Ariba Analysis
Data Load Guide

Version 3.0
July 2004

Table of Contents

Preface

The *Ariba Analysis Data Load Guide* describes how to load data into Ariba Analysis from data sources such as other Ariba Spend Management applications, external databases, or comma-separated value (CSV) files.

Audience and Prerequisites

This document is appropriate for Ariba Solutions Delivery personnel or Ariba certified partners who have successfully installed Ariba Analysis. It is appropriate for a technical audience.

This document is part of the Ariba Analysis document set.

The Ariba Analysis documentation set contains the following books.

Title	Audience	Purpose and contents
Ariba Analysis Installation Guide	System administrators	• Architectural overview, software components, deployment configurations • Step-by-step installation
Ariba Analysis Configuration Guide	System administrators	• Configuration files, command reference, parameters, administration console • System security • Managing users and reports
Ariba Analysis Customization Guide	Systems integrators and data warehouse designers	• Working with and extending the data model • Adding facts, measures, dimensions, and materialized views • Controlling data visibility • Customizing the user interface

Title	Audience	Purpose and contents
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
Ariba Analysis Advanced User Guide	Procurement and sourcing business analysts, systems integrators	• 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

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 Data Load 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 of Data Loading

Ariba Analysis is a tool for analyzing data. To configure Ariba Analysis, you must define how to load the data to be analyzed and the schedule for refreshing that data. For example, you might set up a data load that pulls purchase order data from an Ariba Buyer configuration, and run that data load once a week or once a month, according to your preferred business process.

During the initial configuration phase, you also run a set of integration tasks that define common data, such as the names of currency events. This data is typically loaded just once, or infrequently, and does not need to be updated regularly.

This document describes how to load data into Ariba Analysis. It describes both how to set up data load events that you can run regularly and how to configure your initial set of common data.

This chapter serves as an introduction to the data load process and a roadmap to the rest of the document. It includes the following sections:

How to Load Data

To configure Ariba Analysis, you initialize your database by loading in appropriate data. Data loading can involve any or all of the following:

- *Integration tasks*, which load common data, such as units of measure
- *Data load events*, which load analytic data, such as requisitions
- *Data load tasks*, which process the data, such as by updating database statistics

In a typical configuration, you run a complete set of tasks and events for initial database initialization. The database initialization phase loads common data, loads an initial set of data for analysis, and runs tasks to set up database statistics and indexes.

After database initialization, you can run individual events to refresh specific data or you can do a partial incremental update of the initial database data. For example, you can run an incremental update that updates all of your analytical data, but only the data that has changed recently.

For information on the mechanics of loading data into Ariba Analysis, see the following chapter:

- Chapter 2, "Commands for Loading Data"

Data Load Events

A *data load event* is an Ariba Analysis event that you run to pull analytic data into Ariba Analysis. You use data load events to load data to be included in your analytical reports. Each data load event pulls data from a named *source system*, which specifies the database or application from which you are loading the data.

When you configure Ariba Analysis, you specify a set of source systems (systems from which you load data) and a set of data to be loaded from each source system. You can run data events that load data from a variety of data sources, such as CSV files, database tables, other Ariba Spend Management applications, and ERP systems.

The major steps for defining a data load event are as follows:

- Defining source systems.
- Defining data load events, which specify the data to be pulled from each source system.
- Defining any required data transformations or mapping of fields from the data source to the Ariba Analysis data model.

The default configuration defines an initial set of data load events, which you can modify and customize to suit your configuration.

Defining Source Systems

To load data, you must define a *source system* for each possible data source. Each source system has a *source type*, which can be a CSV file, an ERP system, or another Ariba application. For example, if you are reading data from an Ariba Category Management database, the data is extracted with an Ariba Query API query, but if you are reading data from an external database, the data is extracted with a SQL query.

For information on defining source systems, see the following chapter:

- Chapter 3, "Source Systems for Data"

Defining Data Load Events

Each source system in your configuration has a list of *data load events*. A data load event specifies the data to be loaded, the data source from which the data is pulled, and how to map the data into the destination Ariba Analysis schema.

For information on how to define data load events, see the following chapter:

- Chapter 4, "Data Load Events"

Defining Data Transformations

Ariba Analysis uses *data-loading XML* to define data transformations or mappings between the source system and the destination fact or dimension in Ariba Analysis. A mapping can be a simple naming map or it can include operations such as concatenating two fields together.

The default configuration supplies mappings for standard data load events. To customize your configurations, you can define extension files that build on the default mappings. For example, you can add additional clauses to the query statement used to pull data from a database, or you can add a temporary database staging table to perform database joins or other manipulations.

For information on how to define mappings for new data, or customize the existing mappings, see the following chapter:

- Chapter 5, "Data-loading XML"

Using Incremental Load

A data load event can be defined to be either incremental or full. If a data load is defined to be incremental, Ariba Analysis checks the source system to determine which records have been added, changed, or deleted since the last time you loaded that data, and then updates only those records that have changed. If a data load is defined to be full, Ariba Analysis always loads the full set of data.

For information on how to define and run incremental data load events, see the following chapter:

- Chapter 6, "Incremental Data Loading"

Common Data

When you first set up your configuration, you initialize your configuration with a set of *common data*. Common data is data that is shared by all Ariba Spend Management applications, and represents standard names for entities such as languages, currencies, and units of measure. Common data is data that does not change often; for example, it is rare to introduce a new language or country name.

To load common data, you run integration tasks that pull the data either from another Ariba Spend Management application or from a CSV file on your disk.

For data such as units of measure and currency, you can potentially support several parallel naming conventions. In these case, you load several sets of data and also define mappings between the different systems of nomenclature. For example, for units of measure you can define mappings between US measurements and metric measurements, and for currencies you can pull currency conversion rates from a data source as part of your data loading process.

For information on how to load common data and define mappings between different naming conventions, see the following chapters:

- Chapter 7, "Loading Common Data"

Data Exports

In addition to loading data into Ariba Analysis, you also sometimes need to export data out to disk files. For example, you might want to export data to disk and then re-import it later if you are planning to migrate to a new version of Ariba Analysis, or if you are moving from a development instance to a production instance.

Ariba Analysis defines a set of tasks that you can use to selectively dump users, reports, dimensions or facts to disk files. After dumping the data, you can manipulate the data on disk and then import the disk files back into Ariba Analysis.

For information on how to dump data to disk files, see the following chapter:

- Chapter 8, "Dumping Data"

Troubleshooting

The following chapter provides information on logging, diagnostic tools, debugging techniques, and performance tuning:

- Chapter 9, "Troubleshooting"

Reference Information

The appendixes of this document provide reference information on file formats and syntax:

- Appendix A, "Data-loading XML Reference," provides reference information on the elements and attributes that can appear in data-loading XML files.
- Appendix B, "File Formats," provides reference information on the file formats used in Ariba Analysis, such as the configuration files and CSV data files.
- Appendix C, "Command Reference," provides reference information on the syntax of commands described in this document, such as `initdb` and `runtask`.
- Appendix D, "Data Load Tasks," describes data load tasks that are typically run during full database initialization.

Chapter 2
Commands for Loading Data

This chapter describes how to run data load events or tasks, either as part of a full data load or after changes to a particular dimension. It includes the following sections:

About Data Loading

When you first install and initialize Ariba Analysis, you initialize your database by loading in appropriate data. Data loading can involve any or all of the following:

- *Integration tasks*, which load common data, such as units of measure
- *Data load events*, which load analytic data, such as requisitions
- *Data load tasks*, which process the data, such as by updating database statistics

Ariba Analysis provides configuration files, in table file format, that define the details of the events or tasks to be run. Integration tasks are defined in `MessageConfiguration.table`, data load events are defined in `DataLoadEvents.table`, and data load tasks are defined in `DataLoadTasks.table`.

In a typical configuration, you run a complete set of tasks and events for initial database initialization. The database initialization phase defines common data, loads an initial set of data for analysis, and sets up database statistics and indexes.

After database initialization, you can run individual events to refresh specific data, or you can do a partial incremental update of the initial database data. For example, you can run just the event to load new purchase requisition data from Ariba Buyer, or you can run an incremental update that updates your analytical data by refreshing any data that has changed recently.

To run one or more data load events or tasks, either as a full update or an incremental update, you use the `initdb` command. The `initdb` command reads an input file that describes the events or tasks to run.

To run one event or task, you use the `runtask` command, which takes an argument to specify the event to run.

This chapter describes the syntax of loading data. Later chapters describe the details of the configuration files for each kind of data loading task, and describe how to customize specific data load events.

The initdb Command

To run several events at a time, define the list of tasks in an input file, and use the `initdb` command. For example:

```
initdb -loaddb -file LoadDB.txt
```

The `initdb` command takes as a parameter the name of a *initialization order file*, which lists a set of data load tasks and events. The default initialization order file is `config/LoadDB.txt`. The default configuration also supplies a second initialization order file, `LoadDB.basic.txt`, which lists the events used for loading common data.

The log file for `initdb` is *`AnalysisServerRoot`*`/logs/LoadDBLog.txt`.

Initialization Order Files

The general syntax of lines in an initialization order file is as follows:

```
Task.TaskName
sourceSystemName.DataLoad.dataLoadEventName-qualifier
```

The keywords (such as `Task` and `DataLoad`) specify whether the line refers to an integration task, a data load event, or a data load task. For example, this line refers to a data load event:

```
Default.DataLoad.BuyerPOLoad
```

For complete information on the syntax of an initialization order file, see “Initialization Order Files (LoadDB.txt)” on page 103.

Validating Data

After loading new data, you must run tasks that reshape the dimensions and materialized views in the database, based on the changes that you have made.

When you load data with `initdb`, the last task in `LoadDB.txt` is the `DBOLAPProcessing` task, which creates database statistics and materialized views. The `DBOLAPProcessing` task runs three smaller tasks: `ValidateDimension`, `ComputeDBStats`, and `ComputeMatView`.

For complete information on the `DBOLAPProcessing` task and its subtasks, see "ComputeDBStats" on page 123.

Running Full Database Initialization

If you are doing a full initialization, Ariba recommends that you shut down Ariba Analysis, ensuring no user access, and then load in your new data. For information on how to shut down Ariba Analysis gradually, honoring any pending requests from current users, see *Ariba Analysis Configuration Guide*.

▼ To do full data initialization:

1. Use the `TimeSetup` task to set your date ranges.

   You **must** use this task before you load any data into Ariba Analysis, to ensure that Ariba Analysis knows the beginning and ending date range for your data. For more information about this task, see "Time Setup" on page 132.

2. Run `initdb`:

   ```
   initdb -initdb
   ```

3. Run the scheduled task `LoadDBDonePoller`, or make sure that it is scheduled to run soon. This task recreates the cache used for generating analytical reports. For information on this task, see the *Ariba Analysis Configuration Guide*.

The runtask command

After you have loaded your initial set of data, you can run individual events one at a time. For example, if you define a new data load event, you can load the data for just that new dimension, without reloading your entire database.

To run just one data load event, use the `runtask` command. For example:

```
runtask -task BuyerPOLoad
```

Validating Data

After loading new data, you must run tasks that reshape the dimensions and materialized views in the database, based on the changes that you have made.

If you load data with `initdb`, you run the `DBOLAPProcessing` task, which in turn runs three smaller tasks: `ValidateDimension`, `ComputeDBStats`, and `ComputeMatView`. If you are reloading just a single dimension, you typically create entries for these tasks that recreate just a single dimension.

For complete information on the `DBOLAPProcessing` task and its subtasks, see "ComputeDBStats" on page 123.

Running a Specific Event

This section describes the steps required to run a specific data load event.

▼ To load one new data load event:

1. Use `runtask` to run the event:

   ```
   runtask -task MyNewEvent
   ```

2. Create an entry in `DataLoadTasks.table` that validates the new dimension, and run it. For example if the dimension you are modifying is Department:

   ```
   ValidateDepartment = {
       ScheduledTaskClassName = "ariba.analytics.tasks.ValidateDimension";
       DimensionClass = "ariba.analytics.dimension.Department";
   };
   ```

3 Create an entry in `DataLoadTasks.table` for the `ComputeMatView` task, which drops and recreates a specified materialized view. For example:

```
ComputeMatView = {
   ScheduledTaskClassName = “ariba.analytics.tasks.ComputeMatViews";
   MatView = “POLineItem”;
};
```

4 If your database is Oracle, create an entry that recreates database statistics. For example:

```
ComputeDBStats = {
   ScheduledTaskClassName = "ariba.analytics.tasks.ComputeDBStats";
};
```

Running this task after each data load is not strictly necessary, but if you have changed your data model significantly, you should run this task.

(This task does not apply on IBM DB2 or Microsoft SQL. For more information, see “DBOLAPProcessing” on page 122.)

Chapter 3
Source Systems for Data

This chapter describes how to define and configure the source systems in your configuration. It includes the following sections:

About Source Systems

A source system is a mechanism for segregating data loaded into Ariba Analysis. Each source system represents a distinct data source. For example, you typically define different source systems for each Ariba Spend Management application from which you pull data, and also different source systems for different Ariba Buyer partitions.

Each source system inherits from a *source type*, with definition files that describe the format of data that is loaded from the specified source system. Several source systems can share the same source type. For example, if you are integrating with an Ariba Buyer configuration that has several PeopleSoft partitions, you can use one source type definition, but build different data sources using that source type.

Source Types

A source type is a template for implementations of various configurations of data, metadata XML, data-loading XML and data load events. The configuration files for each source type are grouped together in the file system, in the following location:

`sourceTypes/`*`type`*

The default configuration provides several sample source types, defined in subdirectories of *AnalysisServerRoot*/configureOptions:

Directory Name	Description
ACM	Integrates with Ariba Category Management
buyer-csv	Integrates with Ariba Buyer CSV-based configuration
buyer-none	Integrates with Ariba Buyer's global unpartitioned data
buyer-oracle	Integrates with an Oracle partition for Ariba Buyer
buyer-psoft	Integrates with a PeopleSoft partition for Ariba Buyer
buyer-sap	Integrates with an SAP partition for Ariba Buyer
Global	Defines the default source system, from which other source systems inherit

Source Systems

When you create a new source system, you specify an existing source type to use as the template for the new source system. Ariba Analysis copies configuration files from that source type directory to create the initial version of your new source system.

The files for each source system are located in subdirectories of the source type:

sourceTypes/*type*/sourceSystems/*name*

Ariba Analysis requires one global source system, called Default, and can potentially include any number of additional source systems. You can create source systems with the configurator, or you can create them later from the command line with the addsourceystem command.

Data Files

For a source system that reads data from CSV files, the data files associated with these source systems in the following directories:

config/sourceTypes/*typeName*/sourceSystems/*sourceSystemName*/data

The addsourcesystem Command

You use the `addsourcesystem` command to create a new source system. The `addsourcesystem` command copies a set of template files into your new source system, creating an initial set of configuration files that you can edit. It does the following:

- Adds a definition of the new source system to the `SourceSystems` section of `config/Parameters.table`.
- Creates a file called `DataLoadEvents.table` in each source system subdirectory. You use this file to define the set of events that load data from the specified source system. For more information on this file, see "Data Load Events" on page 31.
- Creates appropriate directories in the file system for your configuration files.

Arguments to addsourcesystem

The `addsourcesystem` takes options that specify the source type, the name of the new source system, and (for source systems that integrate with other applications), information on how to connect to that source system. The following example shows how to create a source system for integration with Ariba Buyer's partition None:

```
addsourcesystem -configOption baseConfig -baseType buyer-None -typeName Global
-systemName Buyer -connKey Buyer -partition None
```

The arguments are as follows:

- `-configOption` specifies whether you are using the base config or the demo config. The valid values are `-baseConfig` and `-demoConfig`. The `demoConfig` option creates a standalone demo system that pulls from CSV files. If you are integrating with a real Ariba Spend Management application intsance, you should always use `baseConfig`.
- `-baseType` specifies the template directory to use when creating this source system. The value must be one of the directory names listed in "Source Types" on page 25.
- `-typeName` specifies the name of the source type you are using.
- `-systemName` specifies the name of the source system you are creating. Your source system names must be unique.
- `-connKey` identifies an instance in `AppInfo.xml`. This value identifies the instance with which you are connecting. This option is necessary if you are connecting to another application, such as Ariba Buyer or Ariba Category Management.

For complete reference information on the available options for addsourcesystem, see "addsourcesystem" on page 113.

Example of Creating Source Systems

This section describes shows one set of examples that use addsourcesystem to create an initial set of source systems. For examples of commands to create source systems for integrating with Ariba Category Management and Ariba Enterprise Sourcing, see "addsourcesystem" on page 113.

▼ To define a new source system:

1. If you did not create a global default source system with the configurator, create one from the command line. Most of the time, this step is **not** necessary, because the Global source system has already been created by the configurator. Do not run this command if you already have a Global source system, because it will override your existing settings.

   ```
   addsourcesystem -configOption baseConfig -baseType Global -typeName Global
   -systemName Default
   ```

2. If you are integrating with Ariba Buyer, create a source system buyer-none. This source system integrates with global (non-partitioned) data in Ariba Buyer and is required for any configuration that integrates with Ariba Buyer. This source system must use -typeName Global. For example:

   ```
   addsourcesystem -configOption baseConfig -baseType buyer-None -typeName
   Global -systemName Buyer -connKey Buyer -partition None
   ```

   For the buyer-none source system, value of the -systemName parameter must match the value of the -connKey parameter.

3. For each additional source type you want to define, use the addsourcesystem script to create initial directories and files for the appropriate source type. For example, to create a source system for an Ariba Buyer partition integrated with PeopleSoft:

   ```
   addsourcesystem -configOption baseConfig - buyer-psoft -typeName PSOFT
   -systemName psoft1 -connKey Buyer -partition PCSV
   ```

4. Store any CSV data files associated with these source systems in the following directories (which are created by addsourcesystem):

   ```
   config/sourceTypes/Global/sourceSystems/Default/data
   config/sourceTypes/PSOFT/sourceSystems/psoft1/data
   ```

Parameters.table Entries for Source Systems

In `config/Parameters.table`, the `SourceSystems` parameter is a nested table that defines a set of parameters for each source system.

For example:

```
SourceSystems = {
  Default = {
    DataLoadEventsFile =
    "config/sourceTypes/Global/sourceSystems/Default/DataLoadEvents.table";
    };
  };
```

This entry illustrates just one source system, the global `Default` system. Other entries are similar. The entry for each source system has a label (`Default`, in this example), and a set of parameters for that source system.

When you create a new source system, the `addsourcesystem` command adds an appropriate entry for that source system, using the parameters you have specified.

Source System Mapping

If you have two source systems that share underlying data, you can specify that relationship by defining a mapping in the `SourceSystems` entry of `config/Parameters.table`. For example, suppose that you have a configuration with the following two source systems:

- an Oracle ERP system
- an Ariba Buyer partition that integrates with the Oracle system

The Ariba Buyer configuration pulls certain basic information from Oracle, such as suppliers. To avoid duplication of the supplier data, you can specify that all source systems for that data are normalized to one source system.

You use the SourceSystemMapping parameter to specify the common source system mapping for each dimension. For example:

```
SourceSystems = {
  sourcing = {
    DataLoadEventsFile =
     "config/sourceTypes/sourcing/sourceSystems/sourcing/DataLoadEvents.table";
       SourceSystemMapping = {
           ariba.analytics.dimension.Commodity = Buyer;
           ariba.analytics.dimension.Supplier = Buyer;
           ariba.analytics.dimension.UserData = Buyer;
       };
    };
  }
```

In this example, the values for the Commodity, Supplier, and UserData dimensions in the sourcing source system are normalized to the values for those same dimensions in the Buyer source system.

Chapter 4

Data Load Events

This chapter describes how to define data load events that load data from the source systems you have defined. It includes the following sections:

DataLoadEvents.table Files

Each source system in your configuration has a file called `DataLoadEvents.table`, which defines a set of data load events for that source system. The location of `DataLoadEvents.table` for each source system is specified in `config/Parameters.table`, in the `SourceSystem` parameter.

The `DataLoadEvents.table` file defines a list of data load events for a source system. Each event defines the data to be loaded, the data source from which the data is pulled, and describes how the source schema is mapped into the destination Ariba Analysis schema.

The general structure of an entry in `DataLoadEvents.table` is as follows.

```
dataLoadEventName = {
   DataLoadName = "some_name" ;
   DataSourceParams = {
      /* parameters specific to the data source ....*/
      }
   }
   DisableFactLookup = true ;
   Operation = either Load or Delete or Update ;
   Threads = numberOfThreads ;
   UnionWithDataLoads = "comma-separated names of data load definitions"
}
```

The *dataLoadEventName* can be any string, not including spaces. You use this label to identify an event when you want to run it, either from as an argument to the runTask command or as an entry in LoadDB.txt.

The DataSourceParams key specifies parameters that define how to connect to the data source. For example, if the event loads from CSV, you specify the Filename parameter to indicate the file being used as the data source, and for an event that loads from a database, you can specify query parameters that are used to tailor the query for extracting data from the database.

This chapter illustrates common examples of defining events that load from CSV files and database. For complete reference information on the keys and values that can appear in a DataLoadEvents.table file entry, see "DataLoadEvents.table" on page 104.

Loading from a CSV File

To load from a CSV file, you must specify the Filename parameter to specify the location of the CSV file. For example:

```
CustLoad = {
  DataLoadName = "customer_load" ;
  DataSourceParams = {
    Filename =
    "config/sourceTypes/Files/sourceSystems/psft/data/customer.csv" };
};
```

This example refers to a <dataLoad> definition called customer_load in a data-loading XML file. The data-loading XML file maps columns in the CSV file to different data in Ariba Analysis. For more information on <dataLoad> definitions, see "Data-loading XML" on page 37.

If the column names in the CSV file are exactly the same as the target structure in Ariba Analysis, and no data transformation is required, set the DataLoadName parameter to the special value DirectCSVLoad. For example:

```
UserLoad = {
    DataLoadName = "DirectCSVLoad";
    DestinationName = "ariba.analytics.dimension.UserData";
    DataSourceParams = {...};
};
```

For information on the format of the CSV file from which you load data, see "CSV Data File Format" on page 107.

Loading from a Database

If you are loading data by running a query to extract data from a database, the data load definition specifies any details of connecting to the database. A simple data load to read from an application such as Ariba Buyer is as follows:

```
UserLoad = {
  DataLoadName = BuyerUser;
  Threads = 4;
}
```

This example refers to a <dataLoad> definition called BuyerUser in a data-loading XML file. You use this definition to map columns in the Ariba Buyer database to fields in Ariba Analysis. For information on how to create or modify <dataLoad> definitions, see "Data-loading XML" on page 37.

The <dataLoad> definition does not need to specify information on how to connect to Ariba Buyer, because that information is defined in the source system itself. The <dataLoad> definitions are made within a source system. However, you can set parameters to specify that an event applies only to a specific version of Ariba Buyer, as described in the rest of this section.

Loading from an External Database

For events that load data from a database, the default is to read database connection information from the source system. The SQLConnection and InterfaceSQLConnection parameters in the SourceSystems section of config/Parameters.table define the default database connections.

To override these defaults, you can define a named connection in the file ConnectionInfo.table and refer to that named connection from DataSourceParams. For example:

```
POLoad = {
  DataLoadName = "POload" ;

  DataSourceParams = {
    SQLConnection = sampleOracle ;
    }
  }
```

In ConnectionInfo.table, you define the corresponding label (sampleOracle in this example) and associated connection details. The label in DataLoadEvents.table refers to the full set of parameters defined in ConnectionInfo.table. For example:

```
{
  DBConnections = {
    sampleOracle = {
       DBType = Oracle;
       Driver = oracle.jdbc.driver.OracleDriver;
       Password = password;
       URL = "jdbc:oracle:thin:@HOSTNAME:PORTNUMBER:SID";
       User = user;
       };
  };
}
```

For information on ConnectionInfo.table, including information on how to encrypt values such as the password, see "ConnectionInfo.table" on page 110.

Using an Interface Table

A data load definition can define an explicit interface table stage, which creates a temporary database table used for intermediate processing before the data is loaded into Ariba Analysis. For example, you might need to use an interface table if you want to join two tables or tables from two or more databases or other data sources.

By default, interface tables are stored in the Ariba Analysis database itself. You can also specify a named database connection, with the InterfaceSqlConnection parameter. The default (storing in the Ariba Analysis database) is equivalent to the following:

```
InterfaceSqlConnection = "AnalysisDB";
```

Note: Interface tables have names of the form INT_ANALYSIS*. Make sure your own interface table names do not conflict with these reserved names.

Modifying Data Loads

This section describes additional settings for modifying the query used to pull data in a data load.

Language for Data Loads

Ariba Analysis loads data in only one language. If you are loading from a source system that has multilingual data, Ariba Analysis uses the parameter `Application.Base.Data.DefaultLanguage` in `config/Parameters.table` to specify the language of the data to be selected. When Ariba Analysis selects data from the source system, the source system selects multilingual data in the specified language.

The default value of this parameter is English. You can override this default language setting either for a given source system or a given data load event:

- To specify the language for a source system, use the `Language` keyword in a source system definition in `config/Parameters.table`.
- To specify the language for a data load event, use the `Language` keyword in your data load event. See "DataLoadEvents.table Files" on page 31.

Regardless of the character set of the source system, the character encoding of the data in Ariba Analysis is always UTF-8.

Adding Query Clauses

For any event that uses a database query (Ariba Query API or SQL) to query the database, you can add additional clauses to the query by specifying the following attributes in your data-loading XML file:

- `fromClause`
- `whereClause`
- `distinctFlag`
- `groupByFlag`
- `incrementalClause`
- `orderByClause`

For details on these attributes, see the reference information in "Database Attributes" on page 97.

Adding Query Parameters

You can use the parameter System.Analysis.DataLoading.QueryParams to pass values from the system parameters to the query layer, as query parameters. For example:

```
System.Analysis.DataLoading.QueryParams = {
    DomainName =
"@Application.ClassificationCodes.SystemCommodityCodeDomainName";
};
```

This example defines a parameter called DomainName, which is set to the value of the parameter SystemCommodityCodeDomainName. (If the value starts with the character @, the value is assumed to refer to another parameter.)

The DomainName parameter is then accessible from the query layer and can be used in the query you use to extract data from the database. For example:

```
whereClause="Code.Domain = ':DomainName' AND Code.UniqueName IS NOT NULL"
```

You can override the value of QueryParams for an individual event by setting QueryParams in DataLoadEvents.table. For example:

```
dataLoadEventName = {
    DataSourceParams = {
        QueryParams = { DomainName="xxxxx"}
    }
```

Chapter 5

Data-loading XML

You use data-loading XML to define data transformations or mappings during data load. Data-loading XML allows you to describe data to extract by writing XML expressions, instead of SQL or Ariba Query API statements.

This chapter includes the following sections:

Data-loading XML File Mechanics

This section describes where to find data-loading XML files, how to create extension files that modify the default configuration, and how to load extension files into your configuration.

Location of Data-Loading XML Files

Data-loading XML configuration files are stored in text files and have the extension `.xml`. They are always located in a directory named `dataLoads`.

Data-loading XML files can apply to one specific source system, or they can apply globally. The location of the file in the file system determines the scope. The default data-loading XML files are as follows:

Location in *`AnalysisServerRoot`*	Scope
`ariba/sourceTypes/Global/dataLoads`	Global
`ariba/sourceTypes/*/dataLoads`	A specific source type

The definitions are loaded in the following order:

1. The `Global` definitions from the `ariba` directory
2. The `Global` definitions from the `config` directory
3. Definitions for each *`sourceType`*, from the `ariba` directory
4. Definitions for each *`sourceType`*, from the `config` directory

Thus, you can change a `Global` definition first, and then add source-type specific changes after that. Within a given directory, definitions are loaded in alphabetical order.

Creating Extension Files

To make changes, you always create XML extension files, and store those extensions in the `config` directory. You never change default data-loading XML files directly.

▼ **To create an extension file:**

1. Create a named set of extensions in an extension file, with extension `.xml`.
2. If your extensions are global, store the file in:

   `config/sourceTypes/Global/dataloads/`

   If your extensions apply to a specific source type, store the file in:

   `config/sourceTypes/`*`yourSourceType`*`/dataloads`

As with the default files, global extensions are loaded first, followed by extensions specific to a given source type.

Elements of Data-loading XML

This section contains a discussion of the general structure of a data-loading definition. For full details of all data-loading XML elements, see Appendix A, "Data-loading XML Reference."

For examples, see the sample extensions in the following directory:

`AnalysisServerRoot``/config/sourceType`

General Structure of a Data Loading Definition

The overall outline of a data-loading definition is as follows:

```
<!DOCTYPE allDataLoads SYSTEM
../../../../ariba/analytics/core/dataLoads.dtd”>
<allDataLoads>
    <dataLoad name="someName">
        <loadStages>...</loadStages>
        <fieldMappings>...</fieldMappings>
    </dataLoad>
</allDataLoads>
```

Each <dataLoad> element defines data loaded from a given data source. Each <dataLoad> element contains the following elements:

- <loadStages>

  Defines the stages for data load. Every data load defines a source stage and a destination stage. For example:

  ```
  <loadStages>
    <csvStage/>
    <analysisStage destinationName="ariba.analytics.dimension.Supplier"/>
  </loadStages>
  ```

  This example specifies that the source is a CSV file and the destination is the named dimension in Analysis. A <loadStages> element can also declare an intermediate stage (a staging table), where you can perform additional processing.

- <fieldMappings>

  Defines relationships between fields or columns in the source or interface tables and their corresponding target fields in Ariba Analysis. For example:

  ```
  <fieldMappings>
    <field name="SupplierId">
       <csvMapping selectField="SupplierId"/>
       <analysisMapping>
          <mapValue implementation="ariba.analytics.mapValue.Decode">
          ...
       </analysisMapping>
    </field>
  ```

Load Stages

This section describes the different source systems you can define with <loadStages>.

CSV Stage

You use a <csvStage/> element to specify data read from a CSV file. For example, to use <csvStage/> to load data into the Commodity hierarchy:

```
<loadStages>
  <csvStage/>
  <analysisStage destinationName="ariba.analytics.dimension.Commodity"/>
</loadStages>
```

To specify the relationship between fields in the CSV file and their corresponding fields in the target, you use a <csvMapping> element.

AQL Stage

You use the <aqlStage/> element to specify data read from an Ariba Buyer or Ariba Category Management database, using an Ariba Query API query. For example, to use <aqlStage/> to load data into the Commodity hierarchy:

```
<loadStages>
  <aqlStage fromClause="CommodityCode INCLUDE INACTIVE"/>
  <analysisStage destinationName="ariba.analytics.dimension.Commodity"/>
</loadStages>
```

Many of the data-loading XML elements have attributes that allow you to specify database-related constructs based on the Ariba Query API. For example, you can added clauses to the query with the following attributes:

- fromClause
- incrementalClause
- orderByClause
- selectColumn
- whereClause

For more information on how to specify attributes to control your database queries, see “Database Attributes” on page 97.

Data Loads for Specific Versions

You create a data load definition with a <dataLoad> element similar to the following:

```
<dataLoad name="BuyerPOLineItem">
```

To extend this data load with a definition specific to a given version of the source system, use the <inDataLoad> element with the version attribute. For example:

```
<inDataLoad name="BuyerPOLineItem" version="7.1">
```

This new definition inherits from the default load, but any changes in the <inDataLoad> definition apply only to the Ariba Buyer 7.1 version of this load.

You can combine the version attribute with the disableLoad attribute to disable a data load for a specific version. For example, to disable any data load definitions related to Ariba Contract Compliance for a Ariba Buyer 7.1 source system:

```
<inDataLoad name="BuyerContract" version="7.1" disableLoad="true">
```

Sourcing Stage

You use the <sourcingStage/> element to specify data read from an Ariba Enterprise Sourcing database. Ariba Enterprise Sourcing uses database views to define the exact fields it sends to Ariba Analysis. These views are fixed, and cannot be modified from Ariba Analysis. When you define a sourcing stage, you cannot specify the fields to be extracted. All available fields are returned.

Ariba Enterprise Sourcing defines data loading views for the following Ariba Analysis facts and dimensions. You specify these views in the fromClause of the <sourcingStage> element, as in this example:

```
<sourcingStage fromClause="RFXAward"/>
<analysisStage destinationName="ariba.analytics.fact.RFXAward"/>
```

Interface Table Stage

You use <interfaceSqlStage/> to create an intermediate staging table. You can use the staging table for operations such as merging two fields or joining data. The <interfaceSqlStage/> element itself declares that you are using an interface table:

```
<interfaceSqlStage/>
```

You then use <interfaceSqlMapping> elements to define operations performed in that staging table. For example, to concatenate two fields:

```
<interfaceSqlMapping selectColumn="FirstName || ' ' || LastName"/>
```

From an extension file, you can add a new staging table, and then define mappings based on it, or create additional mappings for an existing staging table.

Field Mappings

The <analysisMapping> element defines a set of data transformations, which you specify with the <mapValue> element. The data transformations are applied before the value is stored into the destination field in Ariba Analysis.

The overall structure is as follows:

```
<field name="ContractLevel">
  <aqlMapping selectField="TermType"/>
  <analysisMapping>
      <mapValue implementation="ariba.analytics.mapValue.Decode">
          <parameter .../>
```

The <mapValue> element specifies a Java class that performs transformations on the data. The package ariba.analytics.MapValue includes a variety of standard classes, such as SubString, Concatenation, and Decode (which performs transformations of fields). For a list of the classes provided in the default configuration, and the parameters recognized by each, see “<mapValue>” on page 91.

Extension Files

This section describes how to extend or modify the data loading definitions in the default configuration. For a complete description of the syntax of data-loading XML, see "Data-loading XML Reference" on page 73.

Elements for Extending Data Loading Definitions

To modify an existing `<dataLoad>` declaration, you use `<inDataLoad>` or `<derivedDataLoad>`:

- `<inDataLoad>` extends the original definition
- `<derivedDataLoad>` creates a copy of the original definition and modifies that copy

When you use `<inDataLoad>`, you make a global change that affects any data loaded with that definition. When you use `<derivedDataLoad>`, you make a copy, and changes are isolated in that copy. For example, you should use `<derivedDataLoad>` if you want to load data from different sources that are similar, but have minor differences.

Modifying Stages

You can use the `<inAqlStage/>`, `<inSqlStage/>`, and `<inInterfaceSqlStage/>` elements to modify an existing stage. For example, you can append to a `FROM` clause or `INCREMENTAL` clause, or change from `select` to `select distinct`. For a list of the ways you can modify a stage, see "Modifying Existing Stages" on page 100.

Modifying Field Mappings

To change previously declared `<fieldMappings>`, you must first delete the field with the `<deleteField/>` element. After the field is deleted, you can create a new definition, with the `<field>` element.

The `<deleteField/>` element requires the `name` of the field to delete. For example:

```
<fieldMappings>
  <deleteField name="POCount"/>
  <field name="POCOunt">
    ...
  </field>
</fieldMappings>
```

Defining a Custom MapValue Class

In addition to using the standard map value implementations in the package `ariba.analytics.mapValue`, you can write your own Java class and supply it as the implementation.

▼ **To supply a custom mapValue implementation:**

1. Write a class that extends `ariba.analytics.loadMeta.MapValue`.
2. Put your compiled class or jar files in *`AnalysisServerRoot`*`/classes/extensions`, to ensure that it is included in the Ariba Analysis classpath. The directory structure should mirror the package structure. Thus, if your Java class is `MyMapper.class` in package `custom.analytics.mapValue`, your directory structure should be as follows:

   `classes/extensions/custom/analytics/mapValue/MyMapper.class`
3. Specify the full package name of your custom mapping class in the implementation attribute on the `<mapValue>` element, like this:

```
<mapValue implementation="custom.analytics.mapValue.MyMapper">
    ...
</mapValue>
```

Examples of Data Loading

This section includes examples of data-loading definitions. The first example illustrates loading from a CSV file, mapping the incoming data. The second example illustrates the use of an interface table.

CSV to Ariba Analysis with String Substitution

This example shows how to modify a supplier load to add a risk factor to all suppliers. This example assumes the existence of a spreadsheet that assigns a numeric rank to suppliers, such as `0`, `1`, and `2`.

The example in this section loads in the supplier risk data. It does the following:

1. Loads the fields `SupplierId` and `SupplierName` to fields of the same name in the `Supplier` dimension.
2. Uses the `Decode` class on `<mapValue>` to set up a mapping on the `Risk` `<field>` between the numeric risk rank and a textual description of the rank.

```
<!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>
```

CSV to Ariba Analysis with Interface Table

This example illustrates the use of an interface table in an Oracle database to concatenate the first and last names of users. Suppose you have a CSV file with the following columns.

UserID	First Name	LastName	SupervisorID
60009	Joe	Goodguy	69002
61244	Jane	Smith	72001
60121	Laura	Bush	666

...

The following example shows how to concatenate these columns using the Ariba Analysis database as a staging table:

```
<dataLoad name="UserLoad">
  <loadStages>
    <csvStage/>
    <interfaceSqlStage/>
    <analysisStage destinationName="ariba.analytics.dimension.UserData"/>
  </loadStages>
  <fieldMappings>
    <field name="UserId">
       <csvMapping selectField="UserId"/>
    </field>
    <field name="FirstName">
       <csvMapping selectField="FirstName"/>
       <interfaceSqlMapping insertColumm="FirstName"/>
    </field>
    <field name="LastName">
       <csvMapping selectField="LastName"/>
       <interfaceSqlMapping insertColumm="LastName"/>
    </field>
    <field name="UserName">
       /* Note that the following syntax is specific to Oracle */
       <interfaceSqlMapping selectColumn="FirstName || ' ' || LastName"/>
    </field>
    <field name="SupervisorId">
       <csvMapping selectField="SupervisorId"/>
    </field>
  </fieldMappings>
</dataLoad>
```

Chapter 6
Incremental Data Loading

This chapter describes *incremental data loading*, which is the process of updating only data that has changed, without having to reload the entire data set. It includes the following topics:

About Incremental Data Loading

Ariba Analysis supports two modes for data load: *full* and *incremental*. With full data load, all data from the data source is loaded, regardless of whether it has changed since the last load from that source. With incremental load, Ariba Analysis checks the source system to determine which records have been added, changed, or deleted since the last time you loaded that data, and then updates only those records that have changed. Incremental load can provide performance improvements, especially for large data sets.

The default mode is full. To use incremental mode, you must do the following:

- In the `<dataLoad>` definition for the event, include a clause that indicates which data to select.
- Run `initdb` with the `-incremental` option, to indicate that you want to do an incremental load.

The source system must be able to determine which data has changed. If you are loading data from a source system that does not track changes by date, you cannot use incremental mode.

Selecting Data

To use incremental load, you must specify how to select appropriate data from the data source. If you are loading from a database, with data-loading XML, you specify additional constraints clauses on the query that specify the data you want to extract. If you are loading from CSV files, you must manage the content of those files to ensure that they contain only incremental updates.

For database pulls, you use the attribute `incrementalClause` on the `<aqlStage>` and `<sqlStage>` elements to specify which data to select. The `incrementalClause` attribute specifies a clause that is `AND`ed to the `WHERE` clause of the database `SELECT` statement.

For example:

```
<aqlStage fromClause="ExpenseReport
   JOIN ExpenseItem USING ExpenseReport.LineItems
   LEFT OUTER JOIN ariba.common.core.User AS Requester
   whereClause="ExpenseReport.NextVersion IS NULL AND
   ExpenseReport.StatusString IN ('Approved', 'Processing')"
   incrementalClause="ExpenseReport.LastModified &gt;= :IncrementalStartDate
   AND ExpenseReport.LastModified &lt; :IncrementalEndDate"/>
```

This example uses the Ariba Query API parameters `IncrementalStartDate` and `IncrementalEndDate`. These values are determined from the `IncrementalParams` parameter, in `config/Parameters.table`. For example:

```
IncrementalParams = { Interval = 7; Type = Days;};
```

The `Type` specifies a unit such as days, week, months, or years. The `Interval` indicates an interval based on some number of that unit. Every time an incremental data load event is run, that event selects only data that has changed within the defined interval. For example, with the default settings (shown above), incremental load selects data that has changed in the last 7 days.

For complete information on the `IncrementalParams` parameter, and how to specify the beginning of a week period, see the *Ariba Analysis Configuration Guide*.

You can also define query parameters to make additional data available at the query level. For example, if you want to run a query based on the commodity code domain, you can use query parameters to make that value available in your `incrementalClause`. For information on how to use query parameters, see “Adding Query Clauses” on page 35.

Loading Data with Initdb

To take advantage of the `incrementalClause` definition, you must use the `-incremental` option to `initdb`. For example:

```
initdb -incremental -loaddb -file LoadDB.txt
```

In the input file for `initdb`, you can add qualifiers to indicate whether an event is run only on initial load, only on incremental load or both. The format is as follows:

```
mySourceSystem.DataLoad.BuyerLoad-Initial
mySourceSystem.DataLoad.BuyerLoad-Incremental
mySourceSystem.DataLoad.AnotherLoad
```

With the `incremental` option, `initdb` runs events with no qualifier, and also runs events defined with the `-Incremental` suffix.

In this example, the `initdb -incremental` command skips the first event (with the `-Initial` suffix) but runs the other two.

For complete information on the format of the `LoadDB.txt` file, see “Initialization Order Files (LoadDB.txt)” on page 103.

Chapter 7
Loading Common Data

The chapter discusses how to common data such as commodity codes, units of measurement (UOM) and currencies. It also describes how to configure or customize mappings between different naming conventions. It includes the following sections:

About Common Data

This chapter describes how to load common data, such as names of units of measure or currency. You typically load such data once, during database initialization.

In the default configuration, you load common data with Ariba File Channel events, which are listed in `LoadDBBasic.txt`. A typical entry in `LoadDBBasic.txt` is as follows:

```
None.IntegrationTask.ClassificationCodeMapPull
```

The keyword `IntegrationTask` indicates that this is an integration task that uses the Ariba File Channel. These events are defined with the Ariba File Channel configuration files `MessageConfiguration.table` and `MessageDefinition.table`.

In the default configuration, these events read common data from files in the `ariba` directory. If you need to modify or extend the default data, you can define supplemental data files in the `config` directory.

This guide does not describe how to configure or modify Ariba File Channel configuration files. If you find that you do need to customize or modify an integration task that uses the Ariba File Channel, consult the *Ariba Buyer Data Load Guide* for information.

Units of Measure

This section describes the integration tasks, parameters, and mappings that you use to define units of measure in your configuration.

Integration Tasks for Units of Measure

This section describes the integration tasks that you use to define unit of measure information. In the default configuration, this information is loaded with the following Ariba File Channel events, which are run from `LoadDB.txt`:

- UOMSystemPull
- UnitOfMeasurePull
- UOMSupplementalPull
- UOMConversionRatePull

UOMSystemPull

The `UOMSystemPull` integration task defines systems of units of measure, such as "metric." In the default configuration, this integration task reads data from the following file:

`ariba/variants/Plain/partitions/None/data/UOMSystem.csv`

This file contains just one column, which lists the unique name of a system of units of measure. For example:

```
Cp1252
"UniqueName"
"Metric"
"US"
```

If you require changes to this file, you can load a custom version from `config/variants/Plain/partitions/None/data/UOMSystem.csv`.

Note: Adding additional systems of units of measure can have negative performance impact, since each system of UOM requires an extra column in the database.

UnitOfMeasurePull

The UnitOfMeasurePull integration task defines the units of measure for a UOM system. In the default configuration, this task reads data from the following file:

ariba/variants/Plain/partitions/None/data/UnitsOfMeasure.csv

If you require changes to this file, you can load a custom version from config/variants/Plain/partitions/None/data/UnitsOfMeasure.csv.

The fields in UnitsOfMeasure.csv are as follows:

Field	Description
UniqueName	A unique identifier for a unit of measure. The UniqueName fields for each unit of measure must be distinct from one another.
Name	A name that corresponds to the UniqueName. The name is user-visible.
Description	Comments or descriptive text.
AllowNonWhole	A Boolean that indicates whether this unit of measure supports non-whole quantities. For example, a centimeter supports non-whole quantities, while a micron or an atom does not. If this field is missing, the default is false.

The following example shows sample lines from UnitsOfMeasure.csv:

```
06,smallspray,small spray,false,0
05,lift,lift,false,0
08,heat lot,heat lot,false,0
```

UOMSupplementalPull

The UOMSupplementalPull integration task specifies which unit of measure is preferred during conversions. For example, if you are using metric terminology, you must specify whether gram or kilogram or milligram is your preferred unit of weight. When Ariba Analysis converts incoming unit of measure data, it converts to the designated preferred unit.

In the default configuration, this integration task reads data from the following file:

config/variants/Plain/partitions/None/data/UOMSupplementalPull.csv

This data is in config, because there is no default standard for which units are preferred. You must define and configure this data load event for your configuration.

The format of UOMSupplementalPull.csv is as follows:

```
UniqueName,IsPreferred,UOMSystem
KGM,true,Metric
GRM,false,Metric
LBR,true,US
ONZ,false,US
```

The isPreferred Boolean indicates which unit is preferred within each system. In this example, kilogram is preferred over gram in the metric system and pound over ounce in the US system.

Conversion Rates

You use the UOMConversionRatePull integration task to specify conversions between different units of measure. In the default configuration, this event is run once, during initialization, with the task Task.UOMConversionRatePull-Initial, because unit of measure conversion rates do not change over time.

This integration task reads from the file UOMConversionRate.csv. In the default configuration, this file contains conversion rates for the default UOM systems. The format of this file is as follows:

```
FromUOM,ToUOM,Rate
KGM,GRM,1000
LBR,ONZ,16
LBR,KGM,0.45359237
```

Each line defines how to convert from the first UOM to the second, by multiplying. For example, a kilogram is defined as 1000 grams. The FromUOM and ToUOM fields can be from the same unit of measure system (as with grams or kilograms) or across systems (as with pounds and kilograms). When you are converting between rates in the same system, the conversions are used to convert into the preferred unit (as specified by your UOMSupplementalPull event).

Parameters for Units of Measure

In config/Parameters.table, you use the parameter UnitOfMeasureSystems to specify your preferred and secondary unit of measure systems. You must specify a primary system and can optionally specify any number of secondary systems. For example:

```
Application.Analysis.UnitOfMeasureSystems = {
   Primary = Metric;
   Secondary = ( US );
}
```

The values (Metric and US, in this example) must match the unique names of unit of measure systems, as defined by your UOMSystemPull event.

If you do not specify a secondary type is not specified, data is always stored in Ariba Analysis using the primary UOM type and is always displayed using that type. If you specify a secondary type, both values are stored in Ariba Analysis and users can set a preferred unit of measure type system in their preferences.

Conversion Maps for Units of Measure

If your source system uses different naming than your Ariba Analysis configuration, you must either pull mappings from your source system or define new ones in Ariba Analysis. The default configuration uses a mapValue implementation called ValueForMapType to do the mapping.

In your data-loading XML, you pass a parameter that indicates the source of the mapping. If you are loading data from Ariba Buyer, you can also pull mappings from Ariba Buyer. If you are loading units of measure from an ERP system, you must either load the mapping from Ariba Buyer or from a CSV file.

The default configuration supplies a set of data load definitions for mapping units of measure, which are located in the following file:

```
ariba/sourceTypes/Global/dataLoads/SourceSystemMapLoad.xml
```

If you require modifications to these default definitions, you can use them as samples to create your own definitions, in config.

▼ To use mappings from a source system:

1 In your data-loading XML file, there is a `mapValue` implementation for units of measure. For example:

```
<field name="UnitOfMeasure.UnitOfMeasureId">
    <aqlMapping selectField="UnitOfMeasure.UniqueName"/>
        <analysisMapping>
            <mapValue implementation=
                 "ariba.analytics.mapValue.ValueForMapType">
                <parameter name="MapType" value="SourceSystemUOM"/>
            </mapValue>
        </analysisMapping>
</field>
```

If you are loading the map from the source type, the `MapType` must be set to the value `SourceSystemUOM`, as shown here.

2 Load the map data with the event `Buyer.DataLoad.UOMMapLoad` event.

▼ To use mappings from an ERP system:

1 In your data-loading XML file, specify the following `mapValue` implementation:

```
<field name="UnitOfMeasure.UnitOfMeasureId">
    <aqlMapping selectField="UnitOfMeasure.UniqueName"/>
        <analysisMapping>
            <mapValue implementation=
                 "ariba.analytics.mapValue.ValueForMapType">
                <parameter name="MapType" value="ERP_UOM"/>
            </mapValue>
        </analysisMapping>
</field>
```

2 If you are loading the map from Ariba Buyer, use the `ERP_UOMMapFromBuyer` event. If you are loading the map load the map from CSV, use `ERP_UOMMapFromCSV` for units of measure instead, and put your mapping in the file `config/sourceTypes/Global/sourceSystems/`*`instanceName`*`/data/BuyerUOMMap.csv`. The format of a map file is as follows:

```
Cp1252
"Key","Value","Preferred","Comment"
```

Currencies

When you load data load from an external system such as Ariba Buyer, you can load information in any currency. During data loading, Ariba Analysis uses its conversion rates to convert incoming currencies into a currency supported by Ariba Analysis.

In the default configuration, the supported currencies for Ariba Analysis are USD, JPY, and EUR. You can define additional supported currencies, but be aware that each additional currency requires extra database overhead.

Integration Tasks for Currency

This section describes the integration tasks that you use to define currency information:

- CurrencyPull
- CurrencyGroupPull

CurrencyPull Integration Task

You use the CurrencyPull integration task to define a set of standard currencies. In the default configuration, this file defines ISO 4217 currency codes.

In the default configuration, the CurrencyPull task reads from the following file:

```
ariba/variants/Plain/partitions/None/data/Currency.csv
```

CurrencyGroupPull Integration Task

You use the CurrencyGroupPull integration task to set up different lead currencies for different users or currencies. The default configuration uses currency groups to ensure that the euro is the lead currency for users who are in EMU countries.

In the default configuration, CurrencyGroupPull reads from the following file:

```
ariba/variants/Plain/partitions/None/data/CurrencyGroup.csv
```

Currency Conversion Rates

The default configuration loads currency conversion rates by pulling them from Common Data Server. It uses the data loading definition CDSCurrencyConversionRate in the file CDSCurrencyLoad.xml to load the conversion rates as defined in CDS.

If you require additional conversion rates, you can define those conversion rates in a CSV file. Ariba Analysis provides a sample data load definition called CurrencyConversionRatesLoad, which is defined as follows:

```
CurrencyConversionRatesLoad = {
  DestinationName = ariba.analytics.core.CurrencyConversionRate;
  DataLoadName = "DirectCSVLoad";
  DataSourceParams = {
    Filename =
    "config/sourceTypes/Global/sourceSystems/buyer/data/CurrencyConversionRa
    te.csv";
    };
  DateFormat = "EEE MMM d k:m:s  yyyy";
  }
```

The fields in CurrencyConversionRate.csv are as follows:

Field	Description
UniqueName	An identifier for this currency rate.
FromCurrency	The currency you are using as the starting point.
ToCurrency	The currency you are using as the destination.
Rate	The currency rate, which must be a number such as ".632".
Date	The date on which this rate was entered, using the specified date format. For information on date formats, see "Date Formats" on page 109.

The following example shows a typical line in CurrencyConversionRate.csv:

```
"CAD","CAD","USD","0.787526","Wed Jan 20 00:00:00 2003"
```

Conversion Maps for Currency

This section describes how to set up mapping files for currencies. The process for mapping currencies is very similar to the process for mapping units of measure, as described in "Conversion Maps for Units of Measure" on page 55.

If your Ariba Analysis configuration loads data from an Ariba Buyer configuration that shares the same unit of measure map for several partitions, you can configure Ariba Analysis to load the maps by pulling from the source system. If you are loading data from an ERP system that has its own terminology for units of measure, you must load the mapping from Ariba Buyer or from a CSV file.

▼ To use mappings from a source system:

1 In your data-loading XML file, specify the following `mapValue` implementation:

```
<field name="Currency.CurrnencyId">
    <aqlMapping selectField="Currency.UniqueName"/>
        <analysisMapping>
            <mapValue implementation=
                 "ariba.analytics.mapValue.ValueForMapType">
                <parameter name="MapType" value="SourceSystemCurrency"/>
            </mapValue>
        </analysisMapping>
</field>
```

2 Load the map data with the `Buyer.DataLoad.CurrencyMapLoad` event.

▼ To use mappings from an ERP system:

1 In your data-loading XML file, specify the following `mapValue` implementation:

```
<field name="Currency.CurrnencyId">
    <aqlMapping selectField="Currency.UniqueName"/>
        <analysisMapping>
            <mapValue implementation=
                 "ariba.analytics.mapValue.ValueForMapType">
                <parameter name="MapType" value="ERPCurrency"/>
            </mapValue>
        </analysisMapping>
</field>
```

2 If you are loading the map from Ariba Buyer, use the `ERPCurrencyMapFromBuyer`. If you are loading from CSV, use `ERPCurrencyMapFromCSV`, and put your mapping in `config/sourceTypes/Global/sourceSystems/`*`instanceName`*`/data/BuyerCurrencyMap.csv`. The format of a map file is as follows:

```
Cp1252
"Key","Value","Preferred","Comment"
```

Commodity Codes

A *classification code* is a technique for classifying objects, by associating them with identifying codes. A *commodity code* is a specific type of classification code, used for classifying goods and services. Commodity codes are an important tool for data analysis, since accurate groupings of commodity codes can help you analyze your spend by commodity category.

There are a variety of emerging global standards for classification systems, such as UNSPSC (Universal Standard Products and Services Classification) and NAICS (North American Industry Classification System).

Because there are many possible commodity code systems, a commodity code is specified as a {domain,value} pair. The domain is the classification system, such as UNSPSC. The value is the code within that domain. Every Ariba Analysis configuration defines and uses one standard commodity code domain system. In the default configuration, the standard commodity code domain system is UNSPSC.

In Ariba Analysis, commodity codes are represented with the following dimensions:

- `ariba.analytics.dimension.Commodity`
- `ariba.analytics.dimension.ERPCommodity`

Codes in the `CommodityCode` dimension are global data, used by all Ariba Spend Management applications.

Codes in the `ERPCommodityCode` dimension are specific to one ERP system and typically represent data loaded from that ERP system. Most Ariba Spend Management applications use shared global commodity codes, but Ariba Analysis recognizes ERP commodity codes for situations when you prefer to do your analysis based on the codes used in a specific ERP system.

Integration Tasks for Commodity Codes

This section describes the data loading tasks that you use to define commodity codes and commodity code mapping information in your configuration.

CDSCommodityPull

To establish commodity code information in your configuration, you typically run the task `Task.CDSCommodityPull`. This task runs a group of events to pull commodity code and classification code map information from the Common Data Server.

ClassificationCodeMapPull

When you load commodity code information from an external system, those incoming codes are defined as domain/value pairs. If the incoming commodity codes use the same domain as your Ariba Analysis system, no mapping is required. However, if you are loading data from an external system that uses a different commodity code domain, you must ensure that the data load process maps from the incoming commodity code domain system into your preferred Ariba Analysis commodity code system.

When you pull commodity code information from the Common Data Server, the default configuration also pulls information about how to map classification codes between domains.

However, if your Ariba Analysis configuration pulls from a source system that uses an additional domain, such as an ERP-specific classification system, you must define your own mappings that describe how to map that domain into your system domain.

To define your own mappings, you use the `ClassificationCodeMapPull` task. This task reads from the file `ClassificationCodeMap.csv`. The format of the CSV file is as follows:

```
DomainFrom,ValueFrom,DomainTo,ValueTo
```

Parameters for Commodity Codes

If you are loading from an external system that uses a different domain for commodity codes, set parameters to specify the domain used on the source system:

- To specify the commodity code domain used by a specific source system, set the parameter `Application.SourceSystem.name.SystemCommodityCodeDomainName`
- If the source system is an ERP system, with its own set of codes, set the parameter `Application.SourceSystem.name.ERPCommodityCodeDomainName` to specify the domain used on that ERP system.

Mapping Commodity Codes

When loading incoming commodity codes, Ariba Analysis uses a `mapValue` class to translate incoming commodity codes into values that are defined in your configuration.

In the default configuration, Ariba Analysis maps incoming commodity codes to Ariba Analysis commodity codes as follows:

- Retrieves the domain from the source system and stores it in a temporary local field called `SourceCommodityDomain`. If there is no source domain, the default is to use the value specified as `Application.SourceSystem.name.SystemCommodityCodeDomainName`.
- Uses the `CommodityMap` map value implementation to look up incoming commodity codes and translate them into commodity codes in the system domain. For example:

```
<field name="Commodity.CommodityId">
    <aqlMapping selectField="CommodityCode.UniqueName"/>
    <analysisMapping>
       <mapValue implementation="ariba.analytics.mapValue.CommodityMap"/>
    </analysisMapping>
 </field>
```

  The `CommodityMap` first looks for a match on the commodity code field, using the domain defined as `SystemCommodityCodeDomainName` for that source system. If no mapping exists, it next looks for a match using the partitioned commodity code, which is stored in the field `ERPCommodity.CommodityId` field, and uses the `ERPCommodityCodeDomainName`.
- If no mapping can be computed, the commodity code is set to the empty string, which means the item is unclassified.

Chapter 8
Dumping Data

This chapter describes the configuration files that you use to define and run data load events. It includes the following sections:

About Data Dumps

Ariba Analysis provides data load tasks that you can use to write data from the Ariba Analysis database to files on your local disk. You can use these data load tasks for debugging, for backup before migration, or to create new data based on existing data. For example, you could dump your cost center data out to disk files, modify those disk files, and then reload the revised data into Ariba Analysis.

To use these classes, you create data load task entries in `DataLoadEvents.table`, and then run those events. The default configuration provides examples that illustrate how to dump dimensions, facts, reports, and user data (such as dashboards and folders).

Dumping Dimensions or Facts

This section describes how to use a data load task to dump a dimension.

Dumping Dimensions

▼ **To dump a dimension:**

1 Create a data load task such as the following:

```
DumpSupplier = {
    ScheduledTaskClassName = "ariba.analytics.tasks.DumpDimension";
    DimensionClass = "ariba.analytics.dimension.Supplier";
    Hierarchy = "SupplierType";
    Filename = "Supplier.csv";
};
```

For reference information on the `DumpDimension`, see "DumpDimension" on page 124.

2 Run the data load task, with a command such as the following:

```
bin/runtask -task DumpSupplier
```

This task dumps all data from the `SupplierType` hierarchy in the Supplier dimension, and writes it to the file `Supplier.csv`. The output format includes the following information:

- The lookup key for the dimension (`supplierID`, `supplierType`, and `sourcesystem`)
- The hierarchy, as derived from individual fields.
- The individual fields used to derive the hierarchy

To ensure that you do not edit the derived field, the column header for the derived field includes the comment [do not edit]. For example:

```
Supplier Hierarchy [do not edit]
```

3 Edit the output file `Supplier.csv`, using the editor of your choice.

Dumping Facts

To dump a fact, create a data load task that uses `ariba.analytics.util.DumpFact`. For an example, see "DumpFact" on page 125.

Importing and Exporting Data

This section describes tasks that you can use to export and import data such as users and reports to disk files. For example, if you are moving Ariba Analysis from development to production you might want to export the data from your development instance and then import it into the production instance.

Importing and Exporting User Data

You use data load tasks to import or export user data. The default configuration supplies sample tasks for import and export that both use the `ObjectEncoding` class, with appropriate parameters to indicate whether the data is being imported or exported. For example, the default configuration defines the following entries:

```
DumpUsers = {
    ScheduledTaskClassName = "ariba.analytics.tasks.ObjectEncoding";
    Operation = "Export";
    Directory = "dumpUsers";
    ObjectType = "User";
};

LoadUsers = {
    ScheduledTaskClassName = "ariba.analytics.tasks.ObjectEncoding";
    Operation = "Import";
    Directory = "sample/dumpUsers";
};
```

For full information on these tasks, see Appendix D, "Data Load Tasks."

Creating New Users with Import

In addition to using the user import and export commands for migration or upgrade, you can also create new users with the `LoadUsers` task. To do so, export one user first, and then edit the exported data in the XML-based definition file, or look at the sample definitions in *`AnalysisServerRoot`*`/sample/dumpUsers`.

Notes:

- Be careful when you edit the XML defining a user. Corrupted definitions cannot be loaded. You will have to correct them before you can successfully load.
- Specify a password for the user in the definition file in clear text. For example:

```
<Password>yourPasswordInCleartextHere</Password>
```

Importing and Exporting System Data

```
DumpSystem = {
   ScheduledTaskClassName = "ariba.analytics.tasks.SystemDump";
   Operation = "Export";
   Directory = "systemDump";
};

LoadSystem = {
    ScheduledTaskClassName = "ariba.analytics.tasks.SystemDump";
    Operation = "Import";
    Directory = "systemDump";
};
```

DumpSystem and LoadSystem create a SOAP-encoded representation of all user data and report definitions. You typically use `DumpSystem` is to capture all user data prior to migrating to a new release.

Importing and Exporting Reports

You can use the `DumpReport` task to extract report data from the database into CSV files. If you specify no arguments, it dumps all reports. You can also specify a username, and dump data only for that user. For example:

```
DumpReports = {
   ScheduledTaskClassName = "ariba.analytics.migration.DumpReport";
   User = "ashell";
   Directory = "dumpReports";
};
```

To import exported reports, create a task such as the following:

```
LoadReports = {
    ScheduledTaskClassName = "ariba.analytics.tasks.ObjectEncoding";
    ObjectType = Folder;
    FolderPath = "dumpReports";
};
```

Note: If you import an analytical report that contains data that is not present in the Ariba Analysis database, the associated construct is removed silently from the analytical report. Ariba Analysis writes warning messages to the loading log to indicate which constructs it removed.

Chapter 9
Troubleshooting

This chapter provides techniques for debugging and solving common data loading problems. It includes the following sections:

Logging for Data Loading

To debug data loading problems, use following log categories:.

Category	Description
`analysisMaster.analysisDataLoad`	High-level logging
`analysisMaster.analysisLoadDetails`	Low-level logging

To obtain high-level information about data loads, enable the `analysisDataLoad/debug` category for the `MainLogFile` logger in `config/Parameters.table`. This log category provides statistics on wait time, SQL queries run, and database operations such as dropping tables, creating statistics, or building indexes. This log category also provides summary statistics at the end of each data load.

Use the `analysisLoadDetails` category to debug specific load problems with individual data records. This log category generates extensive logging, and can cause OutOfMemory errors, so you should use it with caution and only to debug known issues.

For more information about logging categories and using the Ariba Analysis administration console to enable them, see the *Ariba Analysis Configuration Guide*.

Performance Tuning

This section summarizes parameters and settings for performance tuning.

For more detailed information on performance tuning of data loading, see the performance-tuning white paper on http://connect.ariba.com/.

Caches For Data Loads

The Ariba Analysis server uses two caches to improve data loading performance: a *dimension cache* and an *object cache*.

Dimension Cache

The following parameters control the dimension cache:

- System.Analysis.DataLoading.DimensionCache.LRUStartSize
- System.Analysis.DataLoading.DimensionCache.CacheSize
- System.Analysis.DataLoading.DimensionCache.CacheSegments

Each dimension starts out with its own cache. When the cache size exceeds the specified LRUStartSize, the dimension is moved to an array of LRU caches. The CacheSegments parameter determines the number of LRU caches (hash tables) in the array. The CacheSize parameter specifies the maximum size of each cache in the array. Your smaller dimensions should fit completely in the cache (as specified with LRUStartSize); larger dimensions are cached in the array, using an LRU algorithm.

Object Cache

The following parameters control the object cache:

- System.Nodes.NodesDefault.ObjectCacheSize
- System.Nodes.NodesDefault.ObjectCacheSegments

The *object cache* is the collection of Ariba objects retained in a cache. As users access objects in the database, Ariba Analysis keeps a cache of those objects in memory to improve performance. The object cache is segmented, for multi-threaded access, and is purged when it becomes full.

ObjectCacheSize specifies the cache segment size (the number of elements in each segment). ObjectCacheSegments specifies the number of segments in the cache.

The number of segments in the cache also determines how many concurrent threads can access the cache simultaneously.

To tune the object cache, you typically adjust the number of cache segments (the `ObjectCacheSegments` parameter). You should not change the object cache size. For guidelines and information about the best settings for these parameters, see the performance-tuning white paper on http://connect.ariba.com/.

Ordering of Data Loads

Ideally, all data loaded into Ariba Analysis is valid data, and there is only one record per lookup key. However, if you do have data that has multiple records with the same lookup key, you should order your data load so that the correct record is loaded last, if possible.

For example, the default configuration orders records with the following order by clause:

```
orderByClause="Requester.UniqueName,Requester.Active"
```

Ordering by `Requester` sorts the records so that records with the same lookup key are loaded together, and ordering by the `Active` field ensures that Active records are loaded last.

Thread Setting

In `DataLoadEvents.table`, you can specify the number of threads used for each data load event. If your CPU is not being fully used during data loading, and you have additional CPU available, you can change the number of threads to a higher value.

Transaction Batch Size

The `System.Analysis.DataLoading.BatchSize` parameter in `config/Parameters.table` specifies the number of database records written to or updated in the Ariba Analysis database with a single commit. The default is 20 records.

Data Validation

Ariba Analysis uses materialized views to enable query optimizations. For those materialized views to be created correctly, Ariba Analysis validates your data and enforces certain constraints to ensure that the data can be rolled up correctly.

When you load new data, the `DBOLAPProcessing` task runs validity checks to ensure that the data is valid for the specified database constraints. The behavior of this task is slightly different for different databases, but it runs database checks to report if there are invalid rows in the table, and what relationship is failing. To further diagnose problems, you should run the `diagnosedim` command.

If the data validation tasks report errors, it is essential that you take the time to diagnose and correct those errors. If you do not, your materialized views cannot be created correctly, and performance will degrade considerably.

The diagnosedim Command

The `diagnosedim` command runs diagnostic checks, validating the integrity of the data in one or more dimensions. For example, if the `DBOLAPProcessing` task reports data validation errors after loading new data, you can use `diagnosedim` to determine the exact records with errors.

The syntax is as follows:

```
bin/diagnosedim [-class ariba.analytics.dimension.dimension_name]
```

With no options, `diagnosedim` evaluates all dimensions. With the `-class` argument, it evaluates only the specified dimension. For example, to diagnose problems with only the single dimension `UserData`:

```
bin/diagnosedim -class ariba.analytics.dimension.UserData
```

The `diagnosedim` command prints messages that you can use to identify the exact records in the data source whose values must be corrected.

Common Errors

This section reports two common kinds of errors reported by `diagnosedim`, which are lookup keys that map to different values and parent-child relationships that are circular or inconsistent.

Lookup Key Errors

It is an error if you have multiple records with the same unique key and different text names. As a simple example of such an error, consider the following data:

User	Supervisor UniqueName	Supervisor Name
User1	Sup1	Janice Benning
User2	Sup1	John Benning

This data is invalid, because there are two different values for the unique name `Sup1`. The error message in this case has the following format:

The lookup key %s maps to more than one value in level %s

Errors in Parent-Child Relationships

The `diagnosedim` command reports an error if a given record has multiple parent records. As a simple example of this error, consider the following data:

User	First-Level Supervisor	Second-level Supervisor
User1	Sup1	Sup2
User2	Sup1	Sup3

In this data, `Sup1` reports to `Sup2` in the first row and `Sup3` in the second row. The error message in this case has the following format:

The lookup key %s in level %s has more than one parent in level %s

Oracle Database Constraints

A *database constraint* is a declaration of a condition in the database that must remain true. Ariba Analysis defines a number of database constraints, which are enforced when you insert or update data.

The error message for a database constraint violation has the following format:

```
(analytics:warning:5025): The data row [CostCenterId = 2200, CompanyCode =  ,
SourceSystem = buyer_america] in dimension DIM_COST_CENTER is invalid due to
[ATTRIBUTE]
```

Similarly, the error message for a child of constraint has the following format:

```
(analytics:warning:5025): The data row [CostCenterId = 2210, CompanyCode =  ,
SourceSystem = buyer_america] in dimension DIM_COST_CENTER is invalid due to
[CHILD OF]
```

If you have data that results in database constraint errors, you should run `diagnosedim` to get more information about the source of the errors.

Note: Running a data load event with multiple threads can sometimes cause reports of unique constraint violation errors. These exceptions can safely be ignored.

Appendix A
Data-loading XML Reference

This appendix provides complete reference information on the elements and attributes of Ariba Analysis data-loading XML. It includes the following sections:

Data-loading XML Elements

The following is an alphabetical list of all elements of Ariba Analysis data-loading XML.

<allDataLoads>

		Required?	Default value
Contains			
	<dataLoad>		

The <allDataLoads> element is the outermost tag in every data load definition XML file. For example:

```
<!DOCTYPE allDataLoads SYSTEM ../../ariba/analytics/core/dataLoads.dtd”>
<allDataLoads>
    <dataLoad name="someName">
        <loadStages>...</loadStages>
        <fieldMappings>...</fieldMappings>
    </dataLoad>
</allDataLoads>
```

<analysisMapping>

		Required?	Default value
Attributes			
	enabled	No	(none)
Contained in			
	<field>		
Contains			
	<mapValue>	No	(none)

The <analysisMapping> element defines a set of data transformations, specified with the <mapValue> element. The data transformations are applied before the value is stored into the destination field in Ariba Analysis.

The overall structure is as follows:

```
<field name="ContractLevel">
  <aqlMapping selectField="TermType"/>
  <analysisMapping>
      <mapValue implementation="ariba.analytics.mapValue.Decode">
          <parameter .../>
```

enabled Attribute

The optional enabled attribute is a Boolean that specifies whether the incoming data is persisted to the Ariba Analysis database. If set to true, the field is saved to the database. If set to false, the value is used as temporary storage and not persisted to the database. The default is true.

For example:

```
<field name="SourceCommodityDomain">
  <aqlMapping selectField="CommodityCode.Domain"/>
  <analysisMapping enabled="false"/>
</field>
```

<analysisStage/>

		Required?	Default value
Attributes			
	destinationName	Required	(none)
Contained in			
	<loadStages>		

Every data load definition must contain a <analysisStage/> element. It specifies a dimension class or fact class as the destination for the data being loaded.

destinationName Attribute

The required destinationName attribute specifies the dimension class or fact class to which Ariba Analysis loads the data. For example:

```
<analysisStage destinationName="ariba.analytics.dimension.Commodity">
```

<aqlMapping/>

		Required?	Default value
Attributes			
	isDynamicField	No	(none)
	groupBy	No	(none)
	selectField	Required	(none)
	sourceLookupField	No	
Contained in			
	<field>		

The <aqlMapping/> element specifies a mapping between a source field and the corresponding target field in Ariba Analysis.

isDynamicField Attribute

The optional isDynamicField attribute on the <aqlMapping> element is used in conjunction with dynamicFieldName and dynamicFieldValue attributes. For more information, see “dynamicFieldName and dynamicFieldValue Attributes” on page 77.

groupBy Attribute

The optional groupBy attribute on the <aqlMapping> element specifies the grouping order for the selected data. For example:

```
<field name="ConfirmationTime">
   <aqlMapping selectField="OrderConfirmation"
      groupBy="PurchaseOrder.OrderedDate"/>
</field>
```

This element is used in conjunction with the groupByFlag attribute on the <aqlStage/> element. See "<aqlStage/>" on page 77.

selectField Attribute

The selectField attribute specifies a field in the source, which selects the field that is to be stored into the destination field declared in the name attribute for this <field>. The value of selectField can be a column name in dot notation. For example:

```
<fieldMappings>
   <field name="OrderID">
      <aqlMapping selectField="PurchaseOrder.OrderID"
                  sourceLookupField="OrderID"/>
   </field>
</fieldMappings>
```

sourceLookupField Attribute

The sourceLookupField attribute modifies the selectField attribute by specifying an explicit lookup key. For example:

```
<fieldMappings>
   <field name="UserId">
   <aqlMapping selectField="Requester.UniqueName"
               sourceLookupField="UniqueName"/>
   </field>
</fieldMappings>
```

<aqlStage/>

		Required?	Default value
Attributes			
	distinctFlag	No	false
	dynamicFieldName	No	(none)
	dynamicFieldValue	No	(none)
	extraClause	No	(none)
	fromClause	Required	(none)
	groupByFlag	No	false
	incementalClause	No	(none)
	orderByClause	No	(none)
	whereClause	No	(none)
Contained in			
	<loadStages>		

The <aqlStage/> element defines a data load that uses an Ariba Buyer or Ariba Category Management database for the source stage. For loading from Ariba Enterprise Sourcing, see "<sourcingStage/>" on page 94.

dynamicFieldName and dynamicFieldValue Attributes

The optional dynamicFieldName and dynamicFieldValue attributes apply when the data source can potentially include a vector of values, but the specific values are not known. For example, a purchase order from Ariba Buyer that includes service line items for contractor services can include a vector of attributes that are set by the supplier, and not defined explicitly in the object model.

The following excerpt shows the use of these attributes:

```
<aqlStage fromClause="ariba.purchasing.core.PurchaseOrder
    JOIN ....
    LEFT OUTER JOIN ariba.base.core.NamedPair AS OrderAttributes
         USING LineItemProductDescription.OrderAttributes
    whereClause="..."
    orderByClause="PurchaseOrder.UniqueName, POLineItem.NumberInCollection,
SplitAccounting.NumberInCollection"
    dynamicFieldName="OrderAttributes.Name"
    dynamicFieldValue="OrderAttributes.Value"/>
```

In this example, OrderAttributes is a join of a vector of {name,value} pairs. To save a specific field, by matching the name attribute, the <aqlMapping> element must define a field with the isDynamicField attribute. For example, to store an attribute that matches the name region:

```
<field name="Region.RegionId">
  <aqlMapping selectField="region" isDynamicField="true"/>
</field>
```

<csvMapping/>

		Required?	Default value
Attributes			
	selectField	Required	(none)
Contained in			
	<field>		

The <csvMapping/> element specifies a field in the source data file that maps to its corresponding target field in Ariba Analysis. The <csvMapping/> element is the field mapping counterpart to the source stage <csvStage/>.

selectField Attribute

The selectField attribute specifies a field in the source CSV file that Ariba Analysis inserts into the destination field. For example:

```
<fieldMappings>
  <field name="CompanyId">
    <csvMapping selectField="CompanyId"/>
  </field>
<fieldMappings>
```

<csvStage/>

		Required?	Default value
Contained in			
	<loadStages>		

The <csvStage/> element specifies data read from a CSV file. For example, to use <csvStage/> to load data into the Commodity hierarchy:

```
<loadStages>
   <csvStage/>
   <analysisStage destinationName="ariba.analytics.dimension.Commodity"/>
</loadStages>
```

The location of the file containing the data is defined with the Filename parameter in the DataLoadEvents.table entry. For more information, see "Loading from a CSV File" on page 32.

To specify the relationship between fields in the CSV file and their corresponding fields in the target, use a <csvMapping> element.

<dataLoad>

		Required?	Default value
Attributes			
	name	Required	(none)
Contains			
	<loadStages> <fieldMappings>		

The <dataLoad> element defines a named data load definition. Each <dataLoad> element specifies one or more source stages, one destination stage, and an optional interface table stage for intermediate processing.

name Attribute

The required name attribute specifies the name of the data load definition. For example:

```
<dataLoad name="BuyerCommodity" ...>
```

<deleteField/>

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

The <deleteField/> element appears in extension files and is used to delete a <field> declared in an existing field mapping. To redefine a mapping, use <deleteField/> and follow it with <field> declaration of the same name.

name Attribute

The required name attribute specifies the name of the field you are deleting.

```
<fieldMappings>
  <deleteField name="Customer"/>
  <field name="Customer">
    ...
  </field>
</fieldMappings>
```

<derivedDataLoad>

		Required?	Default value
Attributes			
	extendsLoad	Required	(none)
Contained in			
	<allDataLoads>		
Contains			
	<inLoadStages> <fieldMappings>		

The <derivedDataLoad/> element makes a copy of a previously defined <dataLoad/> and defines extensions to the copy. To override data load definitions from a a previously defined <dataLoad/> (rather than extending them), use <inDataLoad>.

extendsLoad Attribute

The required extendsLoad attribute on <derivedDataLoad> specifies the name of the data load definition you are extending.

```
<allDataLoads>
  <dataLoad name="BasicSAPLoad">
    ...
  </dataLoad>
  <derivedDataLoad extendsLoad="BasicSAPLoad">
    ...
  </derivedDataLoad>
</allDataLoads>
```

<field>

		Required?	Default value
Attributes			
	name	Required	(none)
Contains			
	<csvMapping> <aqlMapping> <sqlMapping> <interfaceSqlMapping> <analysisMapping>		(none)
Contained in			
	<fieldMappings>		

The <field> element defines a mapping between a field in the data source and the target field in Ariba Analysis. The <field> contains a <csvStage> element if the data source is a CSV file, a <sqlStage> element if the data source is a database, and so forth. For example:

```
<fieldMappings>
  <field name="OrderedDate">
      <aqlMapping selectField="PurchaseOrder.OrderedDate"/>
  </field>
  <field name="NeedByDate">
      <aqlMapping selectField="Requisition.NeedBy"/>
  </field>
  ...
```

name Attribute

The required `name` attribute on the `<field>` element specifies a field in the target, indicating the field to be set, which can be an Ariba Analysis dimension or fact or an intermediate field in an interface table. For example:

```
<fieldMappings>
    <field name="CompanyId">
```

<fieldMappings>

		Required?	Default value
Contains			
	`<field>`	Required	(none)
Contained in			
	`<dataLoad>`		

The `<fieldMappings>` elements defines relationships between data in the source to their destination in Ariba Analysis. The `<fieldMappings>` element contains `<field>` elements. Each field element defines how to populate one specific field in Ariba Analysis.

<inAqlStage/>

		Required?	Default value
Attributes			
	appendToExtraClause	No	(none)
	appendToFromClause	No	(none)
	appendToWhereClause	No	(none)
	distinctFlag	No	(none)
	prependToFromClause	No	(none)
	replaceToExtraClause	No	(none)
	replaceToFromClause	No	(none)
	replaceToWhereClause	No	(none)
Contained in			
	<inDataLoad> <derivedDataLoad>		

The <inAqlStage/> element declares that you are extending a previously defined <aqlStage/>. The modifying attributes are used by both <inAqlStage> and <inSqlStage>, and are described in "Modifying Existing Stages" on page 100.

<inDataLoad>

		Required?	Default value
Attributes			
	name	Required	(none)
	version	No	
	disableLoad	No	false
Contained in			
	<allDataLoads>		
Contains			
	<inLoadStages> <fieldMappings>		

The <inDataLoad/> element declares that you are extending a previously defined <dataLoad/>.

The <inDataLoad> element extends the original definition of a data load and is always maintained in a separate .xml file from the definition it extends, as opposed to <derivedDataLoad> (see page 80), which makes a copy of the original data load definition.

name Attribute

The required name attribute specifies the name of the data load definition you are modifying. For example:

```
<inDataLoad name="BuyerPOLineItem" ...>
```

version Attribute

The optional version attribute identifies a specific version of the Ariba application from which you are loading data. For example:

```
<inDataLoad name="BuyerPOLineItem" version="7.1">
```

This attribute defines a data load event that applies only to a specific version of the source application. You use this attribute only if you are integrating with multiple versions of the same application (such as Ariba Buyer), which is rare.

disableLoad Attribute

The optional disableLoad attribute is used in conjunction with the buyerVersion to disable a data load event for a specific version. For example:

```
<inDataLoad name="BuyerPOLineItem" version="7.1" disableLoad="true">
```

The disableLoad attribute is a Boolean. The default value is false.

<inInterfaceSqlStage/>

		Required?	Default value
Attributes			
	appendToExtraClause	No	(none)
	appendToFromClause	No	(none)
	appendToIncrCLause	No	(none)
	appendToWhereClause	No	(none)
	distinctFlag	No	(none)
	replaceExtraClause	No	(none)
	replaceFromClause	No	(none)
	replaceIncrClause	No	(none)
	replaceWhereClause	No	(none)
Contained in			
	<inDataLoad> <derivedDataLoad>		

The <inInterfaceStage/> element declares that you are extending a previously defined <interfaceStage/>. See “<interfaceSqlMapping/>” on page 87.

<inLoadStages/>

		Required?	Default value
Contains			
	<inAqlstage> <inAnalysisStage> <inInterfaceSqlStage> <inSqlStage>		(none)
Contained in			
	<inDataLoad> <derivedDataLoad>		

The <inLoadStages> element modifies source stage definitions originally declared in a <dataLoad> element.

Note: You cannot modify a <csvStage>.

<inSourcingStage>

		Required?	Default value
Attributes			
	appendToFromClause	No	(none)
	appendToIncrCLause	No	(none)
	prependToFromClause	No	(none)
	replaceFromClause	No	(none)
	replaceIncrClause	No	(none)
Contained in			
	<inDataLoad> <derivedDataLoad>		

The <inSourcingStage/> element declares that you are modifying a previously defined <sourcingStage/>.

<inSqlStage>

		Required?	Default value
Attributes			
	appendToExtraClause	No	(none)
	appendToFromClause	No	(none)
	appendToIncrCLause	No	(none)
	appendToWhereClause	No	(none)
	distinctFlag	No	(none)
	prependToFromClause	No	(none)
	replaceExtraClause	No	(none)
	replaceFromClause	No	(none)
	replaceIncrClause	No	(none)
	replaceWhereClause	No	(none)
Contained in			
	<inDataLoad> <derivedDataLoad>		

The <inSqlStage/> element declares that you are extending a previously defined <sqlStage/>.

<interfaceSqlMapping/>

		Required?	Default value
Attributes			
	columnSize	No	(none)
	insertColumn	No	(none)
	selectColumn	No	(none)
Contained in			
	<field>		

The <interfaceSqlMapping/> element specifies the mapping between a column in an interface table (the data source) and its corresponding target field in Ariba Analysis.

If you do not specify the <interfaceSqlMapping/> element, but you have an <interfaceSqlStage>, the default is to insert fields into the interface table automatically, using column names that match the field names in the data source. You use <interfaceSqlMapping/> if you want to selectively ignore certain fields or to insert temporary columns into the interface table.

selectColumn Attribute

The selectColumn attribute specifies a select statement that is used to select a column from the data source and insert that value into the destination field in Ariba Analysis. If you use this attribute, Ariba Analysis populates the target field with the results of the SELECT. No value is inserted into the column in the interface table.

The value is a SQL SELECT statement (without the SELECT keyword). For example:

```
<fieldMappings>
  <field name="POId">
    <aqlMapping selectField="CustCode">
  </field>
  <field name="POCOunt">
    <interfaceSqlMapping selectColumn="(1/count(*) over (partition by
    POId))"/>
  </field>
</fieldMappings>
```

insertColumn Attribute

The `insertColumn` attribute defines the name of a column that is to be inserted into the interface table. You usually insert a column in the interface table when you need to perform some calculation based on the inserted column but do not want the inserted column to ultimately be loaded into Ariba Analysis.

For example, the following inserts a column named `Temp`:

```
<fieldMappings>
  <field name="POCOunt">
    <interfaceSqlMapping insertColumn="Temp"/>
  </field>
</fieldMappings>
```

columnSize Attribute

By default, all columns created in the interface table have 256 characters. You can use the `columnSize` attribute on the `<interfaceSqlMapping/>` element to specify a larger size. The following example sets the column size to 512 characters:

```
<fieldMappings>
  <field name="POId">
    <aqlMapping selectField="CustCode"/>
  </field>
  <field name="POCOunt">
    <interfaceSqlMapping insertColumn="Temp" columnSize="512"/>
  </field>
</fieldMappings>
```

<interfaceSqlStage/>

		Required?	Default value
Attributes			
	distinctFlag	No	false
	extraClause	No	(none)
	fromClause	No	(none)
	groupByFlag	No	false
	orderByClause	Non	(none)
	whereClause	No	(none)
Contained in			
	<loadStages>		

The <interfaceSqlStage/> element defines a new interface table, used for intermediate processing before the data is loaded into Ariba Analysis. If you use an <interfaceSqlStage>, you must use the <analysisStage> to define mappings from columns in the interface table to their destination fields in Ariba Analysis.

To refer to the interface table (for example, to select from it), use the following reserved word:

```
:InterfaceTable
```

For example:

```
<interfaceSqlStage fromClause=":InterfaceTable PO, Invoice"
whereClause="PO.POid (+)= Invoice.InvoiceID"/>
```

<loadStages>

		Required?	Default value
Contains			
	<csvStage> <aqlStage> <sqlStage> <interfaceSqlStage> <analysisStage>		(none)
Contained in			
	<dataLoad>		

The <loadStages> element is contained in <dataLoad>, and defines the data loading stages for one specific data load.

<loadStages> must always contain at least one source stage and the <analysisStage>. A source stage can be any of the following:

Stage Element	Purpose
<csvStage>	Data source is a comma-separated value (CSV) file
<aqlStage>	Data source is an Ariba application
<sqlStage>	Data source is a database, but not an Ariba application database

The <analysisStage> element specifies the fact or dimension class to which Ariba Analysis loads the data.

The <loadStages> element also contain an optional <interfaceSqlStage> element, to define an intermediate staging table for operations such as joins.

<mapValue>

		Required?	Default value
Attributes			
	implementation	Required	(None)
Contained in			
	<analysisMapping>		

The <mapValue> element specifies a data transformation class applied to the value from the source field or column indicated by the <field> element. You can specify any number of <mapValue> definitions in a single data-loading definition.

implementation Attribute

The required implementation attribute is the name of an ariba.analytics.MapValue class. Commonly-used classes include the following:

ariba.analytics.MapValue	Description
Constant	Replace incoming data with a constant value.
Decode	Substitute values in source with different values.
ReplaceNull	Replace incoming data with null.
StringConcat	Prepend or append strings to incoming values.
SubString	Extract a substring from incoming value.
ValueForMapType	Map incoming data values to values in maps in the source system.

The <mapValue> element typically contains one or more <parameter> elements. Each <parameter> element specifies the value for a named parameter. For example:

```
<mapValue implementation="ariba.analytics.MapValue.Constant">
   <parameter name="Constant" value="some_constant_here"/>
</mapValue>
```

The name and value are usually strings, but can also be vectors.

Parameters for Constant

The Constant class replaces incoming data with a constant value. It recognizes just one parameter, Constant. For example:

```
<mapValue implementation="ariba.analytics.MapValue.Constant">
   <parameter name="Constant" value="some_constant_here"/>
</mapValue>
```

Parameters for Decode

The Decode class substitutes values in the incoming data with values you specify. It requires two parameters, mapKeys and mapElements and recognizes the optional parameters defaultValue and nullValue.

The mapKeys and mapElements parameters each contain vectors that specify a key and the corresponding value to substitute for that key.

```
<mapValue implementation="ariba.analytics.MapValue.Decode">
   <parameter name="mapKeys">
     <vector>
           <entry value="<some_source_key_here"/>
     </vector>
   </parameter>
   <parameter name="mapElements">
     <vector>
           <entry value="<some_target_value_here"/>
     </vector>
   </parameter>
   <parameter name="defaultValue" value="default_value_here"/>
   <parameter name="nullValue"
     value="value_to_substitute_for_incoming_null"/>
</mapValue>
```

To specify a default value, which Ariba Analysis substitutes for an incoming value when your mappings match, use the following parameter:

```
<parameter name="defaultValue" value="your_default_value_here"/>
```

If you do not include a defaultValue parameter, and an incoming value does not match any of your mappings, Ariba Analysis returns the incoming value.

To specify a value to which Ariba Analysis should map an incoming null, use a parameter named nullValue:

```
<parameter name="nullValue" value="value_to_substitute_for_incoming_null"/>
```

Parameters for ReplaceNull

The ReplaceNull class replaces null values with a specified default. The required parameter is defaultValue.

```
<mapValue implementation="ariba.analytics.mapValue.ReplaceNull">
     <parameter name="defaultValue" value="0"/>
</mapValue>
```

Parameters for StringConcat

The StringConCat class either prepends a string to the incoming value or appends a string to it, or both. It recognizes two parameters, Prepend and Append. At least one parameter is required.

```
<mapValue implementation="ariba.analytics.MapValue.StringConcat">
   <parameter name="Prepend" value="some_string_for_the_front"/>
   <parameter name="Append" value="some_string_for_the_back"/>
</mapValue>
```

Parameters for SubString

The SubString class extracts a string of a calculated length from the incoming value. It takes two parameters, BeginIndex and EndIndex, which are integers representing the start and ending character position (inclusive) of the substring to be extracted. BeginIndex is required. EndIndex is optional.

```
<mapValue implementation="ariba.analytics.MapValue.SubString">
   <parameter name="BeginIndex" value="some_beginning_index_integer"/>
   <parameter name="EndIndex" value="some_ending_index_integer"/>
</mapValue>
```

Parameters for ValueForMapType

The ValueForMapType class reads mapping information (such as for units of measure or currency) from the source system. For an example of how to use this class, see “Conversion Maps for Units of Measure” on page 55.

```
<mapValue implementation="ariba.analytics.mapValue.ValueForMapType">
   <parameter name="MapType" value="SourceSystemCurrency"/>
</mapValue>
```

<sourcingMapping/>

		Required?	Default value
Attributes			
	selectColumn	Required	(none)
Contained in			
	<field>		

The <sourcingMapping/> element specifies the field in an Ariba Enterprise Sourcing view that corresponds to target <field> in Ariba Analysis.

For more information, see "Sourcing Stage" on page 41.

<sourcingStage/>

		Required?	Default value
Attributes			
	fromClause	Required	(none)
Contained in			
	<loadStages>		

The <sourcingStage/> element specifies a stage for incoming data from an Ariba Enterprise Sourcing configuration.

fromClause Attribute

The fromClause attribute specifies the name of an Ariba Enterprise Sourcing view from which to extract the fields.

In the following example, data are extracted from the RFXAward view to be loaded into the RFXAward fact in Ariba Analysis:

```
<sourcingStage fromClause="RFXAward"/>
<analysisStage destinationName="ariba.analytics.fact.RFXAward"/>
```

The list of possible views is as follows:

View Name	Description	<analysisStage> Destination
BuyerUser	Names of users in the buying organization	ariba.analytics.fact.UserData
Commodities	Commodities in use in Ariba Enterprise Sourcing	ariba.analytics.dimension.Commodity
Currency	Currencies in use in Ariba Enterprise Sourcing	ariba.basic.core.Currency
CurrencyConversion	Conversion rates	ariba.basic.core.CurrencyConversionRate
RFXAward	Line-level RFX (Winning Bid)	ariba.analytics.fact.RFXAward
RFXAwardCommodity	Commodities associated with an awarded RFX	ariba.analytics.fact.RFXAward
RFXCommodity	Commodities associated with a particular RFX	ariba.analytics.fact.RFXSummary
RFXInvitedSupplier	Suppliers associated with a particular RFX	ariba.analytics.fact.RFXSummary
RFXSummary	Header RFX	ariba.analytics.fact.RFXSummary
RFXTemplates	Templates for creating RFXs	ariba.analytics.dimension.RFXTemplate
Scorecard	Supplier scorecards	ariba.analytics.fact.Scorecard
ScorecardCommodities	Commodities associated with a particular scorecard	ariba.analytics.fact.Scorecard
ScorecardKPI	Key performance indicators for a particular supplier scorecard	ariba.analytics.dimension.ScorecardKPI
Supplier	Common supplier	ariba.analytics.dimension.Supplier
SurveyCommodity	Commodities associated with a supplier survey	ariba.analytics.fact.SurveyResponse
SurveyQuestion	Supplier surveys	ariba.analytics.dimension.SurveyQuestion
SurveyResponse	Supplier responses	ariba.analytics.fact.SurveyResponse

<sqlMapping/>

		Required?	Default value
Attributes			
	groupBy	No	(none)
	selectColumn	Required	(none)
Contained in			
	<field>		

The <sqlMapping/> element specifies the mapping between a column in the source database and its corresponding target field in Ariba Analysis. The <sqlMapping/> element is the field mapping counterpart to the source stage <sqlStage/>.

groupBy Attribute

The optional groupBy attribute on the <sqlMapping/> element specifies the grouping order for the selected data. For example:

```
<field name="ConfirmationTime">
  <sqlMapping selectField="OrderConfirmation"
    groupBy="PurchaseOrder.OrderedDate"/>
</field>
```

This element is used in conjunction with the groupByFlag attribute on the <sqlStage/> element. See “<sqlStage/>” on page 97.

selectColumn Attribute

The required selectColumn attribute specifies a column in the source table that Ariba Analysis inserts into the destination field declared in the name attribute for this <field>. The value of selectColumn can be a dotted name to specify the exact field.

```
<fieldMappings>
  <field name="CustomerId">
    <sqlMapping selectcolumn="CustCode"/>
  </field>
</fieldMappings>
```

<sqlStage/>

		Required?	Default value
Attributes			
	distinctFlag	No	false
	extraClause	No	(none)
	fromClause	Required	(none)
	incrementalClause	No	(none)
	groupByFlag	No	false
	orderByClause	No	(none)
	whereClause	No	(none)
Contained in			
	<loadStages>		

The <sqlStage/> element specifies that data are stored in a database other than an Ariba application.

Database Attributes

When the data source is a database, the data-loading XML generates a query that extracts data from that database. This section describes the attributes you can use to tun the query that is generated. The attributes in this section can be used with either <aqlSource> or <sqlSource>.

The initial definition of a data-loading definition can include the attributes fromClause, whereClause, incrClause, distinctFlag, and extraClause, which generate FROM clauses, WHERE clauses, and SELECT DISTINCT statements. In extension files, you can use the corresponding replace-, append-, or prepend- attributes to replace or modify an existing FROM clause, WHERE clause, or SELECT DISTINCT statement.

For information about the Ariba Query API, see the *Ariba Spend Management Query API Guide*.

Note: You must use entity references to refer to special characters in most of the attributes. These characters include but are not limited to the following:

Character	Entity Reference
<	<
>	>
“	"

If you fail to use an entity reference where one is required, on start-up, the Ariba Analysis displays an error message informing you to correct the error.

distinctFlag Attribute

The optional `distinctFlag` attribute specifies that the underlying select statement uses `SELECT DISTINCT`, rather than `SELECT`. For example:

```
<aqlStage fromClause="CommodityCode" distinctFlag="true"/>
```

By default, `distinctFlag` is `false`.

extraClause Attribute

The optional `extraClause` attribute is deprecated.

fromClause Attribute

The `fromClause` attribute defines a clause added to the database `SELECT` statement used to query the data source. The value can be any valid string that can appear after the keyword `FROM` in a query. The `fromClause` usually specifies the name of the table from which the data is taken. The following example illustrates a `fromClause` for an Ariba Query API query:

```
<loadStages>
  <aqlStage fromClause="CommodityCode INCLUDE INACTIVE"/>
  <analysisStage destinationName="ariba.analytics.dimension.Commodity"/>
</loadStages>
```

The following example illustrates a fromClause for a SQL query:

```
<loadStages>
   <sqlStage fromClause="sw_customer"/>
   <analysisStage destinationName="ariba.analytics.dimension.Customer"/>
</loadStages>
```

In the <fieldMappings> section of the data load definition, you must have a corresponding <aqlMapping> or <sqlMapping> element with selectField attributes that specify the exact fields to extract from the source Ariba application and their corresponding fields in Ariba Analysis.

The fromClause attribute is required, but can be null.

groupByFlag Attribute

The optional groupByFlag attribute adds a GROUP BY clause to the underlying query. The value is a Boolean, false by default. To add a GROUP BY clause,

```
<aqlStage fromClause="ariba.purchasing.core.PurchaseOrder
   groupByFlag="true"/>
```

In the <fieldMappings> section of the data load definition, you must have a corresponding <aqlMapping> element with the groupBy attribute that specifies the field by which the data must be grouped. See “<aqlMapping/>” on page 75.

incrementalClause Attribute

The incrementalClause attribute allows you to load only updated data from the source Ariba application, rather than all data. The value of the incrementalClause attribute is ANDed to the WHERE clause of the database SELECT statement in the source Ariba application.

orderByClause Attribute

The value of the orderByClause attribute is appended to the database SELECT statement to sort the selected data. Column names are specified in dot notation, and can include multiple fields.

```
<loadStages>
   <aqlStage fromClause="ariba.purchasing.core.PurchaseOrder
      orderByClause="PurchaseOrder.UniqueName, POLineItem.NumberInCollection,
      SplitAccounting.NumberInCollection"
```

```
</loadStages>
```

whereClause Attribute

The optional whereClause attribute indicates a WHERE clause to be appended to the database SELECT statement. The value is a valid Ariba Query API clause. For example:

```
<aqlStage fromClause="CommodityCode" whereClause="Code &lt; 500"/>

<interfaceSqlStage fromClause="PO, Invoice" whereClause="PO.POid (+)=
Invoice.InvoiceID"/>
```

Modifying Existing Stages

When you modify an existing stage, with <inAqlStage/> or <inSqlStage>, you can use the following attributes.

Attribute	Description
appendToExtraClause	Appends its value to the extraClause on the original corresponding stage
appendToFromClause	Appends its value to the fromClause on the original corresponding stage
appendToIncrementalClause	Appends its value to the incrementalClause on the original corresponding stage
appendToWhereClause	Appends its value to the whereClause on the original corresponding stage
distinctFlag	Set to true or false to return distinct values or not. By default the originally declared value of distinctFlag is used.
prependToFromClause	Prepends its value to the fromClause on the original corresponding stage
replaceExtraClause	Replaces the extraClause on the original corresponding stage
replaceFromClause	Replaces the fromClause on the original corresponding stage
replaceIncrementalClause	Replaces the incrementalClause on the original corresponding stage
replaceWhereClause	Replaces the whereClause on the original corresponding stage

Appendix B
File Formats

This appendix describes the format of data files used in Ariba Analysis. It includes the following sections:

Initialization Order Files (LoadDB.txt)

An *initialization order file* specifies the order in which data is loaded during database initialization. An initialization order file is used as input to `initdb`. The default configuration supplies two such files, `LoadDB.txt` and `LoadDBBasic.txt`.

An initialization order file consists of a list of data load events and tasks. The order in the file is the order in which the events are run. Each entry has three parts, separated by period, in the following form:

`sourceSystem.keyword.eventName`

The following example illustrates typical lines:

```
Default.DataLoad.AmountRangeLoad-Initial
Task.UserManagementGroupSetup
None.IntegrationTask.ClassificationCodeMapPull
```

These entries are interpreted as follows:

- The first part names a source system. (In the case of data load tasks, the *`sourceSystem`* is not specified, since data load tasks are not source-system specific.)
- The second is a keyword: `DataLoad`, `Task`, or `IntegrationTask`.

- The third part is the name of the task or event. Data load events are defined in `DataLoadEvents.table`. Data load tasks are defined in `DataLoadTasks.table`. Integration tasks are defined in `MessageConfiguration.table`.

For data load events, the event name can include an optional qualifier. The valid values for the qualifier are as follows:

`-Incremental`	The event is run only if you use `initdb -loaddb -incremental`. For example: `Default.DataLoad.BuyerERLoad-Incremental`
`-Initial`	The data load event is not run if the `-incremental` option is present. For example: `Default.DataLoad.BuyerInvoiceLoad-Initial`

If no qualifier is present, the event is run regardless of whether the `-incremental` option is present.

DataLoadEvents.table

This section describes the parameters that can appear in an entry in `DataLoadEvents.table`. For general information on how to define and interpret entries in this file, see “Data Load Events” on page 31.

The general structure of an entry in `DataLoadEvents.table` is as follows.

```
dataLoadEventName = {
  DataLoadName = "some_name" ;
  DataSourceParams = {
    /* parameters specific to the data source ....*/
    }
  }
  DisableFactLookup = true ;
  Operation = either Load or Delete or Update ;
  Threads = numberOfThreads ;
  UnionWithDataLoads = “comma-separated names of data load definitions”
```

The *dataLoadEventName* must be an alphanumeric string, with no spaces.

General Parameters

The table below lists the parameters that can appear in an data source definition:

Parameter	Description of Value
DataLoadName	Refers to a <dataLoad> definition in a data-loading XML file, which specifies the data transformations to be applied before loading this data. For more information on <dataLoad> definitions, see "Data-loading XML" on page 37. The special reserved name DirectCSVLoad indicates a simple data loading, with no data-loading XML required.
DataSourceParams	Specifies parameters that define how to connect to the data source. The value is a nested table. For a list of the keys that can appear in the DataSourceParams entry, see "Data Source Parameters" on page 106.
DateFormat	Used only when reading dates. Specifies the expected format of dates. If not specified, the date format defaults to the value specified as System.Analysis.DataLoading.DateFormat. For information on valid date format specifications, see "Date Formats" on page 109.
DestinationName	Target fact or dimension. Used with DirectCSVLoad.
DisableFactLookups	Specifies whether Ariba Analysis looks up a fact before loading it. This is a performance-tuning parameter. If you know whether the data already exists in Ariba Analysis, you can use this parameter to specify whether lookup is required. (Load is faster if you don't need to do the lookup first.) The value is a Boolean. If the operation is Update or Delete, lookups are always enabled. If the event uses incremental load, lookups are always enabled. If the event is an initial load, with the Load operation, lookups are disabled by default. Set this parameter if you are sure whether an event requires lookup. The default is false (lookup is enabled by default)

Parameter	Description of Value
Operation	Specifies whether this event loads new data, updates existing data, or deletes data. The value must be one of the following: • Load . Creates new records or updates existing records • Delete. Deletes records that are in the data source. • Update. Updates existing records (based on their lookup keys) but does not create new records.
Threads	Specifies the number of threads used to run the data load event. Default is 1. The setting for most of the default events is 4.
UnionWithDataLoads	Used when selecting data from SQL or Ariba Query API. Specifies that the data is a union of data selected by other existing data load definitions, as declared with <dataLoad> elements in data-loading XML files. For example: **UnionWithDataLoads = "Event1,Event2";** In this example, the data being loaded is the union of the data defined by the following data-loading XML declarations: <dataLoadname="Event1"> <dataLoad name="Event2">

Data Source Parameters

The DataSourceParams key specifies parameters that define how to connect to the data source. The table below lists the parameters that can appear in DataSourceParams.

Parameter	Description of Value
BuyerConnection	An optional label from ConnectionInfo.table. This parameter is used only when you are overriding the default connection to an Ariba Buyer source system.
BuyerPartition	Specifies the name of an Ariba Buyer partition. This parameter is used with BuyerConnection, to specify an alternative connection to an Ariba Buyer configuration.
DropInterface Table	An optional parameter with a default of true. Set DropInterfaceTable to false to prevent the automatic deletion of an interface table when data loading is complete.
Filename	Path to CSV files, relative to *AnalysisServerRoot*.

Parameter	Description of Value
InterfaceSql Connection	An optional label from ConnectionInfo.table specifying the location for interface tables used during data loading. This parameter is used only when you are overriding the default interface table connection information in the source system.
Language	Specifies the language to be loaded, if the data source has multilingual data. The value can either be the name of a language, such as Japanese, or the special value PartitionDefault to indicate the default language of the partition from which you are loading.
Locale	The abbreviation for a locale specific to this data load event, such fr or de. Only needed if the locale for the loaded data is different from the source system. If no Locale parameter is present, the default is en_US.
QueryParams	Defines a hash table of parameters, which are passed through to the query layer so that the values can be used in constructing queries to select data from a database. The values specified in here override values specified in config/Parameters.table, with System.Analysis.DataLoading.QueryParams. For more information, see "Adding Query Parameters" on page 36.
SQLConnection	An optional label from ConnectionInfo.table specifying how to connect to the source database. This parameter is used only to override the default connection from the source system.

CSV Data File Format

This section describes the format of the CSV data file.

Character Encoding

The first line of the CSV file specifies the character encoding of the data in that file. The encoding can be any value supported by the Java VM. For example, the following line indicates that the character set is ISO 8859-1, also known as ISO Latin-1:

```
8859_1,,
```

If the first line of the file does not contain a character set specification, Ariba Analysis uses the default system character encoding, which is determined by the `config/Parameters.table` entry `System.Base.DefaultEncoding`.

Column Names in CSV Files

A column name in a `DirectCSVLoad` file must match a field name in the target dimension or fact. Ariba Analysis loads only those columns specified in the file, as long as the field names match. If there are fields in Ariba Analysis that do not have matching values in the CSV file, no data is loaded for those fields.

For example, the metadata XML definition of the `UserData` dimension defines the following three fields:

```
<dimension name="ariba.analytics.dimension.UserData">
  <levels>
    <field name="UserName">
       <type class="java.lang.String"/>
       <properties label="User"/>
    </field>
    <field name="UserId">
       <type class="java.lang.String" length="50"/>
    </field>
    <field name="SupervisorId">
       <type class="java.lang.String" length="50"/>
    </field>
  <levels>
</dimension>
```

The following example of a direct CSV data load file contains data matching the first two fields only. Because the CSV file does not contain a column labelled `SupervisorID`, no data are loaded into that field in Ariba Analysis:

```
"UserId","UserName"
"2","Lokesh Smith"
"3","Annabel Jones"
```

A CSV file can have comments that are ignored by Ariba Analysis. For example:

```
/* Ignore this column */

// Ignore this column
```

Date Formats

When you enter dates in configuration files, Ariba Analysis parses those dates as strings and then translates those strings into dates. The following table defines the format of dates.

Key	Explanation	Data Type	Example
G	era designator	(Text)	AD
y	year	(Number)	1996
M	month in year	(Text & Number)	July 07
d	day in month	(Number)	10
h	hour in am/pm (1~12)	(Number)	12
H	hour in day (0~23)	(Number)	0
m	minute in hour	(Number)	30
s	second in minute	(Number)	55
S	millisecond	(Number)	978
E	day in week	(Text)	Tuesday
D	day in year	(Number)	189
F	day of week in month	(Number)	2 (2nd Wed in July)
w	week in year	(Number)	27
W	week in month	(Number)	2
a	am/pm marker	(Text)	PM
k	hour in day (1~24)	(Number)	24
K	hour in am/pm (0~11)	(Number)	0
z	time zone	(Text)	Pacific Standard Time
'	escape for text	(Delimiter)	
''	single quote	(Literal)	'

The number of characters in each field indicates the degree of padding for that field. So, for example, `MMM` indicates three-digit month names (`Jul`, `May`); `MM` indicates month numbers padded to 2 digits (11, 04), and `M` indicates no padding (11, 4).

The rules for padding and number of digits are as follows:

- For text, a pattern of 4 or more pattern letters uses the full form. Less than 4 pattern letters uses the short or abbreviated form if one exists.
- For numbers, the pattern indicates the minimum number of digits. Shorter numbers are padded with zeros to fit that number of digits. For example, DD always pads to 2 digits. Year is handled specially: if the count of y is 2, the year is truncated to 2 digits.
- For a field that can be either a text or a number (such as month), the digit count indicates whether the value uses text or numbers. If the count is 3 or over, use text; otherwise, use a number. So `MMM` uses text for months and `MM` uses numbers.

The following table shows examples that use the `en_US` locale:

Example	Output
`"yyyy.MM.dd G 'at' hh:mm:ss z"`	2004.07.10 AD at 15:08:56 PDT
`"EEE, MMM d, ''yy"`	Wed, July 10, '04
`"h:mm a"`	12:08 PM
`"hh 'o''clock' a, zzzz"`	12 o'clock PM, Pacific Daylight Time
`"yyyyy.MMMMM.dd GGG hh:mm aaa"`	2004.July.10 AD 12:08 PM

For more information on date formats, see the Javadoc for the `java.text.SimpleDateFormat` class:

`http://java.sun.com/j2se/1.3/docs/api/java/text/SimpleDateFormat.html`

ConnectionInfo.table

This section describes the format of the configuration file `ConnectionInfo.table`, which defines a named set of connections to external systems. This file is optional. You use this file only if you want to override the default connection information, as defined in the source system entry in `config/Parameters.table`.

The following is an example of a ConnectionInfo.table file:

```
{
DBConnections = {
    sampleOracle = {
        DBType = Oracle;
        Driver = oracle.jdbc.driver.OracleDriver;
        Password = apps;
        URL = "jdbc:oracle:thin:@cent:1531:vis";
        User = apps;
;
    sampleDB2 = {
        DBType = DB2;
        Driver = COM.ibm.db2.jdbc.app.DB2Driver";;
        Password = sa;
        URL = "jdbc:db2:SERVER";";
        User = sa;
};
```

This example defines entries for two systems, sampleOracle and sampleDB2.

Parameters in ConnnectionInfo.table

The parameters that can appear in a named connection within ConnectionInfo.table are as follows:

Parameter	Explanation
DBType	Specifies the type of the underlying ERP system. The value must be Oracle, MSSQL or DB2.
Driver	JDBC driver used to access the database. The value depends on your chosen database.
Password	Database password.
URL	The URL Ariba Analysis uses to access the JDBC driver.
User	Database user.

Encrypting Parameters in ConnnectionInfo.table

You can encrypt values in `ConnectionInfo.table`. If you supply any encrypted values, you must define a `SecureParameters` key to specify which parameters are encrypted. For example:

```
{
DBConnections = {
    mySampleOracle = {
        DBType = Oracle;
        Driver = oracle.jdbc.driver.OracleDriver;
        Password = "{DES}PU45FU2Gxxx=";
        URL = "jdbc:oracle:thin:@cent:1531:vis";
        User = aribauser;
    };
    sampleDB2 = {
        DBType = DB2;
        Driver = COM.ibm.db2.jdbc.app.DB2Driver";;
        Password = "{DES}LG92JR2Pxxx="
        URL = "jdbc:db2:SERVER";";
        User = sa;
    };
};
SecureParameters = ( DBConnections.mySampleOracle.Password,
DBConnections.sampleDB2.Password);
};
```

The value of `SecureParameters` is a vector of parameter names. Any parameter whose name is listed in this vector is assumed to have an encrypted value.

Appendix C
Command Reference

This appendix provides reference information on the syntax of the commands referenced in this document. It describes the following commands:

The commands described here are located in *AnalysisServerRoot*/bin.

addsourcesystem

You use the addsourcesystem command to create an initial set of configuration files for a new source system. The addsourcesystem command bases each new source system on template files in the directory *AnalysisServerRoot*/configureOptions.

Syntax

The syntax of the addsourcesystem command is as follows (formatted on multiple lines for readability):

```
AnalysisServerRoot/bin/addsourcesystem \
    -configOption templateDirectory \
    -baseType templateType \
    -typeName sourceTypeName \
    -systemName nameOfSourceSystemToCreate
    [-connKey buyerKey -partition partitionName -version versionNumber]
```

Options

The addsourcesystem command has the following options:

-configOption [baseConfig | demoConfig]

Use this option to specify the set of template files to use. This option is required. The baseConfig option creates typical configuration for integration with an Ariba Spend Management application such as Ariba Buyer or Ariba Enterprise Sourcing. The demoConfig option creates a demo configuration that reads from CSV files and doesn't integrate with any live applications. If you specify demoConfig, you must have chosen to install the demonstration configuration files when installing Ariba Analysis.

-baseType *sourceType*

Use this option to specify a template sourceType to use for creating the new source type. This option is required. The value must name a directory within *AnalysisServerRoot*/configureOptions. Each such directory provides a sample set of configuration files for a given source type.

The argument you supply as *sourceType* must be one of the following.

Directory Name	Description
ACM	A sample integration with Ariba Category Management
buyer-csv	Ariba Buyer partition that reads from CSV files
buyer-none	Default Ariba Buyer partition
buyer-oracle	An Ariba Buyer and Oracle integration
buyer-psoft	An Ariba Buyer and PeopleSoft integration
buyer-sap	An Ariba Buyer and SAP integration
Global	Default source system. Every configuration must create one global source system.

-typeName *sourceTypeName*

Use this option to specify the name of your new sourceType subdirectory, which will be created within *AnalysisServerRoot*/config/sourceTypes. This option is required. For example: -typeName PSOFT

`-systemName` *`sourceSystemName`*

Use this option to specify the name of your new `sourceSystem` subdirectory, which will be created within *`AnalysisServerRoot`*`/config/sourceTypes/`*`typeName`*`/`. This option is required. For example: `-systemName psoft1`

`-connKey` *`key`*

Use this option to specify the name of a connection key, which must match the instance name of an Ariba Spend Management application in `AppInfo.xml`. This option is used to connect to that Ariba Spend Management application.

This option is required unless you are using `-baseType Global`.

`-partition` *`partitionName`*

Use this option to specify the name of a partition in the Ariba Spend Management application to which you are connecting.

Examples

This section shows examples of creating source systems.

Source Systems for Ariba Enterprise Sourcing

The following commands illustrate how to create a typical set of source systems for a configuration that integrates with Ariba Enterprise Sourcing:

```
addsourcesystem -configOption baseConfig -baseType Global -typeName Global
-systemName Default

addsourcesystem -configOption baseConfig -baseType sourcing -typeName sourcing
-systemName sourcing -connKey Sourcing -partition sourcing
```

Note: For an Ariba Enterprise Sourcing configuration, the `-partition` option should always be set to `sourcing`.

Source Systems for Ariba Category Management

The following commands illustrate how to create source systems for a configuration that integrates with Ariba Category Management:

```
addsourcesystem -configOption baseConfig -baseType Global -typeName Global
-systemName Default

addsourcesystem -configOption demoConfig -baseType ACM -typeName ACM
-systemName ACM -connKey ACM -partition supplierdirect
```

Note: For an Ariba Category Management configuration, the `-partition` option should always be set to `supplierdirect`.

diagnosedim

The `diagnosedim` command is a diagnostic command that prints diagnostic messages about one or more dimensions.

If `diagnosedim` reports errors, it is essential that you correct your data. For more information on when and how to use this command, see "Data Validation" on page 70.

Syntax

The basic syntax for `diagnosedim` is as follows.

```
bin/diagnosedim [-class ariba.analytics.dimension.dimension_name]
```

Options

`-class ariba.analytics.dimension.`*`dimension_name`*

Use this option to specify the name of a dimension. If you do not specify this argument, the `diagnosedim` command prints output information for all dimensions.

initdb

The `initdb` command initializes the Ariba Analysis database by running a specified set of data load tasks and data load events.

Syntax

The basic syntax for full database initialization is as follows.

```
initdb -initdb
```

Options

The `initdb` command takes the following options.

`-db2stats`

Creates database statistics for IBM DB2. This option is called from `initdb -initdb`. For more information on database statistics, see "ComputeDBStats" on page 123.

`-debug`

Runs `initdb` inside the Java debugger `jdb`

`-emptydb`

Drops and recreates all known tables. (This option does not work with Microsoft SQL Server.)

`-emptydbbuiltin`

Drops and recreates only built-in tables

`-file` *`file`*

Specifies the *`file`* from which to read load events. The *`file`* specification can be a single file or a comma-separated list of files. The location of *`file`* is relative to *`AnalysisServerRoot`*`/config`.

If you do not specify `-file`, the default value is `config/LoadDB.txt`.

`-incremental`

Runs data load events defined for incremental loading, but does not run data load events defined only for initial loading. Must be used with `-loaddb` option. For more information, see "Loading Data with Initdb" on page 49.

`-initdb`

The same as: `initdb -emptydb -loadmeta -loaddb`

If your database is IBM DB2, the command `initdb -initdb` automatically computes your database statistics with the DB2 command `REORGCHK`.

`-initdbbuiltin`

The same as: `initdb -emptydbbuiltin -loadmeta -loaddb`

`-loaddb`

Loads data by running data load events specified in the file `config/LoadDB.txt`. For more information, see "Data Load Events" on page 31.

`-loadmeta`

Reloads metadata from XML files

`-readfromdump` *`directory`*

Specifies the directory from which to read data load definitions previously written with the `-writetodump` option. Must be used with the `-loaddb` option.

`-reshapeDB`

Applies database changes based on changes in metadata XML files. This option updates the database schema after loading extrinsic metadata that affects the shape of the data, such as adding a new fact or dimension.

Changes that require the `reshapeDB` option include the following:

- Adding a new fact or dimension
- Adding a new field to an existing fact or dimension
- Creating an index on a table
- Adding the `unique` attribute to a field

`-writetodump` *`directory`*

Specifies a *`directory`* to which a copy of the data being loaded into the Ariba Analysis database is written. This directory can be used to load data again with the `-readfromdump` option. Must be used with the `-loaddb` option.

runtask

The `runtask` command executes a single data load task.

Syntax

The `runtask` command has the following syntax:

```
runtask -task taskName
```

For example:

```
runtask -task LoadFromStaging
```

Options

The `runtask` command recognizes the following options.

`-task` *`taskName`*

Runs a task defined in *`AnalysisServerRoot`*`/config/DataLoadTasks.table`.

Appendix D
Data Load Tasks

This appendix describes tasks related to data loading. The tasks in this appendix are those that are typically run from `initdb`, during data initialization. For information on scheduled tasks that are run on a regular schedule, see the *Ariba Analysis Configuration Guide*.

Data loading tasks are defined in the file `DataLoadTasks.table`, located in the *`AnalysisServerRoot`*`/config` directory.

This appendix lists the tasks in alphabetical order.

ApplyDataSources

```
ApplyDataSources = {
    ScheduledTaskClassName = "ariba.analytics.tasks.DataSourceMerger";
}
```

The ApplyDataSources task merges changes to dimensions, hierarchies, or levels that a user has made from the administration console's **Data Sources** tab. Until you run this task, the edits are in a temporary database table.

Analysis users who want to override the data loaded from source applications use the **Data Sources** tab to create their additional data. Since this data is meant to override the existing data, you should run this task only after the initial data load tasks are complete.

Compute*<Fact>*Count

```
ComputePOLineCount = {
    ScheduledTaskClassName = "ariba.analytics.tasks.ComputeFractionalCount";
    FactClass =  "ariba.analytics.fact.POLineItem";
    FractionalCountField = "LineItemCount";
    GroupingFields = "SourceSystem,POId,POLineNumber";
}
```

This data load task is one of several related tasks that use the`ComputeFractionalCount` class. This class applies to data that has multiple records or line, or multiple lines per item. The default configuration includes instances of this task that are used to group together line items that have split accounting information, such as purchase orders and expense reports. The example above illustrates how to sum the line items of a purchase order.

For accurate data reporting, Ariba Analysis must count the individual parts of a split line items. So, for example, three split line items of a PO are counted as three lines. `ComputeFractionalCount` stores the count in a field of the associated fact.

This task recognizes the following parameters:

- `FactClass`, which specifies the fully-qualified class name of the fact table containing the items to be aggregated.
- `FractionalCountField`, which specifies the name of the target `<field>` where the result is stored. The value must match the `name` attribute of a `<field>` in this fact and the field must be of `<type>` decimal.
- `GroupingFields`, which is is a comma-delimited list of field names. Taken together, those fields must represent 100% of what is being fractionally counted.

DBOLAPProcessing

```
DBOLAPProcessing = {
    ScheduledTaskClassName = "ariba.analytics.tasks.DBOLAPProcessing";
};
```

The `DBOLAPProcessing` task does the following:

- Drops all materialized views.
- Runs validation checks to validate the data in your tables
- Re-creates the materialized views
- Computes statistics where Oracle is used

You must run this task any time you refresh or reload data into Ariba Analysis or when you construct or redefine materialized views.

It is essential that you check the output carefully, and fix any errors reported. If you do not, the materialized views will not be created correctly.

The `DBOLAPProcessing` task calls the tasks `ComputeMatView`, `ComputeDBStats`, and `ValidateDimension`. You can also create task entries to call these tasks individually.

ComputeMatView

```
ComputeMatView = {
   ScheduledTaskClassName = “ariba.analytics.tasks.ComputeMatViews";
   MatView = “POLineItem”;
};
```

The ComputeMatView task deletes and recreates materialized views. It recognizes the following parameters:

- `MatView`, which specifies the name of the materialized view to recreate. The value of the `MatView` parameter must match the value of the `name` attribute of the `<materializedView>` element in the metadata XML file that defines the view.

If the `MatView` parameter is not specified, this task recreates all materialized views.

ComputeDBStats

```
ComputeDBStats = {
   ScheduledTaskClassName = "ariba.analytics.tasks.ComputeDBStats";
   };
```

The ComputeDBStats task computes database statistics for an Oracle database. Depending upon the size of your data, this computation can take a long time.

Running this task after each data load is not strictly necessary, but if you have changed your data model significantly, you should run this task.

ComputeDBStats applies only to Oracle databases. It has no effect on other IBM DB2 or Microsoft SQL. If you are using IBM DB2, you can use the `initdb -db2Stats` command to generate simple database statistics, but in general you should consult with your database administrator for information on how to gather and use database statistics.

ValidateDimension

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

The ValidateDimension task revaluates a dimension. You should run this task after you change the definition of a dimension. It recognizes the following parameters:

- DimensionClass, which specifies the dimension to validate. The value of the DimensionClass parameter must match the value of the name attribute of the <dimension> element in the metadata XML extension file that defines the dimension.

If the DimensionClass parameter is not specified, this task reevaluates all dimensions.

It is essential that you fix any errors reported. If you do not, your materialized views may not be usable. For more information on how to diagnose errors, see "Data Validation" on page 70.

DumpDimension

```
DumpSupplier = {
    ScheduledTaskClassName = "ariba.analytics.tasks.DumpDimension";
    DimensionClass = "ariba.analytics.dimension.Supplier";
    Hierarchy = "SupplierType";
    Filename = "Supplier.csv";
    UnclassifiedOnly = "true";
};
```

The DumpDimension class writes data from a dimension table to a CSV file. It also provides an option that you can use to dump only dimension hierarchies that are unclassified.

This example illustrates how to dump the supplier dimension. It illustrates the following parameters:

- DimensionClass, which specifies the full path of the dimension to dump.
- Hierarchy, which specifies the name of a hierarchy in the specified dimsension. dump. The value must match the value of the name attribute on a <hierarchy> element.
- Filename , which specifies the name of the output file.
- UnclassifiedOnly, which specifies whether you want to dump all data or just unclassified data. The value is a Boolean, false by default. When this parameter is true, only unclassified dimension values for the given hierarchy or level are written to the file.

The DumpDimension class also recognizes the following additional parameters:

- DumpHierarchyFields, which specifies whether you want to dump the additional fields that make up this hierarchy. The default value is true. For a parent-based hierarchy, set this value to false.
- Encoding, which specifies the character encoding of the output file. The default is the value set by config/Parameters.table entry System.Base.DefaultEncoding. You can override the default character encoding by setting the Encoding parameter.
- ExtraFields, which is an optional parameter that specifies a comma-separated list of additional fields to output.
- Level, which specifies the name of a level you want to dump. You can use this parameter to select only values at a specific level. The value must match the value of the name attribute on the <level> element. If both Level and Hierarchy are set, Hierarchy takes precedence over Level.
- ReferenceHierarchy, which specifies the name of a related hierarchy.

If you want a complete dump of all data for a dimension, specify just the parameters DimensionClass and Filename.

DumpFact

```
DumpPOFact = {
   ScheduledTaskClassName = "ariba.analytics.util.DumpFact";
   FactClass = "ariba.analytics.fact.POLineItem";
   Filename = "POLineItem.csv";
   DimensionField = "Commodity";
   };
```

The DumpFact class writes data from a fact table to a CSV file. This example illustrates dumping the purchase order line item fact table.

DumpFact recognizes the following parameters:

- FactClass, which specifies the full path of a fact table name. For example, ariba.analytics.fact.POLineItem
- DimensionField , which specifies the dimension to be written in the output.
- ExtraFields, which is an optional parameter that specifies a comma-separated list of additional fields to output.
- Filename , which specifies the name of the output file.

- Encoding, which specifies the character encoding of the output file. The default is the value set by config/Parameters.table entry System.Base.DefaultEncoding. You can override the default character encoding by setting the Encoding parameter.

DumpReports

```
DumpReports = {
   ScheduledTaskClassName = "ariba.analytics.migration.DumpReport";
   User = "ashell";
   Directory = "dumpReports";
};
```

You can use the DumpReport task to extract report data from the database into CSV files.

The DumpReport task recognizes the following parameters:

- Directory, which specifies the directory from which or to which the definitions are imported or exported. This directory name is relative to *AnalysisServerRoot*. The default is *AnalysisServerRoot*/dumpReports.
- User, which specifies the username of the user whose data is being imported or exported. If you specify no arguments, it dumps all reports.

DumpSystem

```
DumpSystem = {
   ScheduledTaskClassName = "ariba.analytics.tasks.SystemDump";
   Operation = "Export";
   Directory = "systemDump";
};
```

The DumpSystem task writes user data and report definitions to files in the specified directory. The output files are in XML format, using an encoding format based on the Simple Object Access Protocol (SOAP). You typically use this task to capture all user data prior to migrating to a new release.

The DumpSystem class recognizes the following parameters:

- Operation, which is set to Import or Export.
- Directory, which specifies the directory from which or to which the definitions are imported or exported. This directory name is relative to *AnalysisServerRoot*.

DumpUsers

```
DumpUsers = {
   ScheduledTaskClassName = "ariba.analytics.tasks.ObjectEncoding";
   Operation = "Export";
   Directory = "sample/dumpUsers";
   };
```

The DumpUsers task dumps data with users, such as dashboard definitions and folders, from disk files. This task calls the ObjectEncoding class, which recognizes the following parameters:

- `Operation`, which must be `Import` or `Export`. For dumping data, it is `Export`.
- `Directory`, which specifies the directory to which the definitions are exported. This directory name is relative to *`AnalysisServerRoot`*.
- `ObjectType`, which specifies the type of objects being exported. The possible values are `User`, `Folder`, and `FolderItem`. If this parameter is not specified, the default is to export all users and associated folders and folder contents.
- `UserName`, which specifies the username of the user whose data is being imported or exported. This parameter applies only when `ObjectType=User`.
- `FolderPath`, which specifies the full path for folder items. This parameter applies only when `ObjectType=FolderItem`.

GroupSetup Tasks

```
UNSPSCCategoryGroupSetup = {
    ScheduledTaskClassName = "ariba.analytics.tasks.GroupLevelSetup";
    DimensionClass = "ariba.analytics.dimension.UNSPSC";
    GroupLevel = "UNSPSCCategory";
};
```

The GroupLevelSetup class populates the Ariba Analysis database with the data to define group levels. A `groupLevel` is a parent-child relationship among fields in facts and hierarchies. Once you define a `groupLevel` relationship, you must populate the

You must run a GroupLevelSetup task for every `groupLevel` you have defined in your metadata XML. The example above illustrates one specific example of this task.

This task recognizes the following parameters:

- `DimensionClass`, which is the full package name of a dimension
- `GroupLevel`, which is the name of a `groupLevel`

LoadDataSources

```
LoadDataSources = {
    ScheduledTaskClassName = "ariba.analytics.tasks.DataSourceDump";
    Directory = "sample/dumpDataSources";
    Operation = "Import";
}
```

The LoadDataSources task loads data sources from CSV files on disk to Ariba Analysis administration console's **Data Sources** tab.

This task recognizes the following parameters:

- `Operation`, which can be either `Export` or `Import`. For loading data, the operation is `Import`.
- `Directory`, which names the directory where the CSV files are stored.

LoadFromStaging

```
LoadFromStaging = {
    ScheduledTaskClassName = "ariba.analytics.tasks.LoadFromStaging";
}
```

The LoadFromStaging task moves records from staging tables to versioned dimensions. You need to run this task only if you have enabled slowly-changing dimensions. A *slowly-changing dimension* is one whose field values change over time.

During data loading, versioned data is stored in staging tables. At the end of data loading, use the `LoadFromStaging` task to transfer versioned data from staging tables to the appropriate versioned dimensions. You must run this task before you load any data into fact tables.

For more information on slowly-changing dimensions, see the *Ariba Analysis Customization Guide*.

LoadUsers

```
LoadUsers = {
   ScheduledTaskClassName = "ariba.analytics.tasks.ObjectEncoding";
   Operation = "Import";
   Directory = "sample/dumpUsers";
   };
```

The LoadUsers task is an example of how to load data associated with users, such as dashboard definitions and folders, from disk files. You typically use this task after first exporting, with DumpUsers.

The ObjectEncoding task recognizes the following parameters:

- `Operation`, which must be `Import` or `Export`. For loading data, it is `Import`
- `Directory`, which specifies the directory from which the definitions are imported or exported. This directory name is relative to *`AnalysisServerRoot`*.
- `ObjectType`, which specifies the type of objects being imported. The possible values are `User`, `Folder`, and `FolderItem`. If this parameter is not specified, the default is to import or export all users and associated folders and folder contents.
- `UserName`, which specifies the username of the user whose data is being imported or exported. This parameter applies only when `ObjectType=User`.
- `FolderPath`, which specifies the full path for folder items. This parameter applies only when `ObjectType=FolderItem`.

PopulateSupplierStaging

```
PopulateSupplierStaging = {
   ScheduledTaskClassName = "ariba.analytics.util.PopulateStagingDimension";
   DimensionClass = "ariba.analytics.dimension.Supplier";
   }
```

The `PopulateStagingDimension` class converts an existing non-versioned dimension to a versioned one. This example illustrates how to use this task to convert the Supplier dimension.

This task recognizes the following parameters:

- `DimensionClass`, which specifies the full path of the dimension to be converted.

ResetSupplierRank

```
ResetSupplierRank = {
   ScheduledTaskClassName = "ariba.analytics.tasks.ResetDimFields";
   DimensionClass = "ariba.analytics.dimension.Supplier";
   ResetFields = "SupplierIdRank,SupplierNameRank";
   ResetValue = "Low Activity Suppliers ...";
   };
```

The ResetSupplierRank task resets all previously ranked suppliers to “Low Activity Supplier”. You can use this task if you want to recompute how all of your suppliers are ranked.

After running ResetSupplierRank, you should rerun tasks you have defined to rank the suppliers, as described in “SupplierRankByAmount” on page 130.

This task uses the `ResetDimFields` class, which resets dimension fields to a specified value. It recognizes the following parameters:

- `DimensionClass`, which names the dimension whose fields are to be changed.
- `ResetFields`, which specifies the fields of the specified `DimensionClass` that are to be changed
- `ResetValue`, which specifies the new value to be set for those fields

SupplierRankByAmount

```
SupplierRankByAmount = {
   ScheduledTaskClassName = "ariba.analytics.util.SupplierRankSetup";
   FactClass = "ariba.analytics.fact.POLineItem";
   MeasureField = "Amount";
   SupplierField = "Supplier";
   Threshold = "85";
   UpdateMetric = "Percentage";
   SourceFields = "SupplierId,SupplierName";
   DestFields = "SupplierIdRank,SupplierNameRank";
   };
```

The SupplierRankByAmount task creates the supplier hierarchies and ranks suppliers by specified criteria. Suppliers can be ranked either by percentage or by sheer numbers (counts of purchase orders or invoices, for example). Percentages or counts can be based on any defined measure, such as dollar amount, numbers of purchase order or invoice line items.

Suppliers who match the specified criteria are included in the **Most Active Suppliers** hierarchy visible to Ariba Analysis users.

In the example above, supplier ranking is based on currency amount of purchase orders. Suppliers whose purchase orders constitute 85% of the spend are considered to have met the criteria.

This task recognizes the following parameters:

- FactClass specifies the full path of a fact table name. For example, ariba.analytics.fact.POLineItem
- MeasureField specifies the name of the measure on which the ranking is calculated. For example, Amount. The value must be a measure defined for the fact specified by FactClass.
- UpdateMetric specifies whether the ranking is percentage-based or an absolute count. The value is either Numerical or Percentage. The default is Percentage.
- Threshold represents that cutoff point which a supplier must meet in order to be included. The value is an integer. If UpdateMetric is Percentage, Threshold must be less than 100. If Threshold is not specified, the default is 80% (for the Percentage update metric) and 25 (for the Numerical update metric).
- SourceFields are always as shown above.
- DestFields are always as shown above.

SupplierRankByLineCount

```
SupplierRankByLineCount_Commodity = {
   ScheduledTaskClassName = "ariba.analytics.util.SupplierRankSetup";
   FactClass = "ariba.analytics.fact.POLineItem";
   MeasureField = "LineItemCount";
   SupplierField = "Supplier";
   Threshold = "10";
   UpdateMetric = "Numerical";
   SelectFactDimField = "Commodity";
   SelectDimensionField = "CategoryL1";
   SourceFields = "SupplierId,SupplierName";
   DestFields = "SupplierIdRank,SupplierNameRank";
   };
```

The SupplierRankByLineCount task is like SupplierRankByAmount, except that the metric used to rank suppliers is number of purchase order lines, rather than a percentage.

Most parameters are the same as for SupplierRankByAmount. This task also recognizes the following additional parameters:

- SelectFactDimField, which specifies a hierarchy.
- SelectDimensionField, which specifies a level of the specified hierarchy.

Taken together, these parameters create the active supplier ranking **for each component of the `CategoryL1` level** of the `Commodity` hierarchy. You might want to replicate this task for each level of the `Commodity` hierarchy.

Time Setup

```
TimeSetup = {
   ScheduledTaskClassName = "ariba.analytics.util.DayGenerator";
   DimensionClass = "ariba.analytics.dimension.Time";
   DayField = "Day";
   DateFormat = "yyyy-mm-DD";
   FirstDate = "1998-01-01";
   LastDate = "2006-12-31";
   };
```

The TimeSetup task invokes the `DayGenerator` task class, which sets the beginning and end dates of your data. This task improves performance of fact table data loading, and must be run before any fact data are loaded. You can run it manually, with the following command:

`AnalysisServerRoot``/bin/runtask -task TimeSetup`

In general, a fact table always has a time dimension. The `TimeSetup` task creates entries in the time dimension table so that loading for the fact information is faster.

The parameters are as follows:

- `DimensionClass`, which specifies the full path of the dimension maintaining date information.
- `DayField`, which specifies the field of the specified `DimensionClass` that specifies the day. The value is usually as shown here.
- `DateFormat`, which specifies the format for interpreting the dates you specify as `FirstDate` and `LastDate`. If not set, the date format defaults to the value specified as `System.Analysis.DataLoading.DateFormat`. For information on valid date format specifications, see “Date Formats” on page 109.
- `FirstDate`, which specifies the first date of interest. Set this parameter to a vlaue that is equal to the starting date of your data.
- `LastDate`, which specifies the last date of interest. Set this parameter to a value at least a year after the date you installed Ariba Analysis, and preferably at least a year after the date of your latest data. If you do not set this value far enough in the future, you will have to reset the parameters and re-run the task later.

Index

D

E

F

G

I

J

L

M

N

O

P

Q

R

S

T

U

V

ARIBA, INC.

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

www.ariba.com