your Customizations

Before you begin writing Java, you should create a new subdirectory under config/java called custom. This clearly segregates your code from the rest of the system, which simplifies migration.

Remember also to change the `package` statement in your code to reflect the new structure.

If you put class files in the directory<CodePlaceHolder>AnalysisServerRoot/`classes/extensions` (following the appropriate package hierarchy), you do not need to change the classpaths defined in<CodePlaceHolder>AnalysisServerRoot/`classes/extensions/classpath.txt`. `However,` if you put class files in a different directory, you must modify `classpath.txt` to include that directory.

Writing Java

Your Java program must import the `ariba.analytics.mdql` package:

```
import ariba.analytics.mdql.AccessControlManager;
```

The default access control manager that comes with Ariba Analysis is suitable for you to study:

```
sample/java/ariba/analytics/mdql/DefaultAccessControlManager.java
```

Your custom access control manager must subclass from `AccessControlManager`. It should return an `AccessRule` object for a given `AccessContext`.

The following is the definition of the superclass `AccessControlManager`:

```
abstract public class AccessControlManager
  {
    abstract public AccessRule getRule (AccessContext context);
  }
```

AccessContext Class Methods

The `ariba.analytics.mdql.AccessContext` object encapsulates the user and the fact or dimension table to which an access rule is being applied.

The parameter `tableName` specifies the fact or dimension to which the rule applies.

This class relies on the `ariba.user.core.User` class. Consult the JavaDoc in Ariba Buyer for information about that class. The parameter `user` is the user name to which the rule applies.

The following are all the methods in this class.

Method	Returns
`public String` **`getUniqueName`** `();`	The unique user object to which this access control is applied.
`public String` **`getPasswordAdapter`** `();`	The Password Adapter associated with the user.
`public Vector` **`getAllPermissions`** `();`	All permissions associated with the user.
`public Vector` **`getAllRoles`** `();`	All roles associated with the user.

AccessRule Object Methods

Your custom access control manager should create an `AccessRule` object and use methods in that object to restrict the user to certain values or disallow the user from viewing certain values. The following is the API for `AccessRule`.

Method	Description
`public AccessRule (AccessContext context);`	Constructor for `AccessRule`
`public void allowOnly (String fieldPath, Vector values);`	Allows the user to access only the set of values for the given field path. A field path has the form: `DimensionReferenceName.FieldName` For example, `OrderedDate.Year` is a valid field path.
`public void block (String fieldPath, Vector values);`	Block access to a set of values.

Core User Object Methods

You might need to use the following methods to test against users' roles or permissions.

Method	Description
`public boolean hasRole (String role);`	Returns `true` if the user or group has the specified role.
`public boolean hasPermission (String permission);`	Returns `true` if the user or group has the specified permission.
`public boolean hasRolePermission (String rolePermission);`	Returns `true` if the user or group has the specified role or permission.

Registering Your Access Control Manager

After you write the access control manager, register it in the file `config/Parameters.table` under the key `System.Analysis.AccessControlManager`. You must specify the package name of the access control manager Java class. For example:

```
System.Analysis.AccessControlManager =
“classes.MyClasses.DefaultAccessControl”;
```

In addition, following this example, you must ensure that your custom class is located as follows:

```
<CodePlaceHolder>AnalysisServerRoot/classes/extensions/config/java/accesscont
rol/MyAccessControlManager.class
```

The actual location depends on where you decide to store your class file. See “Where to Store Your Customizations” on page 70.

You must restart the Ariba Analysis for the change to take effect.

Example File-based Rules for Access Control

In the other example of the Ariba Analysis data access control manager, in Ariba Buyer or Ariba Enterprise Sourcing you assign a user to a role or group, as usual. However, in Ariba Analysis you apply rules that govern what data the user can see. Rules are stored in the following file, a hash table

<CodePlaceHolder>AnalysisServerRoot/sample/java/ariba/analytics/mdql/Access
ControlRule.table

Rules can be expressed in terms of a user name, or role, or permission. The rules are evaluated against facts, not dimensions.

Source Code for Example File-based Rules

Ariba Analysis includes the Java source code for the file-based rules example of access control:

```
<CodePlaceHolder>AnalysisServerRoot/sample/java/ariba/analytics/mdql/ \
AccessControlManagerExample.java
```

How the File-based Rules Work

When a user attempts to access data in a fact table, the following processing occurs, in this order:

1. User name: If there is a rule for this user for this fact, Ariba Analysis enforces that rule.
2. Permission: If there is a permission-oriented rule matching one of user's permissions, Ariba Analysis enforces this rule instead.
3. Role: If there is a role-oriented rule matching one of this user's roles, Ariba Analysis enforces this rule instead.
4. Ultimately, allow the user to view only his own data and that of his subordinates.

Formulation of Rules in the Example

Here is an explanation of the allowed parameters in a rule file.

Parameter	Description
User	A single user name or multiple user names delimited by commas, surrounded by parentheses
Role	A single role name or multiple role names delimited by commas, surrounded by parentheses
Permission	A single permission name or multiple permission names delimited by commas, surrounded by parentheses
Fact	Optional. The complete name in dot notation of an Ariba Analysis fact table. If no `Fact` parameter is specified, the rule applies to all facts.
FieldPath	Required. The name of a field in the specified `Fact` or in all facts.
FieldValue	A comma-delimited list of records in the specified `FieldPath`
`Operation`	There are two operations: `allow`: Default. Permit access to the specified objects `block`: Deny access to the specified objects

Note: If no rule matches a user (based on user name, role, and permission), that user can see all facts but only his own transactions and his subordinates. Subordinates are determined by the relationships defined by the Management hierarchy of the User dimension.

The supplied <CodePlaceHolder>AnalysisServerRoot/config/AccessControlRule.table file looks like so:

```
{
Rule1 = {
Users = (qvu);
Facts = (ariba.analytics.fact.POLineItem);
FieldPath = “CostCenter.CostCenterId”;
FieldValues = (10010,20301,20305,20345);
Operation = “allow”;
}
Rule2 = {
Roles = (FinancialUser);
FieldPath = “Account.AccountId”;
FieldValues = (10230,10300,10301,10302,20000,30700);
Operation = “allow”;
}
Rule3 = {
Roles = (FinancialUser);
Facts = (ariba.analytics.fact.POLineItem, ariba.analytics.fact.ERLineItem,
ariba.analytics.fact.ContractFact)
FieldPath = “Account.AccountId”;
FieldValues = (10230,10300,10301,10302,20000,30700);
Operation = “allow”;
}
}
```

The first rule, `Rule1`, works on a single user name. It specifies that a user named `qvu` is allowed to access the records (indicated by `FieldValue`) in the Cost Center field (`FieldPath`) of the PO fact.

The second rule, `Rule2`, specifies that the role `FinancialUser` is allowed to access the records (indicated by `FieldValue`) in the AccountID field (`FieldPath`) of all facts.

Appendix A
Ariba Analysis Metadata XML Reference

This appendix contains the formal syntax of the elements of Ariba Analysis metadata XML.

Ariba Buyer Metadata XML in Ariba Analysis

Ariba Analysis metadata XML builds on much of the existing Ariba Buyer metadata XML. The use of many elements in Ariba Analysis matches their exact use in Ariba Buyer metadata XML. Any element not described in this section is documented in *Ariba Buyer Configuration Reference Guide*. Such elements include but are not limited to the following:

- `<deleteTrigger>`
- `<extension>`
- `<import>`
- `<inField>`
- `<overrides>`
- `<trigger>`

Ariba Analysis Metadata XML

This appendix is an alphabetical list of all elements in Ariba Analysis metadata XML. The description of each element includes the following:

- Attributes of the element and a description of each
- Elements this element is contained in
- Elements that this element can contain

<deleteHierarchy>

		Required?	Default value
Attributes			
	name	Required	(none)
Contained In			
	<inDimension>		

The <deleteHierarchy> element removes a previously declared <hierarchy> from a <dimension>.

name Attribute

The required name attribute indicates the name of the <hierarchy> you are deleting. For example:

```
<inDimension name="ariba.analytics.dimension.Account">
  <inLevel name="Account">
    <!-- definition of Account level goes here -->
  </inLevel>
  <levels name="myHierarchy> maxLevels="3">
    <!-- definition of AccountHierarchy levels goes here -->
  </levels>
  <!-- Remove Account hierarchy to rely on myHierarchy -->
  <deleteHierarchy name="Account"/>
<inDimension>
```

<deleteMVField>

		Required?	Default value
Attributes			
	name	Required	(none)
	level	Optional	(none)
	levelNumber	Optional	(none)
Contained In			
	<inMaterializedView>		

The <deleteMVField> element removes a previously defined <mvField> from a materialized view.

name Attribute

The required name attribute indicates the value of a name attribute of an <mvField> you are deleting from this materialized view.

```
<deleteMVField name="Supplier" ...>
```

level Attribute

The optional level attribute for <deleteMVField> indicates the level of the <mvField> you are deleting.

levelNumber Attribute

The optional levelNumber attribute for <deleteMVField> indicates the levelNumber of the <mvField> you are deleting.

<dimension>

		Required?	Default value
Attributes			
	name	Required	(none)
	datafile	Optional	(none)
	bucketMinField	Optional	(none)
	bucketMaxField	Optional	(none)
	bucketNameField	Optional	(none)
	bucketValues	Optional	(none)
	versioning	Required	false
Contains			
	<defaulter> <hierarchy> <groupLevel> <index> <level> <lookupKey> <overrides> <trigger>		

The <dimension> element declares a new dimension table.

name Attribute

The required `name` attribute specifies the name of the `<dimension>` you are declaring and by convention is the name of a Java class in the `ariba.analytics.dimension` package. For example:

```
<dimension name="ariba.analytics.dimension.SourceSystem" ...>
```

The final component of the value for `name` must be unique.

datafile Attribute

The optional `datafile` attribute specifies where Ariba Analysis stores the `<dimension>` in the database: either in a filespace or a tablegroup. The values for `datafile` on the `<dimension>` element are identical to those described for the `<fact>` element in "datafile Attribute" on page 84.

bucketMinField Attribute

The `bucketMinField` attribute indicates the name of a field that represents the lower bound of a bucket. This is any arbitrary field whose values you have determined you want to use as a lower limit of a bucket.

The range is inclusive of the value of `bucketMinField` and exclusive of the value of `bucketMaxField`.

The field you specify as a `bucketMinField` must be of type `int` and must implement the `java.lang.Comparable` interface.

bucketMaxField Attribute

The `bucketMaxField` attribute indicates the name of a field that represents the upper bound of a bucket. This is any arbitrary field whose values you have determined you want to use as an upper limit of a bucket.

The range is inclusive of the value of `bucketMinField` and exclusive of the value of `bucketMaxField`.

The field you specify as a `bucketMaxField` must be of type `int` and must implement the `java.lang.Comparable` interface.

bucketFlagField Attribute

The optional `bucketFlagField` attribute specifies the name of a field that indicates that the data record is part of the bucketing definition.

Some records are merely data that must be grouped into the buckets. Other records are data, as well, but also define the size of the buckets themselves. That is, these records have `bucketMinField` and `bucketMaxField` values that represent the size of the buckets you want to define.

You indicate that such records define a bucket by specifying the name of a field as the the value of the `bucketFlagField` attribute. The type of a field you indicate as a `bucketFlagField` must be Boolean. For a record to be considered part of a bucket definition, its `bucketFlagField` must be the literal word `true`.

bucketNameField Attribute

The optional `bucketNameField` attribute specifies the name of a field that holds named buckets of the bucketing definition. This field pints to the actual bucketed data in the dimension.

bucketValues Attribute

The optional `bucketValues` attribute specifies that this is a dimension used to bucket the values of the dimension. *Bucketing* is to group the values into ranges or "buckets" for easier use in an analytical report. See "Bucketing a Dimension" on page 42 for an extended example.

The `bucketValues` attribute is a Boolean. Set it either `true` or `false`.

The `bucketValues` attribute is only optional in that you can choose to bucket a dimension or not. If you choose to bucket a dimension, then you must include all related attributes:

```
bucketValues
bucketMinField
bucketMaxField
bucketFlagField
```

Buckets are created dynamically from the data themselves. As such, a bucket must have a lower limit and an upper limit, so that data can be grouped into it. You specify these minimum and maximum values by indicating which fields in the data hold values you want to use as the basis of the groupings.

Buckets are created dynamically from the data themselves. As such, a bucket must have a lower limit and an upper limit, so that data can be grouped into it. You specify these minimum and maximum values by indicating which fields in the data hold values you want to use as the basis of the groupings.

versioning Attribute

The `versioning` attribute on the `<properties>` element associated with a `<dimension>` indicates that Ariba Analysis must maintain a historical record of all versions of the dimension's field values for auditing. You can version dimensions case-by-case.

Except in the demonstration configurations, by default, versioning is not enabled:

```
<properties versioning=”false”>
```

To enable it, set the following:

```
<properties versioning=”true”>
```

Data-loading a Slowly Change Dimension

In `sample/demoConfig/Global/config/sourceTypes/dataLoads/DemoBuyerLoads.xml` the PO load refers to the Contract dimension, which is an example of a slowly changing dimensio. To key the versioning during data loading, an effective date field is specified in the data-load definition:

```
<field name="Contract.ContractId">
  <csvMapping selectField="Contract"/>
</field>
<field name="Contract.VersionEffectiveDate">
  <csvMapping selectField="OrderedDate"/>
</field>
```

The `VersionEffectiveDate` is not a customizable field. This field name is required. However, you can specify either the OrderedDate or ApprovedDate or whatever date you desire to indicate the time that this fact is valid in respect to this slowly chanmging dimension.

<disableVectorDim/>

		Required?	Default value
Attributes			
	name	Required	(none)
Contained in			
	<inFact>		

The <disableVectorDim> element disables a multi-dimension. Ariba Analysis provides several multi-valued dimension fields in its sample definitions. However, some multi-valued dimensions are sometimes not needed. Unneeded definitions in the metadata XML impact the performance of database queries.

You can disable a field that refers to a multi-valued dimension with the <disableVectorDim> element in <inFact>.

```
<inFact name="ariba.analytics.fact.InvoiceLineItem">
  <disableVectorDim name="Approver"/>
</inFact>
```

This essentially removes the field from the Ariba Analysis metadata XML. The field still exists in the data model but is not included in database queries.

For a discussion of multi-valued dimension fields, see “Dimensions with Multi-Valued Fields” on page 45.

name Attribute

The required name attribute on <disableVectorDim> specifies the name of the multi-valued dimension field to disable.

<fact>

		Required?	Default value
Attributes			
	name	Required	(none)
	datafile	Optional	(none)
Contains			
	<defaulter> <field> <index> <lookupKey> <measure> <overrides> <properties> <trigger>		

The <fact> element declares a new fact table.

name Attribute

The required name attribute specifies the name of the <fact> you are declaring and by convention is the name of a Java class in the ariba.analytics.fact package. For example:

```
<fact name="ariba.analytics.fact.POLineItem" ...>
```

The final component of the value for name must be unique.

datafile Attribute

The optional datafile attribute specifies how Ariba Analysis must store the data for this fact in the database: either in a filespace or a tablegroup. For example:

```
<fact name="ariba.analytics.fact.POLineItem"
datafile="mycompany.smallTableDataFile">
```

The value of datafile must be a tablespace or filespace defined in the Parameters.table configuration file and must be one of the following to specify a small, medium, or large index or table space.

```
System.Database.SmallIndexDataFile
System.Database.SmallTableDataFile
```

```
System.Database.MediumIndexDataFile
System.Database.MediumTableDataFile
System.Database.LargeIndexDataFile
System.Database.LargeTableDataFile
```

<field>

		Required?	Default value
Attributes			
	bucketName	No	(none)
	bucketSourceValue	No	(none)
	inlineDimension	No	false

In addition to the functions of the attributes of the <field> element in Ariba Buyer, <field> in Ariba Analysis has the attributes described here.

bucketName Attribute

You use the bucketName attribute to indicate the name of grouping of data (for example, a range of Amounts) in a bucketed dimension. See "Bucketing a Dimension" on page 42 for details about bucketing.

In the following example, the field gets its data from a bucket named Amount in the Amount dimension:

```
<field name="AmountRange"
  nullAllowed="false"
  bucketName="Amount"
  bucketSourceValue="Amount">
  <type class="ariba.analytics.dimension.AmountRange"/>
</field>
```

bucketSourceValue Attribute

You use the bucketSourceValue attribute to indicate the data source of the bucketed dimension of the bucket whose name you specify with the bucketName attribute. See "Bucketing a Dimension" on page 42 for details about bucketing. In the following example, the source field for the AmountRange bucket is the Amount field:

```
<field name="AmountRange"
  nullAllowed="false"
```

```
   bucketName=”Amount”
   bucketSourceValue="Amount">
   <type class="ariba.analytics.dimension.AmountRange"/>
</field>
```

By default this field definition applies to the base currency. If you want to bucket amount values for other currencies, extend your metadata XML to specify a currency. For example, the following specifies a `bucketSourceValue` for the Japanese yen `Amount`.

```
<field name="AmountRangeJPY"
   nullAllowed="false"
   bucketSourceValue="Amount.JPY">
</field>
```

inlineDimension Attribute

You use the `inlineDimension` attribute to indicate the name of a fact that can be added to the row, column, or page fields of the pivot table. See “Inline Dimensions” on page 36 for details about inline dimensions.

For example, the following field definition defines the line type field as an inline dimension:

```
<field name="LineType" nullAllowed="false" inlineDimension="true">
```

<groupLevel>

		Required?	Default value
Attributes			
	name	Required	(none)
	maxLevels	Required	(none)
	sourceLevel	Required	(none)
	sourceLevelLookup	Optional	(none)
	sourceLevelLookupParent	Required	(none)
Contained in			
	<dimension> <inDimension>		
Contains			
	<field> <index> <lookupKey> <properties>		

The <groupLevel> element is similar to <level>. You can use it to group levels together by indicating their parent-child relationships.

For every <groupLevel> you define, you must run the scheduled task GroupLevelSetup, as described in the *Ariba Analysis Configuration Guide*.

name Attribute

The required name attribute specifies the name of the group of level you are defining. For example:

```
<groupLevel name="Organization" ...>
```

maxlevels Attribute

The required maxLevels attribute indicates the maximum number of levels allowed in this group.

sourceLevel Attribute

The required `sourceLevel` attribute indicates the level in the hierarchy to which this group applies.

sourceLevelLookup Attribute

The required `sourcelevelLookup` attribute specifies the lookup key of the level to which this level relates.

sourceLevelLookupParent Attribute

The required `sourcelevelLookupParent` attribute specifies the lookup key of the parent level.

Example

```
<groupLevel name="Organization"
     maxLevels="2"
     sourceLevel="Supervisor"
     sourceLevelLookup="SupervisorId"
     sourceLevelParentLookup="ParentOrgId">
  <index fields="OrganizationName" bitmap="true"/>
  <lookupKey fields="OrganizationId"/>
  <field name="OrganizationName">
    <type class="java.lang.String"/>
    <properties label="Organization"/>
  </field>
  <field name="SupervisorId">
    <type class="java.lang.String" length="50"/>
  </field>
</groupLevel>
```

<hierarchy>

		Required?	Default value
Attributes			
	name	Yes	(none)
	levels	Yes	(none)
	rollupType	No	(none)
	searchSubLevels	No	false
Contains			
	<properties>		
Contained In			
	<dimension> <inDimension>		

The <hierarchy> element constructs a new hierarchy.

levels Attribute

The required levels attribute indicates the previously defined levels that constitute this <hierarchy>. The name of at least one level is required in a hierarchy.

For example, suppose that you had defined the levels lineManager, midManager, and topManager. You can construct a <hierarchy> of these levels like so:

```
<hierarchy name="MyCompanyHierarchy"
levels="lineManager,midManager,topManager">
```

name Attribute

The required name attribute specifies the name of the <hierarchy> you are constructing. For example:

```
<hierarchy name="MyCompanyHierarchy" ...>
```

rollupType Attribute

The optional `rollupType` attribute specifies how the `<hierarchy>` is totaled. Setting `rollupType` to false means that the hierarchy cannot be rolled-up in the normal way without introducing errors in calculation of totals or other unwanted effects. Setting this attribute to `true` (or not specifying the attribute, which is the default) indicates that the hierarchy can be rolled-up correctly.

searchSubLevels Attribute

The `searchSubLevels` attribute controls how deep into a hierarchy a user can drill down to select particular values for a parameterized report. You may want to restrict the user's ability to search only to the top level of a hierarchy.

By default, the value of the `searchSubLevels` attribute is `false`. You must specify `searchSubLevels` only if you want to set it to `true`, as shown in the following example of the Management hierarchy:

```
<hierarchy name="Management"
   levels="Management,UserId"
   searchSubLevels="false">
```

<index>

		Required?	Default value
Attributes			
	fields	Required	(none)
	unique	Optional	false
	bitmap	Optional	false

The `<index>` element creates a database index on a specified field or list of fields. Using the `<index>` element has the same effect as setting the `indexed` attribute on a `field`, except that the attributes of the `<index>` element allow greater flexibility. With the `<index>` element, you can include multiple fields in the index and you can specify a uniqueness constraint.

The only time you can specify the `<index>` element is when you are adding a new fact or dimension class and want a database index on one or more fields in that class. Each `<index>` element constructs one index composed of any `fields` in the fields attribute.

The fields Attribute

The value of the `fields` attribute is either a string or a comma-separated list of strings. For example:

```
<index fields="SupplierID"/>
<index fields="SupplierID,PartNumber"/>
```

The unique Attribute

The `unique` attribute is a Boolean that is passed to the underlying database index create operation to make a unique index. In effect, this attribute equates to the SQL statement `CREATE UNIQUE INDEX`.

The default is `false`, which means the index is not unique. If an index is created as unique, the database disallows duplicate values in the index, and prevents clients from inserting a record into the table.

The bitmap Attribute

The `bitmap` attribute is a Boolean that specifies whether to create a bitmap index in Oracle. This attribute is valid only for Oracle databases.

<inDimension>

		Required?	Default value
Attributes			
	name	Required	(none)
	datafile	Optional	(none)
Contains			
	<defaulter> <hierarchy> <groupLevel> <index> <level> <lookupKey> <overrides> <trigger>		

The <inDimension> element indicates the name of a previously defined <dimension> you wish to change. You must **not** declare the <inDimension> element and the <dimension> it changes in the same metadata XML file.

Most of the elements allowed in <inDimension> are the same as those allowed in <dimension>. The value of an element in <inDimension> overrides the value of the element with the same name declared in the original <dimension>.

name Attribute

The required name attribute specifies the name of the <dimension> you are altering.

```
<inDimension name="ariba.analytics.dimension.SourceSystems" ...>
```

This <dimension> must have been previously declared.

datafile Attribute

With the optional datafile attribute, you can store the <dimension> in a filespace or tablegroup other than where the original <dimension> is stored. The values for datafile on <inDimension> are identical to those described for the <fact> element in “datafile Attribute” on page 84.

<inFact>

		Required?	Default value
Attributes			
	name	Required	(none)
	datafile	Optional	(none)
Contains			
	<disableVectorDim> <defaulter> <deleteTrigger> <field> <index> <lookupKey> <measure> <overrides> <trigger>		

The <inFact> element indicates the name of a previously defined <fact> you wish to change. You must not declare the <inFact> element and the <fact> it changes in the same metadata XML file.

Most of the elements allowed in <inFact> are the same as those allowed in <fact>. The value of an element in <inFact> overrides the value of the element with the same name declared in the original <fact>.

name Attribute

The required name attribute specifies the name of the <fact> you are altering. For example:

```
<inFact name="ariba.analytics.fact.POLineItem" ...>
```

This <fact> must have been previously declared.

datafile Attribute

The optional datafile attribute can be used to store the <fact> in a filespace or tablegroup other than where the original <fact> is stored. The values for datafile on <inFact> are identical to those described for the <fact> element in “datafile Attribute” on page 84.

<inGroupLevel>

		Required?	Default value
Attributes			
	name	Required	(none)
Contains			
	<defaulter> <deleteTrigger> <field> <index> <lookupKey> <measure> <overrides> <trigger>		

The <inGroupLevel> element indicates the name of a previously defined <groupLevel> you wish to change.

Most of the elements allowed in <inGroupLevel> are the same as those allowed in <groupLevel>. The value of an element in <inGroupLevel> overrides the value of the element with the same name declared in the original <groupLevel>.

name Attribute

The required name attribute specifies the name of the <greopLevel> you are altering. For example:

```
<inGroupLevel name="Geography" ...>
```

This <groupLevel> must have been previously declared.

<inLevel>

		Required?	Default value
Attributes			
	name	Required	(none)
	datafile	Optional	(none)
Contains			
	<defaulter> <deleteTrigger> <field> <index> <inField> <lookupKey> <properties>		

The <inLevel> element indicates the name of a previously defined <level> you wish to change. The <inLevel> element and the <level> it changes must **not** be declared in the same metadata XML file.

Most of the elements allowed in <inLevel> are the same as those allowed in <level>. The values of elements specified in <inLevel> override the values of the elements with the same names originally declared in the original <dimension>.

name Attribute

The required name attribute specifies the name of the <level> you are altering.

```
<inLevel name="CostCenter" ...>
```

This <level> must have been previously declared.

<inMaterializedView>

		Required?	Default value
Attributes			
	name	Required	(none)
Contains			
	<deleteMVField> <mvField>		

The <inMaterializedView> element indicates the name of a previously defined <materializedView> you wish to change. You must not declare the <inMaterializedView> element and the <materializedView> it changes in the same metadata XML file.

The <mvField> element in <inMaterializedView> works the same way as in <materializedView>. The value of an <mvField> in <inMaterializedView> overrides the value of the element with the same name declared in the original <materializedView>.

name Attribute

The required name attribute specifies the name of the <materializedView> you are altering.

```
<inMaterializedView name="POLineItem">
```

This <materializedView> must have been previously declared.

<level>

		Required?	Default value
Attributes			
	name	Required	(none)
	displayField	No	(none)
Contained in			
	<dimension> <inDimension>		
Contains			
	<field> <index> <lookupKey> <properties>		

The <level> element defines a level that contains no further sublevels in a <dimension>.

If the <level> you are defining contains other levels, you can use the <groupLevel> element instead of <level> to specify parent-child relationships. Using <groupLevel is sometimes more efficient than explicit creating individual levels, which is labor-intensive. See “<groupLevel>” on page 87.

name Attribute

The required name attribute specifies the name of the <level> you are defining. For example:

```
<level name="TopBoss" ...>
```

displayField Attribute

The optional displayField attribute specifies a label displayed in the user interface for this level. For example:

```
<level name="TopBoss" displayField="HeapBigHoncho">
```

If you do not specify a displayField, Ariba Analysis displays the value of the name attribute.

<lookupKey>

		Required?	Default value
Attributes			
	fields	Required	(none)
Contained in			
	<dimension> <fact>		

The <lookupKey> element specifies the fields of a <dimension> or <fact> that Ariba Analysis uses to create a unique database key for each record.

During data loading, Ariba Analysis searches for a ClusterRoot object that equates to the incoming <lookupKey>. If one is found, the corresponding database record is updated. If not, a new record is created.

<materializedView>

		Required?	Default value
Attributes			
	name	Required	(none)
	fact	Required	(none)
Contained in			
	<dimension> <inDimension>		
Contains			
	<mvField>		

The <materializedView> element declares a new materialized view of an existing fact table.

name Attribute

The required name attribute specifies the name of the materialized view you are constructing.

```
<materializedView name="POSpecialView" ...>
```

fact Attribute

The required fact attribute indicates the name of the fact table to which this view relates.

```
<materializedView name="POSpecialView"
fact="ariba.analytics.fact.POLineItem">
```

This <fact> must have been previously declared.

<measure>

		Required?	Default value
Attributes			
	includeDimension	Optional	(none)
	includeHierarchy	Optional	(none)
	name	Required	(none)
	noPersist	Optional	(none)
	nullAllowed	Required	(none)
Contained in			
	<fact> <inDimension>		
Contains			
	<measureOperator> <properties> <type>		

The <measure> element declares a new analytical measure of a <fact>.

includeDimension Attribute

The optional includeDimension attribute specifies the simple name of a <dimension> that should be included when Ariba Analysis exports this measure to Microsoft Excel:

```
<measure name="Quantity" nullAllowed="false"
  includeDimension="Part"
  includeHierarchy="UnitOfMeasure">
```

includeHierarchy Attribute

The optional includeHierarchy attribute specifies the simple name of a <hierarchy> that should be included when Ariba Analysis exports this measure to Microsoft Excel:

```
<measure name="Quantity" nullAllowed="false"
  includeDimension="Part"
  includeHierarchy="UnitOfMeasure">
```

name Attribute

The required name attribute specifies the name of the <measure> you are defining. For example:

```
<measure name="Quantity" ...>
```

Each <measure> must have a unique name.

noPersist Attribute

By default, a <measure> persists in the database. To prevent database persistence, use the noPersist attribute, but only with measures that use the count <measureOperator>. The noPersist attribute is most common if you want to use a measure only in the user interface, with no value that must be saved in the database.

The attribute noPersist is a Boolean. Valid values are true or false.

```
<measure name="Size" noPersist=true>
  <measureOperator type="count"/>
  <type class="int"/>
</measure>
```

nullAllowed Attribute

The required nullAllowed attribute on the <measure> element indicates whether or not a <measure> may be empty. The attribute nullAllowed is a Boolean. Valid values are true or false. You must set nullAllowed to false because measures must not be empty.

```
<measure name="Size" noPersist="false" nullAllowed="false">
```

Elements Required in <measure>

The <measure> element requires a <type> subelement to define the Java data type associated with the measure or whether the measure supports multiple currencies. The <measure> element also requires a <measureOperator> subelement to declare the mathematical function of the measure. For example:

```
<measure name="SupplierCount" nullAllowed=false>
  <type class="int"/>
  <measureOperator type="count"/>
</measure>
```

Multiple Currency Type for Measures

To indicate that a measure supports multiple currencies, use the <type> element with the special class MultiCurrency. All amount measures in the default configuration allow multi-currency. For example, the Amount measure of the POLineItem fact is defined with a multi-currency type:

```
<measure name="Amount" nullAllowed="false">
  <type class="MultiCurrency"/>
  <measureOperator type="sum"/>
  <properties label="Amount"
    description="Amount of money spent"/>
</measure>
```

<measureOperator>

		Required?	Default value
Attributes			
	type	Required	(none)
Contained in			
	<measure>		

The <measureOperator> element specifies the mathematical function of a <measure>.

type Attribute

The required `type` attribute on `<measureOperator>` indicates the mathematical function of the `<measure>`. Allowable values are as follows. The `<measureOperator>` must also contain a corresponding `<type>` element with a class attribute that specifies the Java data type of the `<measureOperator>`.

Value of type Attribute	Meaning
`sum`	addition
`count`	count of records
avg	average

<mvField>

		Required?	Default value
Attributes			
	`name`	Required	(none)
	`level`	Required	(none)
	`LevelNumber`	Optional	(none)
	`columnName`	Optional	(none)
Contained in			
	`<materializedView>` `<inMaterializedView>`		

An `<mvField>` element specifies the data that constitute a materialized view.

name Attribute

The required `name` attribute specifies the value of a name attribute of a `<field>` in the fact table for which you are constructing this materialized view. This `<field>` must exist in the `<fact>` associated with this view.

```
<mvField name="Supplier" ...>
```

level Attribute

The required `level` attribute on `<mvField>` indicates at what level the data for this view must be rolled up.

An `<mvField>` is just like a `<field>` in a `<fact>`. It has one or more levels associated with it. For example:

```
<mvField name="Supplier" level="TopBoss" ...>
```

levelNumber Attribute

The required `levelNumber` attribute on `<mvField>` indicates what level in a `<groupLevel>` is being rolled up. Rolling up the data requires that Ariba Analysis know the level of the data.

An `<mvField>` is just like a `<field>` in a `<fact>`. It has one or more levels associated with it. For example:

```
<mvField name="Supplier" level="TopBoss" levelNumber="1">
```

columnName Attribute

Ariba Analysis automatically generates the name of the column it displays for this field in the materialized view. You can override this automatic name and specify one that you prefer by using the optional `columnName` attribute.

```
<mvField name="Supplier" level="TopBoss" levelNumber="1" columnName="Exec">
```

<properties>

		Required?	Default value
Attributes			
	constraintHierarchy	No	(none)
	description	No	(none)
	excludeFromExport	No	(none)
	inMemorySearch	No	(none)
	label	No	(none)
	rank	No	(none)
	selfDisplaySuffix	No	(none)
	versioning	No	(none)
Contained in			
	<groupLevel> <hierarchy> <inLevel> <level> <measure>		

Use the <properties> element to accomplish any of the following:

- Constrain data from a hierarchy when a user contextually links from another Ariba application
- Prevent certain data from being exported to Microsoft Excel
- Change the position of a fact in the Source Data pulldown in Step 1 of the Analysis Wizard
- Add explanatory text about an object
- Specify a label in the user interface. This annotation is viewable by a user in creating or editing an analytical report.

constraintHierarchy Attribute

The optional `constraintHierarchy` attribute specifies a hierarchy of an Ariba Analysis dimension to which data should be limited when a user contextually links to Ariba Analysis from another Ariba application. Suppose a dimension has several associated hierarchies. The `constraintHierarchy` attribute indicates the one specific hierarchy that should be used to constrain the user's view of the data.

For example:

```
<override name="ClassProperties">
  <properties label="Region"
    constraintHierarchy="Region"/>
</override>
```

description Attribute

The optional `description` attribute specifies the explanatory text about the object, which Ariba Analysis displays when the user positions a cursor over the object. If you do not specify a `description`, Ariba Analysis displays the value of the `label` attribute. For example:

```
<measure name="Size" noPersist=true nullAllowed=false>
  <properties label="Size"
    description="Size is the total number of employees in an organization.">
  <measureOperator type="count">
</measure>
```

For the use of the @ character in labels and descriptions, see "Descriptions and Labels in Resource Files" on page 57.

excludeFromExport Attribute

You can extend your metadata XML to include fields or other object that you may not want to allow users to export, such as counts you use only internally. Use the `excludeFromExport` attribute to prevent such objects from being eligible for export.

inMemorySearch Attribute

The `inMemorySearch` attribute causes Ariba Analysis to look in memory for values of some kinds of menu options.

label Attribute

The optional `label` attribute specifies text displayed in the Ariba Analysis user interface. If you do not specify a `label`, Ariba Analysis displays the value of the `measure` element's `name` attribute. For example:

```
<measure name="Size" noPersist=true nullAllowed=false>
  <properties label="Size" description="Size is the total number of employees
  in an organization.">
  <measureOperator type="count">
</measure>
```

selfDisplaySuffix Attribute

The `selfDisplaySuffix` attribute specifies a parenthetical label for the parent level of a `<hierarchy>` displayed in the Ariba Analysis pivot table. When a parent level has data associated with it that must be counted along with its children to achieve accurate totals, such a hierarchy is called *ragged*. In a regular hierarchy, only the sublevels contain data. Thus, a total for the entire hierarchy is the sum of its children. A ragged hierarchy, however, contains data at its top level not associated with any of its levels that must be taken into account.

For example, the smooth hierarchy **Management** might contain three sublevels: **Executive**, **Middle**, **Line**. This hierarchy can be accurately totaled by the sum of its levels, thus:

	Total
Management	9
Executive	1
Middle	3
Line	5

Conversely, a ragged hierarchy of spend sourcing projects, say for **Chemicals**, in which some projects are assigned at the topmost level of the hierarchy itself, must include those projects attributed to the topmost level to sum an accurate total. In the following list, the suffix **(Itself)** has been set for the **Chemicals** hierarchy using the `<properties>` attribute `selfSuffixDisplay`:

	Total
Chemicals	11
Sulfur	2
Boron	1
Chemicals (Itself)	**5**
Krypton	3

versioning Attribute

The option `versioning` attribute on the `<properties>` element associated with a `<dimension>` indicates that Ariba Analysis must maintain a historical record of all versions of the dimension's field values for auditing. Such a dimension is called a *slowly changing dimension*.

By default, versioning is not enabled. To enable it, set the following:

```
<properties versioning="true">
```

<type>

		Required?	Default value
Attributes			
	`vector`	No	(none)
Contained in			
	`<field>`		

Use the `<type>` element to specify that a field is a multi-valued field with the attribute `vector=true`.

vector Attribute

Set the optional `vector` attribute to `true` if the field is multi-valued. See "Dimensions with Multi-Valued Fields" on page 45 for a discussion. For example:

```
<field name="Region" nullAllowed="fals">
   <type class="ariba.analytics.dimension.Region" vector="true"/>
</field>
```

Ariba Analysis Glossary

These are definitions of commonly used terms in Ariba Analysis.

Term	Definition
80/20 rule	A widely used Pareto analysis that divides data into a ratio of most/least. For example, 80% of your spend may be accounted for by only a handful of suppliers, while the remaining 20% may represent a large number of suppliers.
AML	See *metadata XML*.
analysis	See *analytical report*.
analytical report	An organized collection of measures and hierarchies designed to investigate particular aspects spend-related facts. An analytical report is manifested as a *pivot table*. See also *reports*.
Applied Filters	The area across the top of the *pivot table* of an analytical report that shows what constraints have been imposed on the data and what level of data is currently displayed in the pivot table. Click any level in the Applied Filters to return the pivot table to trat level.
bucket	A defined grouping of data. For example, a range of dates or of amounts.
calculated field	See *derived measure.*
collapse	To view less data by hiding a *level* in a *hierarchy.*
column edge, column fields	The area across the top of a *pivot table* directly above the data fields where *data fields* or *line-level detail* are placed. Column fields correspond to the columns in a traditional spreadsheet.
compound report	In Ariba Analysis, a compound report is a user-defined composite of pivot tables, charts, and summarized views from existing analytical reports. Compound reports allow you to view many different charts and tables at a single glance.

Term	Definition
computed field	See *derived measure.*
constrain	To reduce the amount of data actively being investigated in an analytical report by applying some exclusion, for example, selecting a particular level of a hierarchy or a particular value in a hierarchy, or selecting a range of dates.
data fields	The area in the middle of a *pivot table* holds the data fields. Also called *measures*, data fields are created in several ways: • *Pre-defined data fields* are created by summing or aggregating *line-level details*, which are transaction data. • *User-defined fields* are created by you by calculating them based on other data fields. The data field area of the pivot table also holds *line-level details,* the basic accounting transactions from which all other fields are derived.
derived measure	A measure computed from some other measures, called *user-defined fields*.
dimension	An aspect of a fact that is of some interest and has some useful purpose in an analytical report. For example, **Supplier** and **Commodity** are dimensions of purchase orders and invoices, just as start date is a dimension of a project or end date is a dimension of a contract.
drill-down	To show finer detail by going to the next level of a hierarchy.
edge	Term referring to the underlying structure of an *OLAP* hypercube represented by the *pivot table*. A hypercube has *row*, *column*, and *page* edges.
expand	To view more data by exposing the next lower *level* of a *hierarchy.*
export	To extract data from Ariba Analysis and download it to your personal computer.

Term	Definition

fact	In traditional OLAP terminology, facts represent the subject—the interesting pattern or event in the enterprise that must be analyzed to understand its behavior. They represent the subject of your investigation. The following are the pre-defined facts in Ariba Analysis:
	• Purchase orders and PO delivery information: commodity purchasing patterns
	• Expense reports, expense report lines and violations: travel expense spending patterns
	• Invoices, invoice lines and exceptions: supplier billing patterns
	• Contracts and contract clauses: agreements established with suppliers from Ariba Contract Workbench
	• Requests: a general term for employee requisitions, travel requests, and similar data, including collaborative requisitions
	• User activity: Ariba Buyer usage and information
	• Projects: Ariba Category Management- and Ariba Contract Workbench-related facts
	  • Contract projects
	  • Sourcing projects
	  • Supplier Performance Management projects and tasks:
	• RFX summaries and awards: Ariba Enterprise Sourcing requests for proposal or information
	• Surveys and scorecards Ariba Enterprise Sourcing supplier evaluation data
	• Temporary labor and time sheets: services spend information
	• Proposals
	•

Field Browser	The palette on the left of an Ariba Analysis pivot table that lists fields that have been added to or can be edited from an analytical report.
grade	A specialized form of *user-defined field* that calculates a numerical score based on the value of the data.

Term	Definition
hierarchy	A stratified collection of data characterized by *levels*. Any particular level has finer-grained data than the level above it and coarser-grained data than the levels beneath it, unless the level is at the top or bottom of the hierarchy.
inline dimension	A *fact* in Ariba Analysis that can be placed on the row, column, or page edges of the Ariba Analysis pivot table, just like a dimension.
interval	A period of time between two events. For example, Ariba Analysis calculates the PO-to-Invoice Interval as the number of days between the PO or ER date and the corresponding invoice date. A negative number means the invoice was created before the PO, which is not desirable. This calculated interval of days is used to classify invoices into several buckets of time ranges, such as 3 to 6 weeks.
KPI	Key Performance Indicator
level	A stratum of a certain granularity of the data in a hierarchy. For example, “month” is a level lower in the time hierarchy than “year.”
line-level detail	The lowest level of detail in a fact. That is, the individual transactions. See also *data field*.
measure	A numerical property of a fact or a calculation based on such a property. For example, Amount is a measure of a purchase order. Measures are also known as *data fields*.
metadata XML	Data structures and data loading are defined in Ariba Analysis with metadata XML. For data structures, this XML is contained in files ending with `*.aml`. For data-loading definitions, files end with `*.xml`.
non-rollup hierarchy	A *hierarchy* in which the total is not represented by a straight sum of the levels, such as a score of “Pass” or “Fail.”

Term	Definition
OLAP	On Line Analytical Processing, a specialized architecture of databases for dealing with aggregation of large datasets. OLAP systems are characterized by *facts*, *measures*, *dimensions*, and *hierarchies*, represented by fields on a hypercube or *pivot table*. For a comprehensive discussion of OLAP, see (for example): `http://infolab.usc.edu/csci585/Spring2002/den_ar/pederson_p40.pdf`
page edge, page field	The area across the top of a *pivot table* directly above the column fields where *data fields* (or *measures*) are placed that can act as a filter on the other fields.
parameterized report	An analytical report that allows parameters to constrain the data. You set parameters such as dates, money amounts, or hierarchies and levels to specific values you are interested in.
pivot table	A spreadsheet-like structure consisting of page fields, row fields, column fields, and data fields that allows you to manipulate the view of the data. It is the end-product of creating an analytical report. See also *Field Browser.*
ragged hierarchy	A *hierarchy* in which the top-level contains data that must be summed to make an accurate total.
range	In a spreadsheet, a collection of cells.
report	Ariba Analysis has three kinds of reports: • *analytical* • *parameterized* • *compound* See the definitions of these terms for distinctions.
row edge, row fields	The area down the left side of a *pivot table* next to the data fields where *dimensions* are placed. Row fields correspond to the rows in a traditional spreadsheet.
slice and dice	See *constrain.*

Term	Definition
slowly changing dimension	A dimension whose individual values might change over time. Also called *versions* or *versioning*. The values of the dimension that change are recorded for auditing.
split line-item	A single line of a purchase order or expense report that is charged to more than one General Ledger account.
summarized view	In an Ariba Analysis compound report, a summarized view is a user-created table that extracts specific values or totals of values from analytical reports to reveal relationships about multiple facts that might not otherwise be readily visible.
UNSPSC	The Universal Standard Products and Services Classification code is a scheme that classifies and identifies commodities. It is used in sell-side and buy-side catalogs and as a standardized account code in analyzing expenditure (Spend Analysis). See `http://eccma.org/unspsc/`
UOM	Unit (or units) of measure
user-defined field	See *derived measure.*
versioning	See *slowly changing dimension*.

Index

Third-Party Software

Apache Software

The Apache Software License, Version 1.1

Copyright (c) 2000 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "Apache" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache," nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

org.apache.bcel-5.0

4. The names "Apache," "Apache Software Foundation," and "Apache BCEL" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

org.apache.commons.collections-2.1

4. The names "The Jakarta Project," "Commons," and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

org.apache.lucene-1.2

4. The names "Apache," "Apache Software Foundation," and "Apache Lucene" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache," or "Apache Lucene," nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

org.apache.oro-2.0

4. The names "Apache," "Apache Software Foundation," and "Jakarta-Oro" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache" or "Jakarta-Oro," nor may "Apache" or "Jakarta-Oro" appear in their name, without prior written permission of the Apache Software Foundation.

Portions of this software are based upon software originally written by Daniel F. Savarese. We appreciate his contributions.

org.apache.poi.hssf-1.5.1

Portions of this software are based upon public domain software originally written at the National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign.

org.apache.regexp-1.2

4. The names "The Jakarta Project," "Tomcat," and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

org.apache.soap-2.2

4. The names "SOAP" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

org.apache.tomcat-3.2.4

4. The names "The Jakarta Project," "Tomcat," and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

org.apache.xerces-1.4

4. The names “Xerces” and “Apache Software Foundation” must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation and was originally based on software copyright (c) 1999, International Business Machines, Inc., http://www.ibm.com. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

com.jclark 1.0

CONTAINS software developed by James Clark

Copyright (c) 1997, 1998 James Clark

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL JAMES CLARK BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Infozip.zip-2.3

CONTAINS software developed by Info-ZIP

Copyright (c) 1990-2003 Info-ZIP. All rights reserved.

For the purposes of this copyright and license, “Info-ZIP” is defined as the following set of individuals:

Mark Adler, John Bush, Karl Davis, Harald Denker, Jean-Michel Dubois, Jean-loup Gailly, Hunter Goatley, Ian Gorman, Chris Herborth, Dirk Haase, Greg Hartwig, Robert Heath, Jonathan Hudson, Paul Kienitz, David Kirschbaum, Johnny Lee, Onno van der Linden, Igor Mandrichenko, Steve P. Miller, Sergio Monesi, Keith Owens, George Petrov, Greg Roelofs, Kai Uwe Rommel, Steve Salisbury, Dave Smith, Christian Spieler, Antoine Verheijen, Paul von Behren, Rich Wales, Mike White

This software is provided “as is,” without warranty of any kind, express or implied. In no event shall Info-ZIP or its contributors be held liable for any direct, indirect, incidental, special or consequential damages arising out of the use of or inability to use this software.

Infozip.zip-2.3, continued

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. Redistributions of source code must retain the above copyright notice, definition, disclaimer, and this list of conditions.

2. Redistributions in binary form (compiled executables) must reproduce the above copyright notice, definition, disclaimer, and this list of conditions in documentation and/or other materials provided with the distribution. The sole exception to this condition is redistribution of a standard UnZipSFX binary (including SFXWiz) as part of a self-extracting archive; that is permitted without inclusion of this license, as long as the normal SFX banner has not been removed from the binary or disabled.

3. Altered versions--including, but not limited to, ports to new operating systems, existing ports with new graphical interfaces, and dynamic, shared, or static library versions--must be plainly marked as such and must not be misrepresented as being the original source. Such altered versions also must not be misrepresented as being Info-ZIP releases--including, but not limited to, labeling of the altered versions with the names “Info-ZIP” (or any variation thereof, including, but not limited to, different capitalizations), “Pocket UnZip,” “WiZ” or “MacZip” without the explicit permission of Info-ZIP. Such altered versions are further prohibited from misrepresentative use of the Zip-Bugs or Info-ZIP e-mail addresses or of the Info-ZIP URL(s).

4. Info-ZIP retains the right to use the names “Info-ZIP,” “Zip,” “UnZip,” “UnZipSFX,” “WiZ,” “Pocket UnZip,” “Pocket Zip,” and “MacZip” for its own source and binary releases.

ARIBA, INC.

807 11th Avenue
Sunnyvale, CA 94089 USA
Telephone: 650.390.1000
Fax: 650.390.1100

www.ariba.com