Oracle Commerce Guided Search Platform Services

Forge Guide

Version 11.1 • July 2014

ORACLE

COMMERCE

Contents

Preface....9
About this guide....9
Who should use this guide....9
Conventions used in this guide....9
Contacting Oracle Support....10

Part I: Basic Pipeline Development....11

Chapter 1: The Endeca ITL....13
Introduction to the Endeca ITL....13
Endeca ITL components....14

Chapter 2: Endeca ITL Development....17
Endeca ITL development process....17
Endeca tools suite....17
A closer look at data processing and indexing....21

Chapter 3: Overview of Source Property Mapping....27
About source property mapping....27
About using a single property mapper....27
About using explicit mapping....27
Minimum configuration....28
About mapping unwanted properties....28
About removing source properties after mapping....28
Types of source property mapping....29
About adding a property mapper....30
The Mappings editor....32

Chapter 4: Match Modes....35
About choosing a match mode for dimensions....35
Rules of thumb for dimension mapping....37
Dimension mapping example....37

Chapter 5: Advanced Mapping Techniques....39
The Property Mapper editor Advanced tab....39
About enabling implicit mapping....39
Enabling default mapping....40
About the default maximum length for source property values....41

Chapter 6: Before Building Your Instance Configuration....43
Endeca Application Controller directory structure....43
Pipeline overview....43

Chapter 7: About Creating a Basic Pipeline....47
The Basic Pipeline template....47
Record adapters....48
Dimension adapter....49
Dimension server....49
Property mapper....50
Indexer adapter....51

Chapter 8: About Running Your Basic Pipeline....53
Running a pipeline....53

Viewing pipeline results in a UI reference implementation....53

Chapter 9: After Your Basic Pipeline Is Running....55
Additional tasks....55
About source property mapping....55
Setting the record specifier property....58
About specifying dimensions and dimension value order....59
Additional pipeline components....59
Additional index configuration options....60

Part II: Joins....63

Chapter 10: Overview of Joins....65
Record assemblers and joins....65
About performing joins in a database....66
Join keys and record indexes....66
Join types....68

Chapter 11: About Configuring Join Keys and Record Indexes....75
Creating a record index....75
Creating a join key for a record cache....76
Join keys with multiple properties or dimensions....77

Chapter 12: About Implementing Joins....79
Implementing a join....79

Chapter 13: Advanced Join Behavior....83
Records that have multiple values for a join key....83
Sources that have multiple records with the same join key value....84
About tweaking left joins....85

Chapter 14: Tips and Troubleshooting for Joins....87
Joins that do not require record caches....87
Working with sources that have multiple records with the same join key value....87
Best practice for choosing left and right side of joins....87
Combining equivalent records in record caches....88
Forge warnings when combining large numbers of records....89

Part III: Advanced Dimension Features....91

Chapter 15: Externally-Created Dimensions....93
Overview of externally-created dimensions....93
XML requirements....95
Importing an externally-created dimension....97

Chapter 16: Externally-Managed Taxonomies....99
Overview of externally-managed taxonomies....99
Including externally-managed taxonomies in your project....99
XSLT and XML requirements....100
Pipeline configuration....102
About updating an externally-managed taxonomy in your pipeline....105
Unexpected default-mapping behavior....105

Part IV: Other Advanced Features....107

Chapter 17: The Forge Logging System....109

Overview of the Forge logging system....109
Log levels reference....109
About logging topics....109
The command line interface....110

Chapter 18: The Forge Metrics Web Service....113
About the Forge Metrics Web service....113
About enabling Forge metrics....114
About using Forge metrics....114
The MetricsService API....115

Appendix A: Forge Flag Reference....117
Forge flag options reference....117

Appendix B: File Formats Supported by the Document Conversion Module.123
Word processing formats....123
Text and markup formats....125
Spreadsheet formats....126
Vector image formats....127
Raster image formats....128
Presentation formats....130
Archive formats....130
Database formats....131
E-mail formats....131
Other formats....132

Appendix C: Advanced JDBC Column Handler....133
About the Advanced JDBC Column Handler....133
JDBC configuration options....133
Storing data on disk....135
Using the Advanced JDBC Column Handler....135
Output....137
Troubleshooting....139

Copyright and disclaimer

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Preface

Oracle Commerce Guided Search is the most effective way for your customers to dynamically explore your storefront and find relevant and desired items quickly. An industry-leading faceted search and Guided Navigation solution, Guided Search enables businesses to influence customers in each step of their search experience. At the core of Guided Search is the MDEX Engine™, a hybrid search-analytical database specifically designed for high-performance exploration and discovery. The Oracle Commerce Content Acquisition System provides a set of extensible mechanisms to bring both structured data and unstructured content into the MDEX Engine from a variety of source systems. The Oracle Commerce Assembler dynamically assembles content from any resource and seamlessly combines it into results that can be rendered for display.

Oracle Commerce Experience Manager enables non-technical users to create, manage, and deliver targeted, relevant content to customers. With Experience Manager, you can combine unlimited variations of virtual product and customer data into personalized assortments of relevant products, promotions, and other content and display it to buyers in response to any search or facet refinement. Out-of-the-box templates and experience cartridges are provided for the most common use cases; technical teams can also use a software developer's kit to create custom cartridges.

About this guide

This guide describes the major tasks involved in developing the instance configuration, including the pipeline, of an Endeca application.

It assumes that you have read the *Oracle Commerce Getting Started Guide* and are familiar with the Endeca terminology and basic concepts.

Who should use this guide

This guide is intended for developers who are building applications using Oracle Commerce.

Conventions used in this guide

This guide uses the following typographical conventions:

Code examples, inline references to code elements, file names, and user input are set in `monospace` font. In the case of long lines of code, or when inline monospace text occurs at the end of a line, the following symbol is used to show that the content continues on to the next line: ¬

When copying and pasting such examples, ensure that any occurrences of the symbol and the corresponding line break are deleted and any remaining space is closed up.

Contacting Oracle Support

Oracle Support provides registered users with answers to implementation questions, product and solution help, and important news and updates about Guided Search software.

You can contact Oracle Support through the My Oracle Support site at *https://support.oracle.com*.

Part 1

Basic Pipeline Development

- *The Endeca ITL*
- *Endeca ITL Development*
- *Overview of Source Property Mapping*
- *Match Modes*
- *Advanced Mapping Techniques*
- *Before Building Your Instance Configuration*
- *About Creating a Basic Pipeline*
- *About Running Your Basic Pipeline*
- *After Your Basic Pipeline Is Running*

Chapter 1

The Endeca ITL

The Endeca Information Transformation Layer (ITL) is a major component of Oracle Commerce. This section provides an introduction to the Endeca ITL and its componenets.

Introduction to the Endeca ITL

The Endeca Information Transformation Layer (ITL) reads in your source data and manipulates it into a set of indices for the Endeca MDEX Engine. The Endeca ITL consists of the Content Acquisition System and the Data Foundry.

Although the original source data is not changed, this transformation process may change its representation within your Endeca implementation. The Endeca ITL is an off-line process that you run on your data at intervals that are appropriate for your business requirements.

Endeca Content Acquisition System

The Content Acquisition System includes the Endeca Web Crawler and the Endeca CAS Server, as well as a rich set of packaged adapters.

These components crawl unstructured content sources and ingest structured data. This includes relational databases, file servers, content management systems, and enterprise systems such as enterprise resource planning (ERP) and master data management (MDM).

Packaged adapters reach the most common systems, including JDBC and ODBC. The Content Adapter Development Kit (CADK) allows developers to write custom adapters and Java manipulators.

Endeca Data Foundry

The Endeca Data Foundry aggregates information and transforms it into Endeca records and MDEX Engine indices.

During the data processing phase, the Data Foundry:

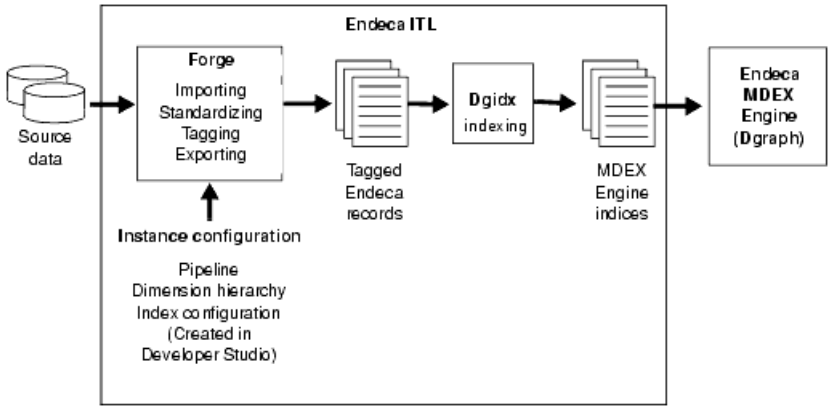
- Imports your source data
- Tags it with the dimension values used for navigating and Endeca properties used for display.
- Stores the tagged data—along with your dimension specifications and any configuration rules—as Endeca records that are ready for indexing.

- Indexes the Endeca records it produced during its data processing phase, and produces a set of indices in Endeca MDEX Engine format.

Endeca ITL components

At a base level, the Endeca ITL is a combination of programs and configuration files. The Endeca ITL has additional components that support a variety of features.

This illustration shows a high-level view of the Endeca ITL architecture.



The components described in this section are the core components that all Endeca implementations use, regardless of the additional features they implement.

Pipeline components will be discussed in this guide as is appropriate. For more detailed information about pipeline components, see the Developer Studio online help.

Data Foundry programs

Data Foundry component is composed of two core programs, Forge and Dgidx.

- **Forge** is the data processing program that transforms your source data into standardized, tagged Endeca records.
- **Dgidx** is the indexing program that reads the tagged Endeca records that were prepared by Forge and creates the proprietary indices for the Endeca MDEX Engine.

Configuration files

Forge and Dgidx use an *instance configuration* to accomplish their tasks. An instance configuration includes a pipeline, a dimension hierarchy, and an index configuration.

Pipeline

The *pipeline* functions as a script for the entire process of transforming source data to Endeca records.

The pipeline describes a data processing workflow as a graph of data transformation stages, known as components, connected by links across which data flows.

The components specify the format and the location of the source data, any changes to be made to the source data (manipulation), and how to map each record's source properties to Endeca properties and dimensions.

If you intend to run partial updates, your instance configuration will contain two pipelines: one for running baseline updates and one for partial updates. See the *Endeca Partial Updates Guide* for details on setting up the partial updates pipeline.

Dimension hierarchy

The *dimension hierarchy* contains a unique name and ID for each dimension, as well as names and IDs for any dimension values created in Developer Studio. The Data Foundry uses these unique names and IDs when it maps your data's source properties to dimensions.

These names and IDs can be created in three different ways:

- Automatically, by the Data Foundry.
- In Developer Studio.
- In an external system, and then imported either into the Data Foundry or Developer Studio.

The dimension hierarchy is used during indexing to support the incremental filtering that is the essence of Guided Navigation.

Index configuration

The *index configuration* defines how your Endeca records, Endeca properties, dimensions, and dimension values are indexed by the Data Foundry. The index configuration is the mechanism for implementing a number of Endeca features such as search and ranking.

Chapter 2

Endeca ITL Development

The Endeca Information Transformation Layer components enable you to develop your data processing back end. This section provides an overview of the development process and Endeca tools suite, and a closer look at data processing and indexing.

Endeca ITL development process

The Endeca ITL uses an instance configuration to process, tag, and locate data.

Creating an instance configuration is an iterative process. Oracle recommends that you first create a very simple instance configuration to test your data. After the simple configuration is working as you expect, you can make additional modifications, view your results, and make changes as necessary. Also, it is often useful to work on a subset of your data, for quicker turnaround of data processing, while you are developing your instance configuration.

Endeca ITL development consists of the following steps:

1. Use Developer Studio to create an instance configuration.

   This defines how your data should be indexed and displayed. It includes Content Acquisition System components, such as a JDBC Adapter.
2. Use an Endeca Deployment Template application to do the following:
   a) Run Forge, referencing the instance configuration, to process your source data into tagged Endeca records.
   b) Run Dgidx on the Forge output to create MDEX Engine indices from the tagged Endeca records.
   c) Run Dgraph to start a MDEX Engine and point it at the indices created by Dgidx.
3. View the results and repeat these steps to make changes as necessary.

Endeca tools suite

The Endeca distribution includes two tools that help you create and edit your instance configuration, and maintain your Endeca implementation: Endeca Developer Studio and Oracle Endeca Workbench. This section provides a brief introduction to these tools.

Endeca Developer Studio

Endeca Developer Studio is a Windows application that you use to define all aspects of your instance configuration.

With Developer Studio, you can define:

- Pipeline components for tasks such as loading, standardizing, joining, mapping, and exporting data.
- Endeca properties and property attributes such as sort and rollup.
- Dimensions and dimension values, including dimension hierarchy.
- Precedence rules among dimensions that provide better control over your implementation's navigation flow.
- Search configurations, including which properties and dimensions are available for search.
- Dynamic business rules that allow you to promote certain records on your Web site using data-driven business logic. Dynamic business rules are used to implement merchandising and content spotlighting.
- User profiles that tailor the content returned to an end-user based upon preconfigured rules.

Developer Studio uses a project file, with an `.esp` extension, that contains pointers to the XML files that support an instance configuration. Editing a project in Developer Studio edits these underlying files.

Oracle Endeca Workbench

Oracle Endeca Workbench is a Web-based application that provides access to reports that describe how end-users are using an Endeca implementation.

The two primary audiences for Endeca Workbench are:

- Business users who define business logic such as merchandising/content-spotlighting rules and thesaurus entries.

  Endeca Workbench lets business users make changes to parts of an Endeca implementation after the implementation's core functionality has been developed. For example, a developer uses Developer Studio to specify which Endeca properties and dimensions are available for search, then a business user uses Endeca Workbench to specify thesaurus entries that support search functionality.

- System administrators who maintain and manage an Endeca implementation.

  Endeca Workbench lets system administrators provision applications, components and scripts to the Endeca Application Controller, monitor the status of an Endeca implementation, and start and stop system processes.

Endeca Workbench can report the most popular search terms, the most popular navigation locations, search terms that are most often misspelled, and so forth.

About system provisioning tasks in Endeca Workbench

System provisioning lets you assign resources to a new Endeca application in Endeca Workbench, and modify the resources in an existing application. You can provision more than one application to the EAC, using the EAC Admin Console page of Endeca Workbench.

Typically, you provision resources to the Endeca configuration in the following order:

1. Add, edit or remove an Endeca application.
2. Add, edit or remove hosts from the application.
3. Add, configure or remove Endeca components on one or more hosts.

   Endeca components include Forge, the Indexer (Dgidx), Aggregated Indexer, MDEX Engine (Dgraph), Log Server, and Report Generator.

4. Add, edit, or remove an EAC script.

About system operations tasks in Endeca Workbench

System operations let you run Endeca components by using Endeca Workbench to call underlying EAC processes.

On the EAC Admin Console page of Endeca Workbench, you can do the following:

- Start and stop the Endeca applications and components you provision.

  Typically, each provisioned application can have its own set of components, such as Forge, the Indexer, the MDEX Engine, the Log Server and the Report Generator. You can then start and stop these components.

- Start and stop the EAC scripts you provision. These could include the scripts that perform a baseline update and report generation for the application.
- Monitor the status of Endeca components.

Finding more information on tools setup and usage

You can find tool setup and usage information in the following locations:

- The *Oracle Endeca Workbench Administrator's Guide* provides in-depth information about tool setup and configuration.
- The *Oracle Endeca Developer Studio Help* and the *Oracle Endeca Workbench Help* provide details on using each individual tool's features.

About controlling your environment

While not part of the Endeca ITL development per se, before you can begin building and running pipelines, you must put into place a mechanism for controlling the resources in your Endeca implementation. This mechanism provides process execution and job management facilities.

About using the Endeca Application Controller

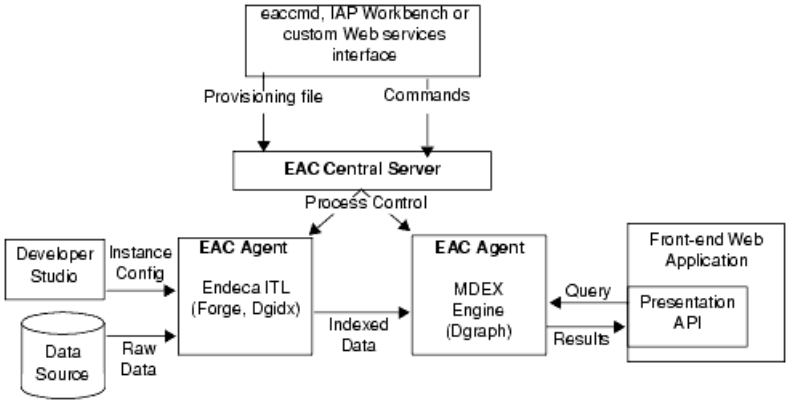
The Endeca Application Controller is the interface you use to control, manage, and monitor your Endeca implementations.

The use of open standards, such as the Web Services Descriptive Language (WSDL), makes the Application Controller platform and language agnostic. As a result, the Application Controller supports a wide variety of applications in production. In addition, the Application Controller allows you to handle complex operating environments that support features such as partial updates, delta updates, phased Dgraph updates and more.

Application Controller architecture

Most implementations that use the Application Controller will follow the general setup outlined below.

The following illustration shows the architecture of a typical implementation that uses the Application Controller.



In this architecture diagram, the following happens:

1. The developer creates an instance configuration, using Developer Studio, that determines what data and features will be incorporated into the index.
2. The developer creates a provisioning document in XML format that defines all the hosts and components in the implementation.
3. The developer sends the provisioning files to the EAC Central Server machine. The developer can use three methods for the provisioning tasks:
   - Endeca Workbench
   - The eaccmd utility
   - A custom Web services interface.
4. Once the Agent machines in the implementation are provisioned, the developer sends commands (again using either eaccmd, Endeca Workbench, or a custom interface) to the EAC Central Server. The EAC Central Server communicates these tasks to its Agents, which reside on each machine that is running Endeca components.
5. The Application Controller manages the entire data update process, according to the instructions it receives. This includes running Forge and the Indexer (Dgidx) to create indexed data, and starting the MDEX Engine (Dgraph) based on that indexed data.

For detailed information on configuring and using the Endeca Application Controller, see the *Oracle Endeca Application Controller Guide*.

Ways of communicating with the Endeca Application Controller

You have three ways in which you can communicate with the EAC Central Server:

- Endeca Workbench
- The eaccmd utility
- A custom Web services interface (using the Endeca WSDL).

About using Endeca Workbench to communicate with the EAC Central Server

Endeca Workbench lets you provision the resources in your environment, such as applications, components and logging, and start and stop these resources as needed. Endeca Workbench communicates this information to the EAC Central Server to coordinate and execute the processes that result in a running Endeca implementation.

Endeca Workbench is one of the ways of communicating with the EAC Central Server (the other two are the eaccmd utility and a custom Web services interface).

The primary benefit of using Endeca Workbench as a means of communication with the EAC Central Server is that it relieves you of the burden of using the command line utility eaccmd, or of creating a custom Web services interface.

Endeca Workbench allows multiple users to edit the same implementation while avoiding conflicting changes. Only one Endeca Workbench user can edit a particular implementation module at any given time, locking out all other users from that module.

Important: Concurrent project editing can only happen in Endeca Workbench. There is no built-in allowance for concurrent users of Endeca Workbench and Developer Studio. Therefore, to prevent changes from being overwritten or otherwise lost, a project should be active in only one of these tools at a time.

A closer look at data processing and indexing

It is important to have a clear understanding of how the Data Foundry works with source records before you begin building your instance configuration. Read the following sections for a behind-the-scenes look at the data processing and indexing functions in the Data Foundry.

Data processing

The data processing workflow in the Data Foundry is defined in your pipeline and typically follows a specific path.

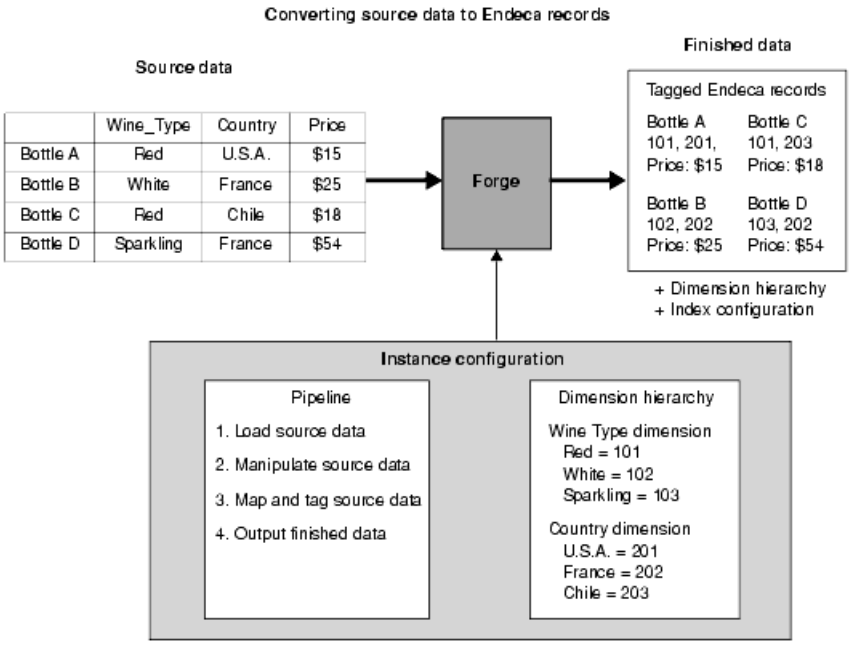
The Forge and Dgidx programs do the actual data processing, but the components you have defined in the pipeline dictate which tasks are performed and when. The Data Foundry attempts to utilize all of the hardware resources available to it, both by processing records in multiple components simultaneously, and by processing multiple records simultaneously within the same component.

The data processing workflow typically follows this path:

1. Load the raw data for each source record.
2. Standardize each source record's properties and property values to create consistency across records.
3. Map the source record's properties into Endeca properties and/or dimensions.
4. Write the tagged Endeca records, along with any dimension hierarchy and index configuration, as finished data that is ready for indexing.
5. Index the finished data and create the proprietary indices used by the MDEX Engine.

Data processing workflow

The following illustration shows a simple conversion of source data into tagged Endeca records:



Source data

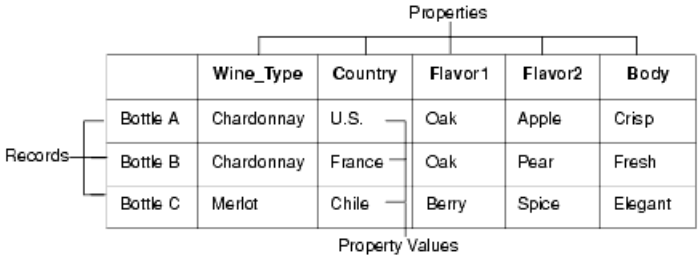
You can load source data from a variety of formats using the Content Acquisition System components.

Your Endeca applications will most often read data directly from one or more database systems, or from database extracts. Input components load records in a variety of formats including delimited, JDBC, and XML. Each input component has its own set of configuration properties. One of the most commonly used type of input component loads data stored in delimited format.

About loading source data

Source data may be loaded into the Data Foundry from a variety of formats. The easiest format to use is a two-dimensional format similar to the tables found in database management systems.

Database tables are organized into rows of records, with columns that represent the source properties and property values for each record. The illustration below shows a simple example of source data in a two-dimensional format.



You specify the location and format of the source data to be loaded in the pipeline. Forge loads and processes one source record at a time, in sequential order. When Forge loads a source record, it transforms the record into a series of property/property value pairs.

Source Data

	Wine_Type	Country	Flavor1	Flavor2	Body
Bottle A	Chardonany	U.S.	Oak	Apple	Crisp

Loaded Data

Bottle A
Wine_Type: Chardonany
Country: U.S.
Flavor1: Oak
Flavor2: Apple
Body: Crisp

Standardizing source records

You specify any standardization of source properties and property values in the pipeline. Standardization cleanses the data so that it is as consistent as possible before mapping begins.

You can take the following steps to standardize your data:

Note: The functionality described below supports limited data cleansing. If you have an existing data cleansing infrastructure, it may be more advantageous to use that facility instead.

1. Fix misspellings in your source properties and property values.
2. Explicitly specify the encoding type (e.g., UTF-8, CP-1252, or Latin-1) of the source data when Forge reads it into a Pipeline. If you are loading text-based source data in a Record Adapter, you specify the encoding type in the Encoding field of the General tab. If an incorrect encoding is specified, then Forge generates warnings about any characters that do not make sense in the specified encoding. For example, in the ASCII encoding, any character with a number above 127 is considered invalid. Invalid characters are replaced with strings prefixed by %X, so the invalid characters are not loaded into Forge.
3. Remove unsupported characters.

   The only legal Unicode characters are U+09, U+0D, U+0A, U+20-U+7E, U+85, U+A0-U+D7FF, and U+E000-U+FFFD. In particular, source data should not contain Unicode characters from the range 0x00 through 0x1F with the exceptions of 0x09 (tab), 0x0A (newline), and 0x0D (carriage return). For example, records based on databases may use 0x00 (null) as a default empty value. Other characters that are often in existing database sources are 0x1C (field separator), 0x1E (record separator), and 0x1F (unit separator).

   If a data source contains additional control characters as defined by the chosen encoding, remove or replace the control characters. For example, Windows-1252 specifies 0x7F-0x81, 0x8D-0x90, 0x9D-0x9E as control characters, and Latin-1 specifies x7F and x9F as control characters.

   The following are some notes and suggestions for dealing with control characters:

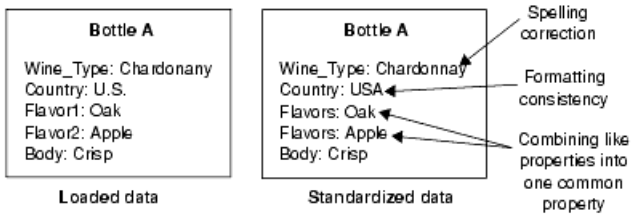
   - The default input adapter encoding (LATIN-1) for delimited and vertical record input adapters in Forge makes the assumption, for throughput efficiency, that input data does not contain control characters (i.e. x00-x1F [except x09, x0A, x0D] and x7F-x9F).
   - For data sources that contain control characters because of character data in a non-Latin encoding (e.g., UTF-8 or Windows-1252), the recommended and best practice solution is to explicitly specify the encoding type (e.g., "UTF-8" or "Windows-1252").
   - For data sources that contain character data in more than one non-Latin encoding (e.g., a mixture of UTF-8 and Windows-1252), the recommended and best practice solution is to explicitly specify the more conservative encoding type (e.g., UTF-8).
   - For data sources where the data-cleanliness assumption is not satisfied because of real control characters (i.e., x00-x1F [except x09, x0A, x0D] and x7F), the recommended and best practice solution is to clean the data ahead of time to remove or replace those control characters. If data sources contain additional

control characters as defined by the chosen encoding, these should also be removed or replaced.For data sources where the data-cleanliness assumption is not satisfied because of real control characters (i.e., x00-x1F [except x09, x0A, x0D] and x7F), the recommended and best practice solution is to clean the data ahead of time to remove or replace those control characters. If data sources contain additional control characters as defined by the chosen encoding, these should also be removed or replaced.

4. Edit source property values to use a consistent format (for example, USA instead of United States or U.S.).
5. Re-assign similar source properties to one common property. (for example, you could assign a Flavor1 property and a Flavor2 property to a generic Flavors property).

Example of standardized source records

The following image shows a simple standardization example:



About mapping source properties and property values

After a source record has been standardized, Forge maps the record's source properties to dimensions and Endeca properties.

- Mapping a source property to a dimension indicates that the record should be tagged with a dimension value ID from within that dimension. This enables navigation on the property.
- Mapping a source property to an Endeca property indicates that the property should be retained for display and search.

Related Links

Overview of Source Property Mapping on page 27

The property mapper is a pipeline component used to map properties on the records in your source data to Endeca properties and/or dimensions to make them navigable, displayable, both, or neither. The property mapper is a key component in developing a pipeline, so it is important to understand its functions well.

About writing out tagged data

After all the source records have been mapped, the Forge program writes its finished data.

The finished data consists of:

- The Endeca records along with their tagged dimension value IDs and Endeca properties.
- The names and IDs for each dimension and dimension value, along with any dimension hierarchy.
- Any index configuration specified.

About indexing

After Forge creates the tagged data, Dgidx indexes the output and creates the proprietary indices for the Endeca MDEX Engine.

Chapter 3

Overview of Source Property Mapping

The property mapper is a pipeline component used to map properties on the records in your source data to Endeca properties and/or dimensions to make them navigable, displayable, both, or neither. The property mapper is a key component in developing a pipeline, so it is important to understand its functions well.

About source property mapping

Source property mappings dictate which dimension values are tagged to each record and which property information is available for record search, sort, and display.

Note that before you can map a source property to an Endeca property or dimension, you must have created that Endeca property or dimension.

Source properties can be mapped in three different ways. They can be:

- Mapped to an Endeca property (for search, sort, and display only).
- Mapped to a dimension (for search, sort, display, and navigation).
- Ignored by specifying a null mapping.

You use a property mapper component to establish source property mappings. Typically, the property mapper is placed in the pipeline after the Perl manipulator (if one exists) that is used to clean and prepare source properties. You should use a single property mapper to map all of your source properties to both Endeca properties or dimensions.

About using a single property mapper

You should use a single property mapper to map *all* of your source properties to *both* Endeca properties or dimensions. Although there are rare cases where multiple property mappers may be used, Oracle strongly recommends that you use only one property mapper in any given pipeline.

About using explicit mapping

When you specify a source property and a target Endeca property or dimension to map to, you are creating an explicit mapping. In general, explicit mapping is the type of mapping Oracle recommends you use.

However, Developer Studio also offers some advanced techniques that allow you to automate the mapping process. These techniques are intended to facilitate the process of building prototypes and should not be used for building production-ready implementations.

Related Links

Advanced Mapping Techniques on page 39
You can specify mapping techniques and default behavior using the **Property Mapper editor Advanced** tab.

Types of source property mapping on page 29
There are four types of source property mappings:

Minimum configuration

At a minimum, a property mapper requires both a record source and a dimension source to define the components that will supply it with record and dimension data.

The dimension source must be a dimension server. You can leave the other settings at their defaults while developing your initial working pipeline, then add mappings as needed.

About mapping unwanted properties

Mapping properties that do not add value to the application is wasteful in terms of processing time and resources. Oracle recommends, therefore, that you only create mappings for those source properties you intend to use in your final application.

Source properties that do not have mappings specified for them are ignored during the mapping process, unless you use the advanced mapping techniques on the **Property Mapper** editor **Advanced** tab.

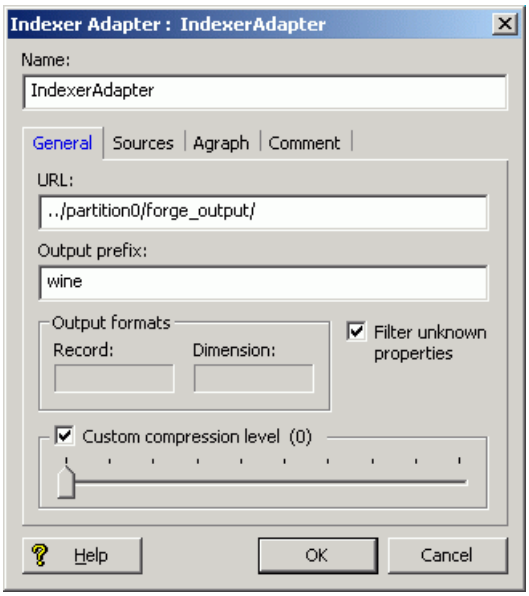
Related Links

Advanced Mapping Techniques on page 39
You can specify mapping techniques and default behavior using the **Property Mapper editor Advanced** tab.

About removing source properties after mapping

After mapping, source properties still exist as part of the Endeca record. You can remove them and create a record that consists exclusively of Endeca properties and dimension values by enabling the **Filter Unknown Properties** setting in your pipeline's **indexer adapter**.

The following example shows this option:



Types of source property mapping

There are four types of source property mappings:

- **Explicit mapping** — Explicit mappings are created when you use the property mapper's **Mappings editor** to specify a source property and a target Endeca property or dimension to map to. In other words, the mapping does not exist until you explicitly create it. In general, this is the type of mapping Oracle recommends that you use.
- **Null mapping** — Null mappings are a type of explicit mapping, because you have to use the **Mappings editor** to explicitly create one. The difference is that while explicit mappings map a source property to an Endeca property or dimension, a null mapping tells the Data Foundry that it should not try to map a specific source property.

  Explicit null mappings provide a means to prevent an implicit or default mapping from being formed for a particular source property. In other words, you can enable either implicit or default mapping, and then turn off mapping altogether for selected source properties using explicit null mappings.

- **Implicit mapping** — When implicit mapping is enabled, any source property that has a name that is identical to an existing dimension is automatically mapped to that dimension. The like-named dimension, and any of its constituent dimension values, must already exist in your dimension hierarchy.

  Note: Implicit mapping works only if no explicit mapping exists.

  Implicit mapping is limited to mappings between source properties and dimensions. Implicit mapping cannot take place between source properties and Endeca properties.

  You enable implicit mapping from the **property mapper Advanced** tab.

- **Default mapping** — This option defines the default that Forge uses to handle source properties that have neither explicit nor implicit mappings. You can specify that Forge ignore source properties without explicit or implicit mappings, create a new Endeca property to map to the source property, or create a new dimension to map to the source property.

  You enable default mapping from the **property mapper Advanced** tab.

Important: Techniques to automate the mapping process are intended to facilitate the process of building prototypes and should not be used for building production-ready implementations. Implicit and default mapping techniques can have unexpected results if you're not careful when using them.

Related Links

About enabling implicit mapping on page 39
The first advanced option, **Map source properties to Endeca dimensions with the same name**, enables implicit mapping.

Enabling default mapping on page 40
The default mapping option defines the default that Forge uses to handle source properties that have neither explicit nor implicit mappings. There are three possible settings.

Priority order of source property mapping

Forge uses a specific prioritization when mapping source properties.

1. Forge looks for an explicit mapping for the source property.
2. If no explicit mapping exists and **"Map source properties to Endeca dimensions with the same name"** is enabled, Forge tries to create an implicit mapping between the source property and a like-named dimension.
3. If no explicit or implicit mapping exists, Forge uses the **"If no mapping is found, map source properties to Endeca: Properties/Dimensions"** option to determine how to handle the mapping.

About adding a property mapper

This section provides a quick overview to adding a property mapper to the pipeline, including:

- Determining where to place the property mapper in the pipeline.
- Creating the property mapper in Developer Studio.
- Using the **Mappings editor**, which you use to create explicit and null mappings.

Determining where to add the property mapper

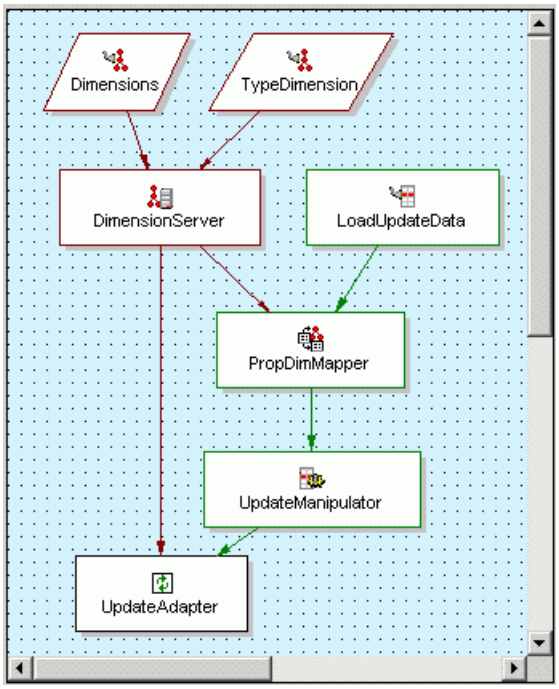
The fundamental requirements for the placement of a property mapper in the pipeline are:

- The property mapper must come *after* a record input component (such as a record adapter) and a dimension input component (such as a dimension server).
- The property mapper must come *before* the indexer adapter.

In a basic pipeline, the property mapper uses the record adapter as its record source and the dimension server as its dimension source, and then the indexer adapter takes the property mapper's output as its record source.

Partial Update Pipeline

Pipelines used for partial updates also use a property mapper, as explained in the *Endeca Partial Updates Guide*. The Pipeline Diagram example below shows a partial update pipeline:



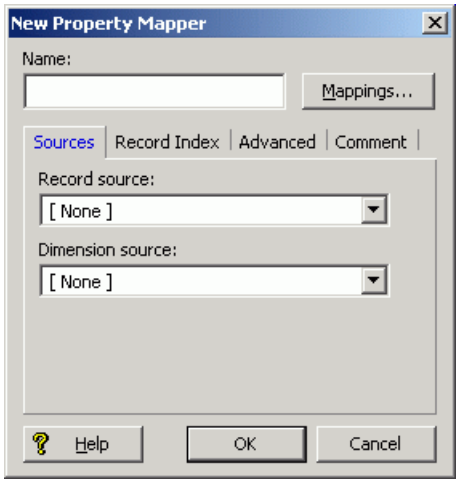
In this partial update pipeline, the property mapper (PropDimMapper) uses the record adapter (LoadUpdateData) as its record source and the dimension server as its dimension source. The record manipulator (UpdateManipulator) uses the property mapper as its record source.

Creating the property mapper

The Developer Studio help provides a step-by-step procedure of how to add a property mapper to your pipeline. This section gives an overview of the general steps.

To create a property mapper:

1. In Developer Studio, open the **Pipeline Diagram** dialog.
2. Select **New** > **Property Mapper**.
   A **New Property Mapper editor** is displayed.



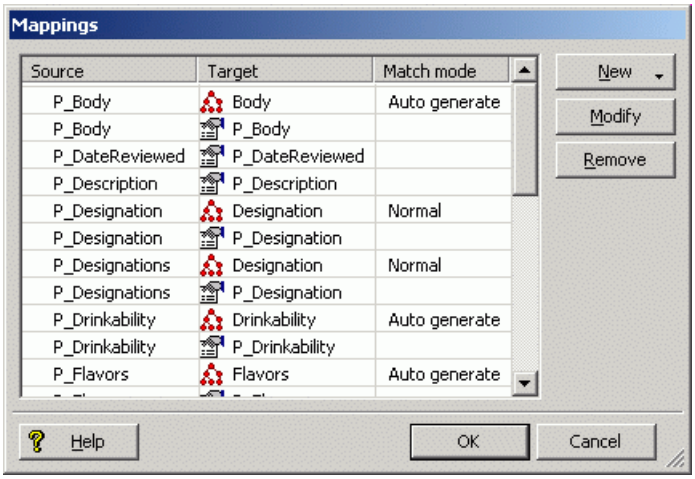
3. Enter a name for the property mapper, a record source, and a dimension source. You can leave the other settings at their defaults while developing your initial working pipeline.
4. To add the property mapper, click **OK**.

The next sections will give overviews of the functions available in the **Mappings editor**.

The Mappings editor

The **Mappings editor** is where you create your source property mappings. You access this editor from the **Property Mapper editor** by clicking the **Mappings** button.

When you open the **Mappings** editor, it displays a table of the existing source property mappings:



The meanings of the table columns are:

- **Source** – The name of the source property to be mapped.
- **Target** – The name of an Endeca property or dimension to which the source property will be mapped. This cell will be empty if the source property has a null mapping.
- **Match mode** – Indicates the type of match mode used for a dimension mapping (the cell will be empty for properties).

Related Links

About choosing a match mode for dimensions on page 35

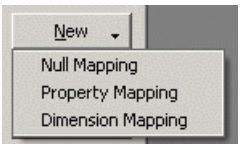
In Developer Studio, you set the type of dimension value handling, on a per mapping basis, by selecting a mode from the **Match mode** list in the **Dimension Mapping** editor, as illustrated below:

Creating new source mappings

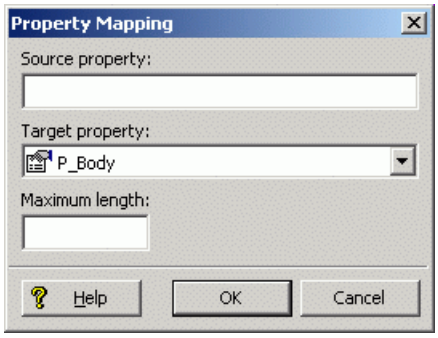
The **New** button lets you create a new source property mapping.

To create a new mapping:

1. Left-click the **New** button.
   Three choices are displayed.



2. Select the type of mapping you wish to create.
   The corresponding editor appears. For example, selecting **Property Mapping** displays the **Property Mapping** editor.



3. Enter the name of the source property and select a target Endeca property or dimension to which the source property will be mapped.

   The **Maximum Length** field defines the maximum source property value length allowed when creating mappings. That is, source properties that have values that exceed this length are not mapped.

The *Oracle Endeca Developer Studio help* also provides information on the **Property Mapping** editor and the **Dimension Mapping** editor.

Using null mappings to override implicit and default mappings

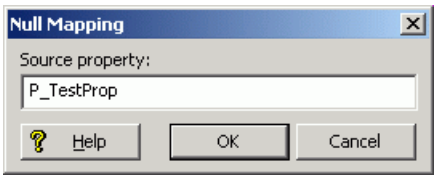
Explicit null mappings provide a means to prevent an implicit or default mapping from being formed for a particular source property. In other words, you can enable either implicit or default mapping, and then turn off mapping altogether for selected source properties using explicit null mappings.

To create a null mapping:

1. Select **New > Null Mapping** in the **Mappings** editor.
2. Enter the source property name in the **Null Mapping** editor.

Example

The following example shows a source property named P_TestProp that will have a null mapping:



About assigning multiple mappings

You can assign more than one mapping to a source property—for example, you can map a source property to both a dimension and an Endeca property. A typical source property that you may want to map to both a dimension and an Endeca property is Price.

You can map the Price source property in the following ways:

- To a Price Range dimension that allows the end-user to search for records within a given price range (for example, wines that cost between $10 and $25).
- To an Endeca property that allows you to display the discrete price of each individual record.

Conversely, you can assign more than one source property to a single dimension or Endeca property. For example, if you have multiple source properties that are equivalent, most likely they should all be mapped to the same dimension or Endeca property. Flavor and Color are example properties that might require this behavior.

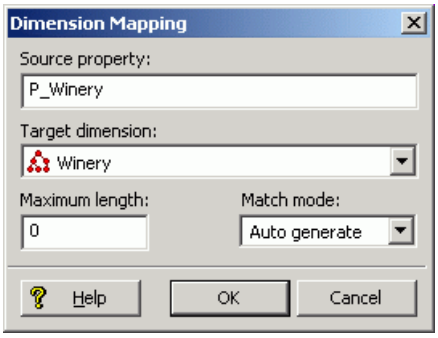
Chapter 4

Match Modes

When Forge maps a source property value to a dimension value, the dimension value it uses can either be explicitly defined in the dimension hierarchy or automatically generated by Forge. You control this behavior by using match modes.

About choosing a match mode for dimensions

In Developer Studio, you set the type of dimension value handling, on a per mapping basis, by selecting a mode from the **Match mode** list in the **Dimension Mapping** editor, as illustrated below:



There are three match modes you can choose from:

- **Normal**
- **Must Match**
- **Auto Generate**

Note: Match modes only apply to dimensions. They are not used when mapping source properties to Endeca properties.

Normal mode

Normal match mode maps only those source property values that have a matching dimension value explicitly defined in the dimension hierarchy.

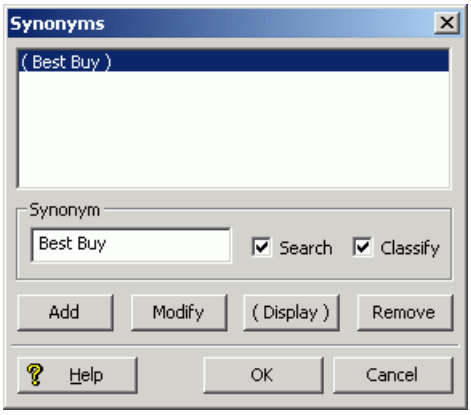
Forge assigns the IDs for any matching dimension values to the Endeca records. Any source property values that do not have matching dimension values in the dimension hierarchy are ignored.

In order for a source property value to match a dimension value, the dimension value's definition must contain a synonym that:

- Is an exact text match to the source property value.
- Has its **Classify** option enabled.

Example

This example shows the **Synonyms** dialog in the **Dimension Value editor** with a dimension value synonym that has its **Classify** option enabled:



Must Match mode

Must Match behaves identically to Normal, with the exception that Must Match issues a warning for any source property values that do not have matching dimension values.

Related Links

The Forge Logging System on page 109
This section provides a brief introduction to the Forge logging system. Its command-line interface allows you to focus on the messages that interest you globally and by topic.

Auto Generate mode

Auto Generate specifies that Forge automatically generates a dimension value name and ID for any source property value that does not have a matching dimension value in the dimension hierarchy. Forge uses these automatically-generated names and IDs to tag the Endeca records the same as it would explicitly-defined dimension values.

Auto Generate mode dramatically reduces the amount of editing you have to do to the dimension hierarchy. However, auto-generated dimensions are always flat. Auto-generated names and IDs are persisted in a file that you specify as part of a dimension server component.

Related Links

Dimension server on page 49
Dimension servers work in conjunction with dimension adapters, and serve as a centralized source of dimension information for all other pipeline components.

Rules of thumb for dimension mapping

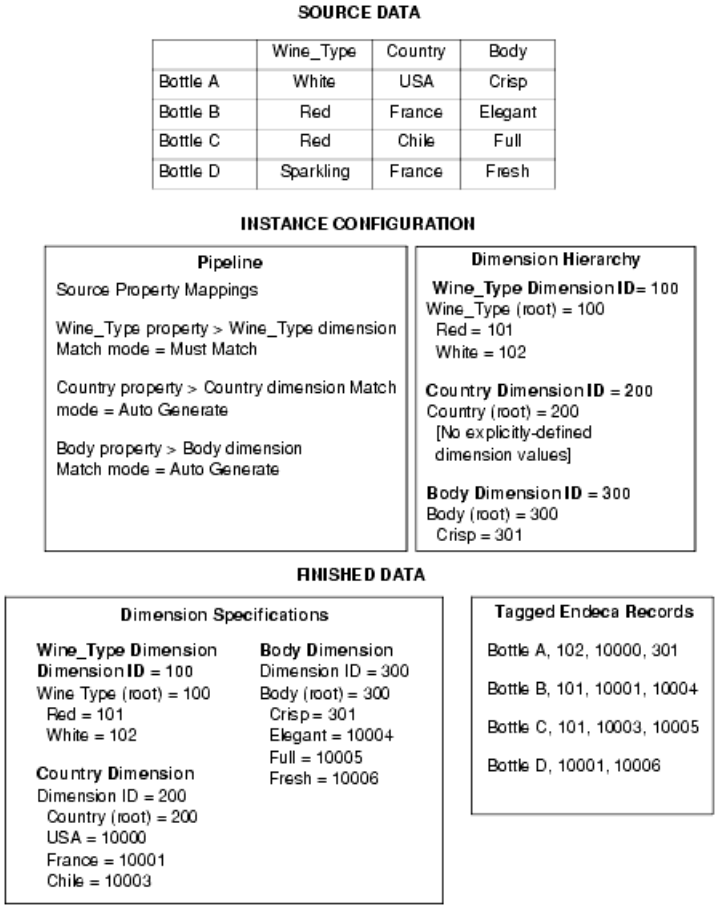
When you choose the match mode to use for generating your dimension values, keep in mind the following two rules of thumb:

- If you manually define dimension values in the dimension hierarchy, the Normal, Must Match, and Auto Generate features behave identically with respect to those dimension values.
- Any source property value that does not have a matching dimension value specified in the dimension hierarchy will not be mapped unless you have set the dimension to Auto Generate in the pipeline.

Dimension mapping example

The following illustration shows a simple dimension mapping example that uses a combination of generation methods. The sections after the illustration describe the mapping behavior in the example.

Dimension mapping



Wine_Type dimension

The Red and White property values have matching Red and White dimension values specified in the dimension hierarchy. These property values are mapped to the Red and White dimension value IDs, respectively. Bottles

B and C are tagged with the Red dimension value ID, and Bottle A is tagged with the White dimension value ID.

The Sparkling property value does not have a matching dimension value in the dimension hierarchy. The Wine Type dimension is set to Must Match, so this property is ignored and a warning is issued. As a result, Bottle D does not get tagged with a dimension value ID from the Wine Type dimension.

Country dimension

There are no dimension values explicitly defined in the dimension hierarchy for the Country dimension. However, this dimension is set to Auto Generate, so all three of the Country property values (USA, France, and Chile) are mapped to automatically-generated dimension value IDs.

Bottle A is tagged with the auto-generated ID for the USA dimension value. Bottles B and D are tagged with the auto-generated ID for the France dimension value. Bottle C is tagged with the auto-generated ID for the Chile dimension value.

Body dimension

The Crisp property value has a matching dimension value specified in the dimension hierarchy, so the Crisp property value is mapped to the Crisp dimension value. Bottle A is tagged with the Crisp dimension value ID.

The other three property values (Elegant, Full, and Fresh) do not have matching dimension values in the dimension hierarchy but, because the Body dimension is set to Auto Generate, these three property values are mapped to automatically-generated dimension value IDs.

Bottle B is tagged with the auto-generated ID for the Elegant dimension value. Bottle C is tagged with the auto-generated ID for the Full dimension value. Bottle D is tagged with the auto-generated ID for the Fresh dimension value.

Regardless of how they were generated, all of the dimension value IDs are included in the finished data that Forge produces for indexing.

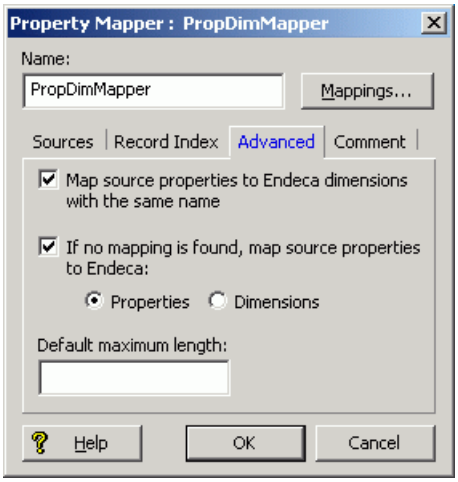
Chapter 5

Advanced Mapping Techniques

You can specify mapping techniques and default behavior using the **Property Mapper editor Advanced** tab.

The Property Mapper editor Advanced tab

The **Property Mapper editor Advanced** tab (shown below) lets you configure advanced mapping techniques when you are building prototypes.



The following sections describes these techniques.

Important: Oracle strongly recommends that you use the explicit mapping techniques, because the advanced mapping techniques can have unexpected results if you are not careful when using them.

About enabling implicit mapping

The first advanced option, **Map source properties to Endeca dimensions with the same name**, enables implicit mapping.

When implicit mapping is enabled, any source property that has a name that is identical to an existing dimension is automatically mapped to that dimension. The like-named dimension, and any of its constituent dimension

values, must already exist in your dimension hierarchy (in other words, you've already defined them using the **Dimensions** and **Dimension Values editors**).

Implicit mapping uses the Normal mapping mode where only those source property values that have a matching dimension value explicitly defined in the dimension hierarchy are mapped. Forge assigns the IDs for any matching dimension values to the Endeca records. Any source property values that do not have matching dimension values in the dimension hierarchy are ignored.

Note: Implicit mapping is limited to mappings between source properties and dimensions. This means that implicit mapping cannot take place between source properties and Endeca properties. In addition, implicit mapping only works if no explicit mapping exists.

Enabling default mapping

The default mapping option defines the default that Forge uses to handle source properties that have neither explicit nor implicit mappings. There are three possible settings.

Use the default mapping option with caution because:

- With this option enabled, all source properties will ultimately be mapped and mapped properties use system resources. Ideally, you should only map source properties that you intend to use in your implementation so that you minimize the use of system resources.
- Many production-level implementations automatically pull and process new data when it is available. If this data has new source properties, these properties will be mapped and included in your MDEX Engine indices. Again, this uses system resources unnecessarily but, perhaps more importantly, this situation may also result in the display of dimensions or Endeca properties that you do not want your users to see.

To set the default mapping options:

1. Select the **Advanced** tab in the **Property Mapper editor**.
   The tab includes the following option:

   If no mapping is found, map source properties to Endeca:

   - **Properties**
   - **Dimensions**

2. Select one or neither of the two settings:

Option	Description
Neither	Uncheck the option altogether to ignore source properties that do not have an explicit or implicit mapping defined.
Properties	Check **Property** to create a mapping between the source property and an Endeca property. Forge does this by creating a new Endeca property that uses the same name and value as the source property and assigning it to the record.
Dimensions	Check **Dimension** to create a mapping between the source property and a dimension. Forge does this by creating a new dimension, using the source property's name. Forge uses the Auto Generate mode to populate the dimension with dimension values that match the source property's values.

About the default maximum length for source property values

The **Default Maximum Length** option defines the maximum source property value length allowed when creating mappings. Source properties that have values that exceed this length are not mapped, and a warning is issued by the Forge Logging system, if so configured.

If you *do not* explicitly specify a Default Maximum Length, Forge checks against the following limits when determining whether to map a value:

- Source properties that are mapped to Endeca properties can have values of any length.
- Source properties that are mapped to dimensions must have values that are 255 characters or less.

If you do explicitly specify a Default Maximum Length, that length is applied to both Endeca property and dimension mappings.

Related Links

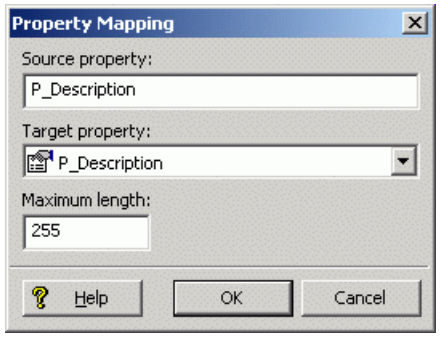
The Forge Logging System on page 109
This section provides a brief introduction to the Forge logging system. Its command-line interface allows you to focus on the messages that interest you globally and by topic.

About overriding the default maximum length setting

You can override the **Default Maximum Length** setting on a *per-mapping* basis by using the **Maximum Length** field in both the **Property Mapping** and **Dimension Mapping editors**.

Example

Suppose you use the Default Maximum Length to limit the length of all your source property mappings to be 100 characters. However, you want to allow the P_Description property to have a greater limit (say, 255 characters). You would then use the **Property Mapping editor** to set an override for the P_Description source property that allows the description to be up to 255 characters:



Chapter 6

Before Building Your Instance Configuration

Before you start building your instance configuration, you must create a directory structure to support your data processing back end.

Endeca Application Controller directory structure

While the Endeca Application Controller builds the directory structure it requires, you first have to build two directories:

- **Endeca instance configuration directory** — You create this directory and its contents with Developer Studio (using the **File > New Project** menu). The directory contains the Developer Studio project file, the baseline pipeline file, the partial updates pipeline file (if you are running partial updates), and the index configuration files (XML). You then use Developer Studio to send the instance configuration to Endeca Workbench.
- **Incoming directory** — This directory contains the source data to be processed by Forge. You then provision this directory in Endeca Workbench by using the **EAC Administration > Admin Console** menu, and selecting the **Forge** component tab.

You must create these directories before you use Endeca Workbench to provision your application and its components to the EAC. Be sure to copy your source data to the `incoming` directory on the machine that will be running Forge. This is the location where Forge looks for source data.

Pipeline overview

Your *pipeline* functions as the script for the entire data transformation process that occurs when you run the Forge program. The pipeline specifies things like the format and location of the source data, any changes to be made to the source data (*standardization*), and the mapping method to use for each of the source data's properties.

A pipeline is composed of a collection of components. Each component performs a specific function during the transformation of your source data into Endeca records. Components are linked together by means of cross-references, giving the pipeline a sequential flow.

About adding and editing pipeline components

You add and edit pipeline components using the **Pipeline Diagram editor** in Developer Studio.

The pipeline diagram depicts the components in your pipeline and the relationship between them. It describes the flow of events that occur in the process of converting raw data to a format that the Endeca MDEX Engine can use, making it easy for you to trace the logic of your data model. The pipeline diagram is the best way to maneuver and maintain a high-level view of your pipeline as it grows in size and complexity.

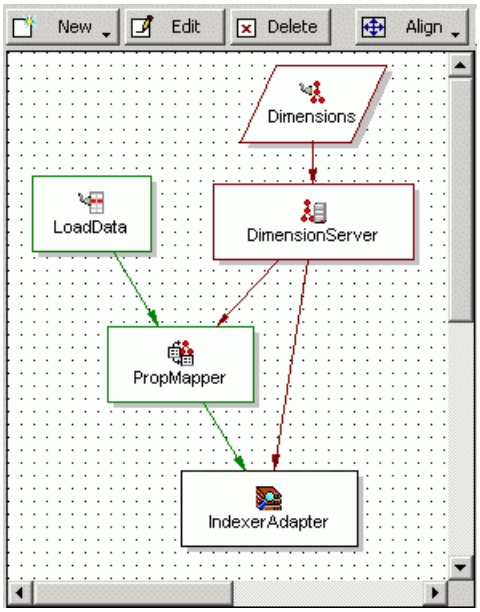
For details on adding and editing pipeline components, see the *Oracle Endeca Developer Studio Help*.

About creating a data flow using component names

You must give every component in your pipeline a unique name that identifies it to the other components. You use these names to specify cross-references between components, effectively creating a flow of data through the pipeline.

Pipeline Example

For example, by tracing the data flow backwards in the following illustration and starting from the bottom, you can see that:



1. IndexerAdapter gets its data from PropMapper and DimensionServer.
2. PropMapper gets its data from LoadData and DimensionServer.
3. DimensionServer gets its data from Dimensions.
4. LoadData and Dimensions both get their data from source files (this is indicated by the lack of arrows feeding them).

When you specify a data source within a component's editor, you are indicating which of the other components will provide data to that component. Components can have multiple data sources, such as the PropMapper component above, which has both a record source, LoadData, and a dimension source, DimensionServer.

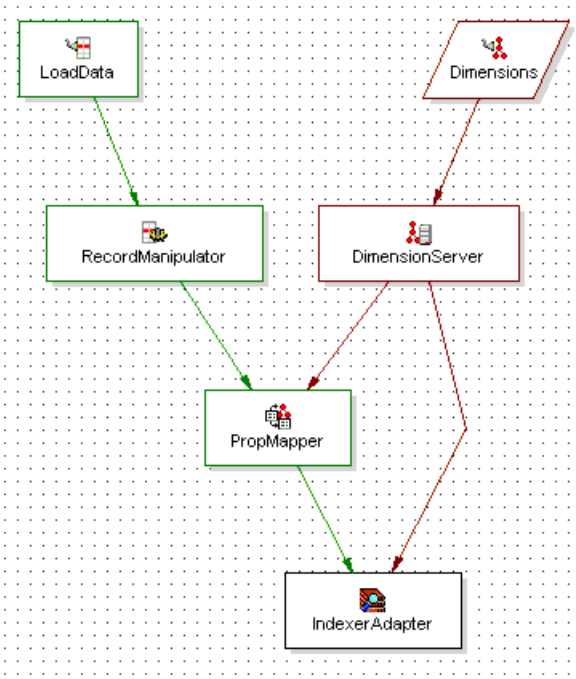
Pipeline Example: Adding a Pipeline Component

Alternatively, you can connect pipeline components graphically in the **Pipeline Diagram editor**.

When you add and remove components, you must be careful to make any data source changes required to maintain the correct data flow. To illustrate this point, the example above is modified to include another

component, RecordManipulator, that comes between LoadData and PropMapper in the data flow of the pipeline. Adding RecordManipulator in this location requires that:

- RecordManipulator's data source is set to LoadData.
- PropMapper's data source is changed to RecordManipulator.



Similar care must be taken when removing a component from a pipeline.

URLs in the pipeline

Some of the components in the pipeline require URLs that point to external files, such as source data files. All of these URLs are relative to the location of the `Pipeline.epx` file.

This file contains the pipeline specifications that you have created in Developer Studio. Developer Studio automatically generates a `Pipeline.epx` file when you create a new project and saves it in the same directory as your .esp project file.

Note: As a rule, you should not move the `Pipeline.epx` file, or any other automatically generated files, from their location in the same directory as the `.esp` project file.

Chapter 7

About Creating a Basic Pipeline

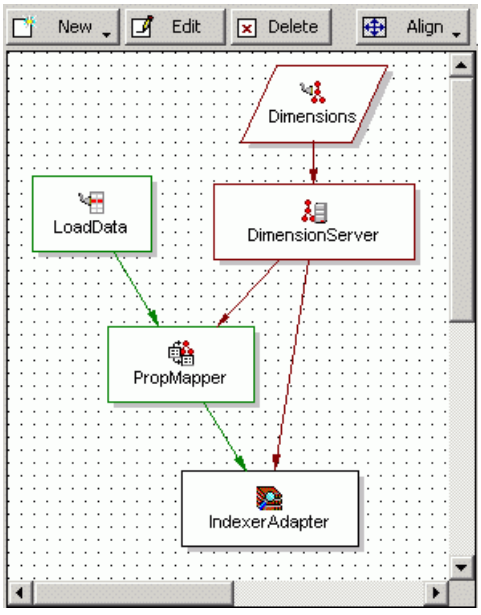
Endeca Developer Studio provides a Basic Pipeline template that helps you get started when building your pipeline from scratch. The goal of the Basic Pipeline template is to get you up and running with a working pipeline as quickly as possible. A working pipeline is defined as a pipeline that can read in source records and output finished records, ready for indexing.

The Basic Pipeline template

The Basic Pipeline template streamlines the setup for a pipeline that contains the following five components:

- Record adapter (LoadData) for loading source data.
- Property mapper (PropMapper) for mapping source properties to Endeca properties and dimensions.
- Indexer adapter (IndexerAdapter) for writing out data that is ready to be indexed by the Dgidx program.
- Dimension adapter (Dimensions) for loading dimension data.
- Dimension server (DimensionServer) that functions as a single repository for dimension data that has been input via one or more dimension adapters.

The following illustration shows the pipeline diagram for a basic pipeline:



Oracle recommends that you leave most of the Basic Pipeline component options at their default settings and customize them later, after you have a basic pipeline functioning. Endeca also recommends that you do not include other components to perform additional tasks until after you have a functioning pipeline. The remainder of this section describes how to get a Basic Pipeline working.

Note: This section does not describe all of the features of a basic pipeline's components in exhaustive detail. It describes the minimum you need to know to create a functioning pipeline. Detailed information on individual components is included in subsequent chapters of this book and in the *Oracle Endeca Developer Studio Help*.

Record adapters

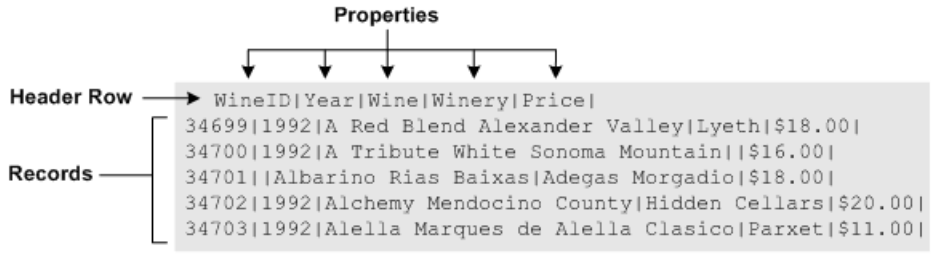
Record adapters load and save records in a variety of formats, including delimited, binary, ODBC (Windows only), JDBC, and Microsoft Exchange. Each record adapter format has its own set of attributes.

This section describes the most common type of record adapter: an input record adapter that loads data stored in delimited format. See the *Oracle Endeca Developer Studio help* for detailed information on the other record adapter types.

Note: Output record adapters are primarily used as a diagnostic tool and for translating formats.

Source data in delimited format

A delimited file is a rectangular file with columns and rows separated by specified characters. Each row corresponds to a record and each column corresponds to a property.



The records in a delimited file must have identical properties, in terms of number and type, although it is possible for a record to have a null value for a property.

About the Record Index tab

The **Record Index** tab allows you to add dimensions or properties that are used in the record adapter's record index to control the order in which records are read in for downstream components.

A record index is used to support join functionality, and is needed only if a downstream component will need to request records by ID. For example, a cache needs to be able to respond to a record assembler's (left join) request for a particular record.

If the order of the records being used by the downstream component do not matter, then you should not add a record index to the record adapter. For example, a switch join does not require a record index on components above it because it does not matter what order the records are pulled in.

If the record adapter has a record index that is not required, you may see a Forge log WARN message about an ID conflict, as illustrated by the following example:

```
FORGE {baseline}: The RecordAdapter 'LoadMainData' has records
that do not follow the index specified.
Record number '14' violates the index sort order with record key
[R_VHNR] => {'PVal [value= 361945]'} (the previous record key
was [R_VHNR] => {'PVal [value= 957483]'})!
```

If you see this warning, remove the record index from the record adapter and Forge will stop removing records that do not conform to the record index.

Note: There are two cases where join keys are not required for data sources and, hence, neither are record indexes.

Related Links

Joins that do not require record caches on page 87
There are two join cases that do not require record caches:

Dimension adapter

You use dimension adapters to load dimension data.

When you create a new project in Developer Studio, a default dimensions file, called `Dimensions.xml`, is created for you and stored in the same directory as your `.esp` project file. As you make changes to your dimension hierarchy in Developer Studio, this file is updated to reflect the changes.

Note: Dimension adapters can also save dimension information for diagnostic purposes. Saving dimensions is an advanced topic and it is not covered in this section.

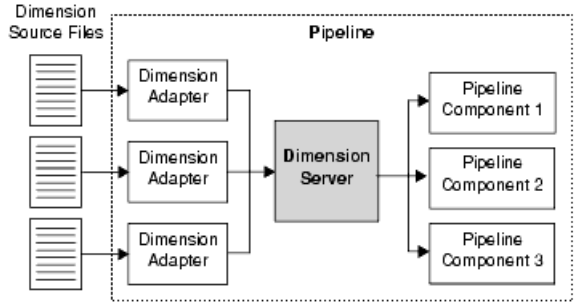
Dimension server

Dimension servers work in conjunction with dimension adapters, and serve as a centralized source of dimension information for all other pipeline components.

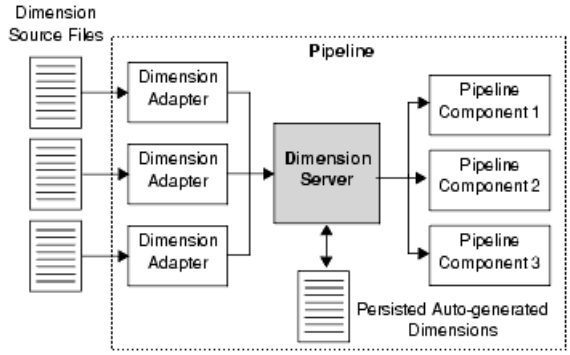
Dimension information typically follows the path outlined below:

1. Dimension adapters load dimension information from your dimension source files.
2. The dimension server gets its dimension information from the dimension adapters.
3. Other pipeline components get their dimension information from the dimension server.

Setting up your pipeline with a dimension server allows you to change your dimension adapters as needed without having to change the dimension source for all other pipeline components that require dimension information.



In addition to functioning as a centralized source for dimension information, dimension servers also coordinate the loading and saving of dimension information that is generated when using the Auto Generate option during source property-to-dimension mapping. Auto-generated dimensions are persisted in the file location that is specified as part of the dimension server component.



Typically, there is only one dimension server per pipeline.

Related Links

Overview of Source Property Mapping on page 27

The property mapper is a pipeline component used to map properties on the records in your source data to Endeca properties and/or dimensions to make them navigable, displayable, both, or neither. The property mapper is a key component in developing a pipeline, so it is important to understand its functions well.

Property mapper

You use a property mapper component to establish mappings between source properties, and Endeca properties and dimensions. These mappings dictate which dimension values are tagged to the current record and which property information is available for record search and display.

Oracle strongly recommends that you have only one property mapper per pipeline.

At a minimum, a property mapper requires both a record source and a dimension source to define the components that will supply it with record and dimension data. You can leave the other settings at their defaults while developing your initial working pipeline.

Important: The property mapper is a crucial component and you should be very familiar with its settings.

Related Links

Overview of Source Property Mapping on page 27

The property mapper is a pipeline component used to map properties on the records in your source data to Endeca properties and/or dimensions to make them navigable, displayable, both, or neither. The property mapper is a key component in developing a pipeline, so it is important to understand its functions well.

Indexer adapter

An indexer adapter writes out data that is ready to be indexed by the Dgidx program. An indexer adapter requires two data sources: one for record data and one for dimension data. Typically, there is only one indexer adapter per pipeline.

Chapter 8

About Running Your Basic Pipeline

After you have created your basic pipeline, you should run it and view the results. Your initial goal is to make sure that your source data is running through the entire pipeline and being incorporated into the MDEX Engine indices.

Running a pipeline

This task describes the steps you use to run your basic pipeline.

The Basic Pipeline template does not contain a source data file. Therefore, before you run the Basic Pipeline, make sure you have created an incoming directory that contains source data. Alternatively, you can use the `incoming` directory in the sample_wine_data reference implementation.

See the *Oracle Endeca Workbench Administrator's Guide* for more details on running a pipeline under the Endeca Application Controller.

To run a pipeline:

1. In Endeca Workbench, provision your application and its components to the EAC Central Server, as documented in the *Oracle Endeca Workbench Administrator's Guide*.
2. In Developer Studio, use the **Tools > Endeca Workbench** menu option to send your instance configuration to **Endeca Workbench** by using the **Set Instance Configuration** option.
3. In Endeca Workbench, run a baseline update script to process your data and start the MDEX Engine (optionally, you can run a baseline update script using the eaccmd utility, or the custom Web services interface).

Viewing pipeline results in a UI reference implementation

Once you have an MDEX Engine running, you can use a generic front-end, called a *UI reference implementation*, to view the data. UI reference implementations are sample Web applications included with the Endeca distribution.

This procedure assumes that the JSP UI reference implementation that is shipped with the Endeca Workbench is running.

To test your basic pipeline using a UI reference implementation:

1. Open Internet Explorer 6.0 or later.

2. Navigate to the JSP reference implementation; for example:
   `http://localhost:8888/endeca_jspref`
3. Enter the host and port for your MDEX Engine and click **Go**.

At this point in the process, you should see a list of records but no Endeca properties or dimensions.

You must define and map Endeca properties and dimensions before they can appear in your Web application.

Related Links

After Your Basic Pipeline Is Running on page 55
After you get your basic pipeline running, you can begin crafting your Endeca implementation in earnest. Again, Oracle recommends a stepped approach where you implement a small set of features, test them to make sure your implementation is behaving as expected, and then implement additional features.

Chapter 9

After Your Basic Pipeline Is Running

After you get your basic pipeline running, you can begin crafting your Endeca implementation in earnest. Again, Oracle recommends a stepped approach where you implement a small set of features, test them to make sure your implementation is behaving as expected, and then implement additional features.

Additional tasks

Additional tasks you will most likely want to do include:

- Create Endeca properties and dimensions, and then map them to your source properties.
- Designate an Endeca property to be the record specifier.
- Add pipeline components for various tasks such as joining source data and manipulating source data properties.
- Specify additional index configuration settings such as search configuration, dimension groups, and so forth.

Important: The information in this section gives a high level overview of these additional tasks and is not intended to be complete. Refer to other sections in this documentation and the *Oracle Endeca Developer Studio Help* for detailed information on implementing the features listed here, as well as many others.

About source property mapping

Source property mappings dictate which dimension values are tagged to each record and which property information is available for record search, sort, and display.

Before you can map a source property to an Endeca property or dimension, you must create the Endeca property or dimension. This section covers how to create Endeca properties and dimensions as well as how to map source properties to them. It also tells you how to create null mappings.

Source properties can be mapped in three different ways. They can be:

- Mapped to an Endeca property (for search, sort, and display only).
- Mapped to a dimension (for search, sort, display, and navigation).
- Ignored by specifying a null mapping.

Note: The mapping described in this section is known as explicit mapping. In general, this is the type of mapping Oracle recommends that you use.

Related Links

Overview of Source Property Mapping on page 27

The property mapper is a pipeline component used to map properties on the records in your source data to Endeca properties and/or dimensions to make them navigable, displayable, both, or neither. The property mapper is a key component in developing a pipeline, so it is important to understand its functions well.

Adding and mapping Endeca properties

Preparing an Endeca property for display within an Endeca implementation is a two-step process.

Note: The UI reference implementation has been written to iterate over all the Endeca properties that are returned with a query and display them, so you don't have to do any additional coding to get the Endeca property to display in the UI.

You must:

1. Add the Endeca property to your project. You do this in the **Property editor** in Developer Studio.
2. Create a mapping between a source property and the Endeca property. You do this in the **Property Mapper editor** in Developer Studio.

   This step instructs the Data Foundry to populate the Endeca property with the value from the source property. Without this mapping, the Endeca property will not be available for display.

Continue adding Endeca properties and mapping them to source properties. You can map multiple source properties to a single Endeca property.

Adding and mapping dimensions

Similar to creating an Endeca property, the process for adding a dimension to your implementation has several steps.

To create a dimension, you must:

1. Add the dimension to your project. You do this in the **Dimension editor** in Developer Studio.
2. Add any dimension values that you want to create manually.
3. Create a mapping between a source property and the dimension in the Developer Studio **Property Mapper editor**. Without this mapping, the dimension will be removed from the MDEX Engine.

Related Links

Overview of Source Property Mapping on page 27

The property mapper is a pipeline component used to map properties on the records in your source data to Endeca properties and/or dimensions to make them navigable, displayable, both, or neither. The property mapper is a key component in developing a pipeline, so it is important to understand its functions well.

About synonyms

Synonyms provide a textual way to refer to a dimension value, rather than by ID alone. You specify the way each synonym is used by the MDEX Engine in the **Dimension Value Synonyms editor** in Developer Studio.

A dimension value can have multiple synonyms. You can choose from **Search**, **Classify**, and **(Display)** options as follows:

- Enabling the **Search** option indicates that this synonym should be considered during record and dimension searches. You can enable search for multiple synonyms, allowing you to create a more robust dimension value for searching.
- Enabling the **Classify** option indicates that this synonym should be considered when attempting to map a source property value to this dimension value. In order for a source property value to match a dimension value, the dimension value's definition must contain a synonym that:
  - Is an exact text match to the source property value.
  - Has its **Classify** option enabled.

  If a synonym does not have its **Classify** option enabled, it is ignored during mapping, regardless of whether or not it is a text match to a source property value.

  Again, by enabling classification for multiple synonyms, you increase the mapping potential for a dimension value because a source property can map to any of the synonyms that have been marked with **Classify**.
- While you can have multiple synonyms for a dimension value, only one synonym can be marked for display. This is the synonym whose text is displayed in your implementation whenever this dimension value is shown. By default, the first synonym you create is set to be displayed, as is indicated by the parentheses around the synonym's name, but you can set any synonym for display in the **Synonyms** dialog box

To better understand these three options, consider the following example.

Example

This dimension value has an ID of 100 (automatically assigned by Developer Studio) and three synonyms:

```
Dimension Value ID = 100
Synonyms =
 2002 SEARCH=enabled CLASSIFY=enabled DISPLAY=yes
 '02 SEARCH=enabled CLASSIFY=enabled DISPLAY=no
 02 SEARCH=enabled CLASSIFY=enabled DISPLAY=no
```

In this example, records with source property values matching any of the following terms would be tagged with the dimension value ID 100, and dimension searches on those terms would return that dimension value ID:

```
2002
```

```
'02
```

```
02
```

Additionally, anytime the dimension value with an ID of 100 is displayed in the implementation, the text used to represent the dimension value is "2002".

After you have created the dimension and defined any manual dimension values, you create the mapping between a source property and the dimension.

Note: The UI reference implementation has been written to iterate over all the dimensions that are returned with a query and display them, so you don't have to do any additional coding to get the dimension to display in the UI.

Continue adding dimensions and mapping them to source properties. You can map multiple source properties to a single dimension.

About null mappings

A null mapping, set in the Developer Studio **Property Mapper editor**, indicates that a source property should be ignored.

Explicit null mappings provide a means to prevent an automated mapping from being formed for a particular source property. In other words, you can enable automated mapping, and then turn off mapping for selected source properties using explicit null mappings.

Related Links

Types of source property mapping on page 29
There are four types of source property mappings:

Setting the record specifier property

Developer Studio lets you configure how records should be identified by your application. The `RECORD_SPEC` attribute allows you to specify the property that you wish to use to identify specific records.

Records can have only one record spec during updates and at startup. You may set the `RECORD_SPEC` attribute's value to `TRUE` in any property where the values for the property meet the following requirements:

- The value for this property on each record must be unique.
- Each record should be assigned exactly one value for this property.

Only one property in the project may have the `RECORD_SPEC` attribute set to `TRUE`.

For example, Forge uses the `RECORD_SPEC` property value to identify the records that it is transforming. If the project does not have a designated the `RECORD_SPEC` property, Forge assigns a unique record specifier value to each record. As another example, implementing partial updates requires that the project have an assigned `RECORD_SPEC` property.

Although it is valid for a project to not have a specific `RECORD_SPEC` property, it is recommended that you assign one. For example, you may wish to use a field such as `UPC`, `SKU`, or `part_number` to identify a record.

To configure a `RECORD_SPEC` attribute for an existing property:

1. In the **Project** tab of Developer Studio, double-click **Properties**.
2. From the **Properties** view, select a property and click **Edit**.
   The **Property editor** is displayed.
3. In the **General** tab, check **Use for Record Spec**.
4. Click **OK**.
   The **Properties** view is redisplayed.
5. Select **File > Save**.

About specifying dimensions and dimension value order

The MDEX Engine returns dimensions and dimension values in the order in which they are specified in the Developer Studio **Dimensions** and **Dimension Values editors**, respectively. As a result, you may want to reorder your dimensions and dimension values to better control their display.

Additional pipeline components

After you have added your dimensions and Endeca properties to your project, you may want to include other pipeline components to perform additional tasks. The following table describes the components you can add:

Component	Description	For More Info
Record assemblers	Join data from one or more secondary data sources to the current record.	"Adding a record assembler" in this guide and in the *Oracle Endeca Developer Studio Help*.
Record caches	Store a temporary copy of record data that has been read in by a record adapter. Record caches are generally used in conjunction with record assemblers and are set up to contain data from secondary data sources.	"Adding a record cache" in this guide and in the *Oracle Endeca Developer Studio Help*.
Java manipulators	A Java manipulator is your own code in Java that you can use to perform data manipulation on properties and records. Java manipulators provide you with the most generic way of changing records in the Forge pipeline. A Java manipulator contains a class that is based on the Java API **Adapter** interface in the Content Adapter Development Kit (CADK).	For information on how to write your own Java manipulator and for a sample code, see the *Endeca Content Adapter Development Kit (CADK) Guide*.
Perl manipulators	Allow you to write custom Perl code that changes the data associated with an Endeca record. Perl manipulators are useful for such tasks as manually adding or removing source properties, changing the value of a source property, retrieving records based on a particular key, and so on.	See "Using Perl Manipulators to Change Source Properties" in the *Oracle Endeca Developer Studio Help*. For details on Perl code syntax, see the *Endeca Forge API Guide for Perl*.
Spiders	Crawl document hierarchies on a file system or over HTTP. From a root URL, a spider spools URLs of documents to crawl.	"Creating a spider" in this guide.
Record manipulators	Provide support, such as URL extraction, for a content acquisition system, such as a crawler implementation.	See "Record Manipulators and Expressions" in the *Oracle Endeca Developer Studio Help*.

Component	Description	For More Info
Update adapters	Provide support for partial (rapid) updates.	See the *Endeca Partial Updates Guide*.

Related Links

Adding a record cache on page 79

Use the options in the **Record Cache editor** to add and configure a record cache for each of your record sources.

Additional index configuration options

The Endeca MDEX Platform offers a rich set of index configuration options that allow you to customize your Endeca implementation. You use the index configuration to specify things like search configurations, precedence rules, dynamic business rules, and so on.

The major index configuration features are described in the table below. Refer to other sections of this guide as well as to the *Endeca Basic Development Guide* and the *Endeca Advanced Development Guide* for information on all of the features you can choose to implement.

Component	Description	For More Info
Dimension groups	Allow you to organize dimensions into explicit groupings for presentation purposes.	See the "Working with Dimensions" chapter in *Endeca Basic Development Guide*. See "Configuring Dimension Groups" in the *Oracle Endeca Developer Studio Help*.
Search interfaces	Allow you to control record search behavior for groups of one or more properties or dimensions. Some of the features that can be specified for a search interface include relevance ranking, matching across multiple properties and dimensions, and partial matching.	See the "Working with Search Interfaces" chapter in *Endeca Basic Development Guide*. See "Configuring Search Interfaces" in the *Oracle Endeca Developer Studio Help*.
Thesaurus entries	The thesaurus allows the MDEX Engine to return matches for related concepts to words or phrases contained in user queries. For example, an thesaurus entry might specify that the phrase "Mark Twain" is interchangeable with the phrase "Samuel Clemens".	See the "Using Stemming and Thesaurus" chapter in the *Endeca Advanced Development Guide*. See "Configuring Search" in the *Oracle Endeca Developer Studio Help*. See the *Oracle Endeca Workbench Help*.

Component	Description	For More Info
Stop words	Stop words are words that are set to be ignored by the Endeca MDEX Engine. Typically, common words like "the" are included in the stop word list.	See the "Advanced Search Features" section in the *Endeca Advanced Development Guide*. See "Configuring Search" in *Oracle Endeca Developer Studio Help*.
Search characters	Allow you to configure the handling of punctuation and other non-alphanumeric characters in search queries.	See the "Search Characters" chapter in the *Endeca Basic Development Guide*. See "Configuring Search" in *Oracle Endeca Developer Studio Help*.
Stemming	Stemming allows the word root and word derivations of search terms to be included in search results. For example, a search for the term “children” would also consider “child” (which is the word root). This means that singular and plural forms of nouns are considered equivalent and interchangeable for all search operations. Preconfigured stemming files are shipped for supported languages. You cannot modify these files, but you can enable or disable stemming with Developer Studio.	See the "Using Stemming and Thesaurus" chapter in the *Endeca Advanced Development Guide*. See “Configuring Search” in the *Oracle Endeca Developer Studio Help*.
Precedence rules	Allow your Endeca implementation to delay the display of a dimension until the user triggers it, making navigation through the data easier and avoiding information overload.	See "Configuring Precedence Rules" in the *Oracle Endeca Developer Studio Help*.
Dynamic business rules	Dynamic business rules allow you to promote contextually relevant result records, based on data-driven rules, to users as they navigate or search within a dataset. For example, you can show a list of best-selling merlots when a user has navigated to a record set made up of merlots. Dynamic business rules make it possible to implement features such as merchandising and content spotlighting.	See "Promoting Records with Dynamic Business Rules" in the *Endeca Advanced Development Guide*. See "Configuring Dynamic Business Rules" in *Oracle Endeca Developer Studio Help*. See "Working with dynamic business rules" in the *Oracle Endeca Workbench Help*.

Part 2

Joins

- *Overview of Joins*
- *About Configuring Join Keys and Record Indexes*
- *About Implementing Joins*
- *Advanced Join Behavior*
- *Tips and Troubleshooting for Joins*

Chapter 10

Overview of Joins

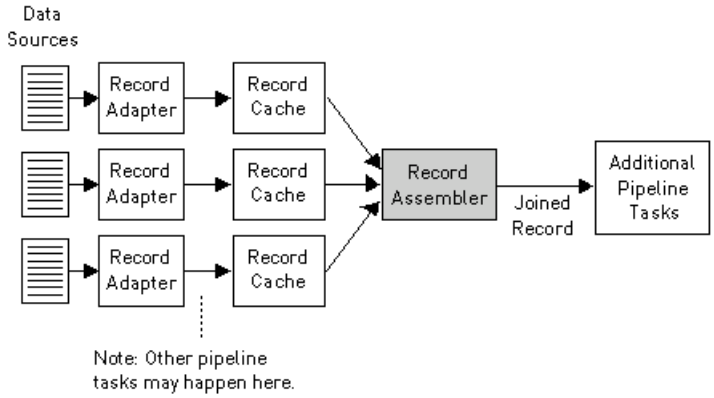
Generally, applications consist of more than one data source. For example, an application used to navigate books would have records that contain both title and author information. If the title and author source data reside in different locations, you would need to join them together to create a single record with both pieces of information.

Record assemblers and joins

You add a record assembler component to your pipeline to join data from one or more data sources. To use a record assembler, you must define:

- The data sources to be joined. With two exceptions, all data sources feeding a join must be record caches, described below.
- The type of join to perform.

Record caches give Forge random access to the data, allowing it to look up records by join key. Forge uses available RAM for the cache and then allocates hard drive space as necessary.



When you configure a join in a record assembler, you specify a *join key* for each source. Join keys are dimension or property names. Forge uses these keys to find equivalent records within the data sources participating in the join.

During a record assembly, the following happens:

1. Forge finds the value for the join key in the current record.
2. Forge looks for a matching value to the join key within the record cache. If Forge finds a record with a matching value, that record is considered equivalent to the current record.

3. Forge performs the join according to the configuration that you have specified.

Related Links

Joins that do not require record caches on page 87
There are two join cases that do not require record caches:

Overview of Joins on page 65
Generally, applications consist of more than one data source. For example, an application used to navigate books would have records that contain both title and author information. If the title and author source data reside in different locations, you would need to join them together to create a single record with both pieces of information.

About performing joins in a database

While the Data Foundry offers a large variety of join types and functionality, you are encouraged to perform joins within a database prior to exporting the information to the Data Foundry, if possible. The advantages of using a database to perform the join include:

- Many users are more familiar with this technology.
- Databases typically provide support for more data types.
- If the data is already in a database, existing indexes may be used, eliminating the need to recreate the index.
- Eliminating joins from your pipeline makes for simpler pipelines.
- Using the database, in some cases, may reduce I/O by collapsing data in the join.

However, it is not always possible to join information in a database. Data may exist outside of a database or in incompatible databases, may require a transformation prior to aggregation, and so on. It is for these cases that the Data Foundry provides its extensive join facility.

Join keys and record indexes

Join keys determine how records are compared by the record assembler. For each data source feeding a join, you designate one or more properties or dimensions to function as the source's join key.

During the course of the join, the record assembler compares the values within each source's join key. Records that have the same values for their respective keys are considered equivalent for the purposes of the join. With two exceptions, all joins require a join key for each data source.

Comparisons are based solely on property and dimension values, not names. It is not a requirement, therefore, that the properties and dimensions you specify for your record keys have identical names.

Example

As an example, consider the following left join with four record sources. Source 1 and Source 2 use `Id` as their join key. Source 3 and Source 4 use `Pid` as their join key. The other properties are not part of the join key for any of the sources.

Source 1 (Left)

Id	Prop1
A	Val1a
C	Val1c
B	Val1b

Join Key = Id

Source 2

Id	Prop2
C	Val2c
D	Val2d

Join Key = Id

Source 3

Pid	Prop3
A	Val3

Join Key = Pid

Source 4

Pid	Prop4
B	Val4

Join Key = Pid

For this data, we know:

- The join key for the first record in Source 1 is Id=A. The second record's key is Id=C. The third record's key is Id=B.
- The join key for the first record in Source 2 is Id=C. The second record's key is Id=D.
- The join key for the record in Source 3 is Pid=A.
- The join key for the record in Source 4 is Pid=B.

The resulting left join looks like this:

Results of Left Join

Id	Prop1	Prop2	Prop3	Prop4
A	Val1a		Val3	
B	Val1b			Val4
C	Val1c	Val2c		

In this example, the following occurred:

- Record Id=A from Source 1 is joined to record Pid=A from Source 3.
- Record Id=B from Source 1 is joined to record Pid=B from Source 4.
- Record Id=C from Source 1 is joined to record Id=C in Source 2.
- Record Id=D from Source 2 has no equivalent in the left source, so it is discarded.

Note: Join keys rarely incorporate dimensions. One reason is that if you use dimensions in a key, the records must have previously been processed and mapped by Forge. That is, the records must have the dimensions tagged on them before the join begins.

Related Links

Joins that do not require record caches on page 87
There are two join cases that do not require record caches:

About matching record indexes for join sources

In addition to a join key, you must also configure a record index for each data source that feeds a join. A record index is a key that indicates to the record assembler how it can identify records from that source.

A source's record index key must match its join key. In other words, the key that tells the record assembler how to find a source's records must be the same as the key that the record assembler uses to compare records from that source.

Note: There are two cases where join keys are not required for data sources and, hence, neither are record indexes.

Related Links

Joins that do not require record caches on page 87
There are two join cases that do not require record caches:

About Configuring Join Keys and Record Indexes on page 75
In addition to a join key, you must also configure a record index for each data source that feeds a join. A record index is a key that indicates to the record assembler how it can identify records from that source.

Join types

The following sections describe the join types supported by the Data Foundry. Each section provides a simple example for the join type being discussed. Note that while most of the examples use two record sources, many of the join types accept more than two sources, while other join types accept only one. Also note that in the examples, `Id` is the name of the join key for all sources.

Left join

With a left join, if a record from the left source compares equally to any records from the other sources, those records are combined. Records from the non-left sources that do not compare equally to a record in the left source are discarded.

In a left join, records from the left source are always processed, regardless of whether or not they are combined with records from non-left sources.

In the example below, the left source is Source 1. Records A, C, and D from Source 1 are combined with their equivalents from Source 2. Record E is discarded because it comes from a non-left source and has no equivalent in the left source. Record B is not combined with any other records, because it has no equivalent in Source 2, but it is still processed because it comes from the left source.

Source 1 (Left source)

Id	Name	Brand
A	Shirt	Acme
B	Pants	ABC
C	Sweater	ABC
D	Shirt	Acme

Source 2

Id	Retail	Wholesale
A	$40	$36
C	$55	$48
D	$35	$30
E	$50	$45

Results of Left Join

Id	Name	Brand	Retail	Wholesale
A	Shirt	Acme	$40	$36
B	Pants	ABC		
C	Sweater	ABC	$55	$48
D	Shirt	Acme	$35	$30

Inner join

In an inner join, only records common to all sources are processed. Records that appear in all sources are combined and the combined record is processed. Records that do not exist in all sources are discarded.

In the example below, Records A, C, and D are combined and processed. Records B and E are not common to all sources and are discarded.

Source 1

Id	Name	Brand
A	Shirt	Acme
B	Pants	ABC
C	Sweater	ABC
D	Shirt	Acme

Source 2

Id	Retail	Wholesale
A	$40	$36
C	$55	$48
D	$35	$30
E	$50	$45

Results of Inner Join

Id	Name	Brand	Retail	Wholesale
A	Shirt	Acme	$40	$36
C	Sweater	ABC	$55	$48
D	Shirt	Acme	$35	$30

Outer join

In an outer join, all records from all sources are processed. Records that compare equally are combined into a single record.

With an outer join, records that do not have equivalents in other data sources are not combined, but are still processed and included in the join output. An outer join requires two or more record sources.

In the example below, Records A, C, and D have equivalents in both Source 1 and Source 2. These records are combined. Records B and E do not have equivalents but they are still processed. As a result, Record B does not have values for `Retail` and `Wholesale` because there is no Record B in Source 2. Correspondingly, Record E has no values for `Name` and `Brand` because there is no Record E in Source 1.

Source 1

Id	Name	Brand
A	Shirt	Acme
B	Pants	ABC
C	Sweater	ABC
D	Shirt	Acme

Source 2

Id	Retail	Wholesale
A	$40	$36
C	$55	$48
D	$35	$30
E	$50	$45

Results of Outer Join

Id	Name	Brand	Retail	Wholesale
A	Shirt	Acme	$40	$36
B	Pants	ABC		
C	Sweater	ABC	$55	$48
D	Shirt	Acme	$35	$30
E			$50	$45

Disjunct join

In a disjunct join, only records that are unique across all sources are processed. All other records are discarded.

In this example, records B and E are unique across all sources, so they are processed. Records A, C, and D are not unique and therefore are discarded. Note that, in this example, the results for the join appear odd, because a record will never have both `Name/Brand` properties and `Retail/Wholesale` properties. Typically, this join is most useful when working with sources that share a common set of properties.

Source 1

Id	Name	Brand
A	Shirt	Acme
B	Pants	ABC
C	Sweater	ABC
D	Shirt	Acme

Source 2

Id	Retail	Wholesale
A	$40	$36
C	$55	$48
D	$35	$30
E	$50	$45

Results of Disjunct Join

Id	Name	Brand	Retail	Wholesale
B	Pants	ABC		
E			$50	$45

Switch join

In a switch join, given *N* sources, all records from Source 1 are processed, then all records from Source 2, and so on until all records from all *N* sources have been processed.

Note that records are never compared or combined, and all records from all sources are processed. Generally, a switch join is applied to sources that have similar properties but unique records, with respect to record keys, across the sources.

In this example, all the records from Source 1 are processed, then all the records from Source 2 are processed.

Source 1

Id	Name	Brand
A	Shirt	Acme
B	Pants	ABC
C	Sweater	ABC
D	Shirt	Acme

Source 2

Id	Name	Brand
J	Pants	AAA
K	Shoes	Acme
L	Shirt	ABC
M	Shirt	AAA

Results of Switch Join

Id	Name	Brand
A	Shirt	Acme
B	Pants	ABC
C	Sweater	ABC
D	Shirt	Acme
J	Pants	AAA
K	Shoes	Acme
L	Shirt	ABC
M	Shirt	AAA

Sort switch join

In a sort switch, all records from all sources are processed in such a way as to maintain the record index. The record index specifies that records should be processed in a sorted order, determined by record key comparison.

With a sort switch join, records are never combined. If a record from Source 1 compares equally to a record from Source 2, the record from Source 1 is processed first, consistent with the order of the sources as specified in the join settings.

In the example below, records A, C, and D are common to both Source 1 and Source 2. For each of these records, the Source 1 instance is processed before the Source 2 instance. Records B and E do not have equivalents, but they are processed in the order dictated by the record index which is, in this case, the Id key.

Source 1

Id	Name	Brand
A	Shirt	Acme
B	Pants	ABC
C	Sweater	ABC
D	Shirt	Acme

Source 2

Id	Retail	Wholesale
A	$40	$36
C	$55	$48
D	$35	$30
E	$50	$45

Results of Sort Switch Join

Id	Name	Brand	Retail	Wholesale
A	Shirt	Acme		
A			$40	$36
B	Pants	ABC		
C	Sweater	ABC		
C			$55	$48
D	Shirt	Acme		
D			$35	$30
E			$50	$45

First record join

In a first record join, the sources are prioritized such that, if a record from a higher priority source compares equally to records from lower priority sources, the record from the highest priority source is processed and the records from the lower priority sources are discarded.

Sources are listed in order of decreasing priority in the join configuration.

Records are never combined. The most common use of this join is for handling incremental feeds. For incremental feeds, history data (previously processed records) is given a lower priority and the latest data feed takes precedence. Records from the latest feed replace records in the history data, and records from the history data are processed only if a corresponding record does not exist in the latest feed.

In this example, records A, C, and D from Source 1 are processed, while their equivalents in Source 2 are discarded. Records B and E are both processed because they have no equivalents.

Source 1

Id	Name	Brand
A	Shirt	Acme
B	Pants	ABC
C	Sweater	ABC
D	Shirt	Acme

Source 2

Id	Retail	Wholesale
A	$40	$36
C	$55	$48
D	$35	$30
E	$50	$45

Results of First Record Join

Id	Name	Brand	Retail	Wholesale
A	Shirt	Acme		
B	Pants	ABC		
C	Sweater	ABC		
D	Shirt	Acme		
E			$50	$45

Combine join

A combine join combines like records from a single data source. Combine is a pseudo-join that operates on a single source.

In the example below, there are multiple records with Id=A, Id=C, and Id=D. These records are combined. Only one records exists for Id=B and Id=E, so neither of these records is combined, but both are processed and included in the joined data.

Source

Id	Name	Brand	Retail	Wholesale
A	Shirt	Acme		
A			$40	$36
B	Pants	ABC		
C	Sweater	ABC		
C			$55	$48
D	Shirt	Acme		
D			$35	$30
E			$50	$45

Results of Combine Join

Id	Name	Brand	Retail	Wholesale
A	Shirt	Acme	$40	$36
B	Pants	ABC		
C	Sweater	ABC	$55	$48
D	Shirt	Acme	$35	$30
E			$50	$45

Note: Combining large numbers of records will cause Forge to print warning messages about slow performance.

Related Links

Forge warnings when combining large numbers of records on page 89

When combining a large number of records (via either a Combine join or a record cache with the **Combine Records** setting enabled), Forge will issue a warning that performance may be slow. The default number of records at which this warning is issued is 100.

Chapter 11

About Configuring Join Keys and Record Indexes

In addition to a join key, you must also configure a record index for each data source that feeds a join. A record index is a key that indicates to the record assembler how it can identify records from that source.

Creating a record index

You specify a record index for a data source in the source's editor. The following example describes how to create a record index for a record cache.

We use a record cache in this example because, with two exceptions, all data sources that feed a join must be record caches.

To create a record index for a record cache:

1. In the pipeline diagram, double-click the record cache you want to edit to open it in the **Record Cache editor**.
2. Click the **Record Index** tab.
3. Click **Add**.
4. In the **Type** frame, do one of the following:
   - Choose **Custom Property**. Type a name for the property in the **Custom Property** text box.
   - Choose **Dimension**. Select a dimension name from the **Dimension** list.
5. (Optional) Repeat steps 2 and 3 to add additional dimensions or properties to the index.
6. (Optional) To reorder the components in the index, select a property or dimension and click **Up** or **Down**.
7. Click **OK**.

Example

The following illustration shows a record cache called `LeftDataCache` with a record index of `P_Name, P_Price`.

You specify a record cache's join key in the **Record Assembler editor** that uses the cache.

A source's record index key must match its join key. In other words, the key that tells the record assembler how to find a source's records must be the same as the key that the record assembler uses to compare records from that source.

Related Links

Joins that do not require record caches on page 87
There are two join cases that do not require record caches:

Join keys with multiple properties or dimensions on page 77
You can specify multiple properties or dimensions, called *key components*, for a single join key in order to join records based on more than one characteristic.

Creating a join key for a record cache on page 76
The following example describes how to create a join key for a record cache.

Creating a join key for a record cache

The following example describes how to create a join key for a record cache.

In addition to a join key, you must also configure a record index for each data source that feeds a join. A record index is a key that indicates to the record assembler how it can identify records from that source.

To create a join key for a record cache:

1. In the pipeline diagram, double-click the record assembler that uses the cache to open it in the **Record Assembler editor**.
2. Click the **Record Join** tab.
   The list of join entries corresponds with the data sources you specified in the **Sources** tab.
3. Select the record cache and click **Edit**.
   The **Join Entry editor** appears.
4. Click **Add**.
   The **Key Component editor** appears.
5. Using the steps below, create a join key that is identical to the record index key you created for the record cache.
   a) In the **Type** frame, do one of the following:

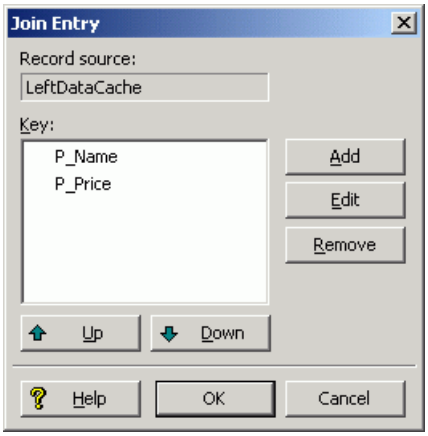
- Choose **Custom Property**. Type a name for the property in the **Custom Property** text box.
- Choose **Dimension**. Select a dimension name from the **Dimension** list.

b) Click **OK** to return to the **Join Entry editor**.
c) (Optional) Repeat these steps for each component you want to add to the key.
d) (Optional) To reorder the components in the key, select a component in the **Join Entry editor** and click **Up** or **Down.**
e) Click **OK** to close the **Join Entry editor**.

6. Repeat steps 3 through 5 for each record source that is participating in the join.
7. When you are done configuring your join, click **OK** to close the **Record Assembler editor**.

Example

The join key for `LeftDataCache` should look like this:



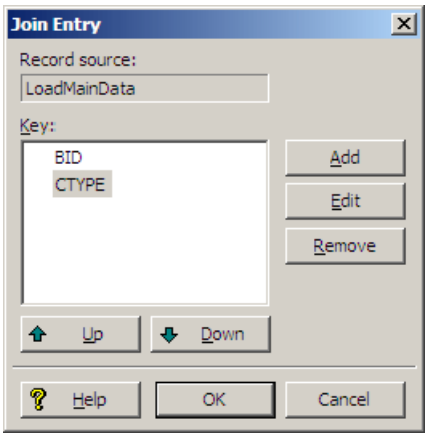
Related Links

Creating a record index on page 75
You specify a record index for a data source in the source's editor. The following example describes how to create a record index for a record cache.

Join keys with multiple properties or dimensions

You can specify multiple properties or dimensions, called *key components*, for a single join key in order to join records based on more than one characteristic.

For example, consider the task of joining book data to corresponding price data. Assume that the primary key component for a book is `BID` and price is determined by this `BID` plus another characteristic, the cover type `CTYPE`. Therefore, the join must be configured to join on both `BID` and `CTYPE`, as shown below:



For consistency in the comparison, the join key for each source participating in a join must be parallel. In other words, they must have the same number of key components, in the same order. Also, the type of each join key component must be parallel for all join entries in a given record assembler. This means that a dimension value key component cannot be compared to a property name key component.

Chapter 12

About Implementing Joins

With two exceptions, all data sources feeding a join must be record caches, so the procedures in this section are written from that perspective.

Implementing a join

In order to implement a join, you must add the join and the records it will process into your pipeline, and configure the join accordingly.

Implementing a join is a three-step process:

1. Add a record cache to your pipeline for each record source that will feed the join.
2. Add a record assembler to your pipeline.
3. Configure the join in the record assembler.

Each step is described in the following sections.

Adding a record cache

Use the options in the **Record Cache editor** to add and configure a record cache for each of your record sources.

To add a record cache for each record source that will feed the join:

1. In the **Pipeline Diagram editor**, click **New**, and then choose **Record > Cache**.
   The **Record Cache editor** appears.
2. In the **Name** text box, type a unique name for this record cache.
3. (Optional) In the **General** tab, you may do the following:
   a) If the cache should load fewer than the total number of records from the record source, type the number of records to load in the **Maximum Records** text box. This features is provided for testing purposes.
   b) If you want to merge records with equivalent record index key values into a single record, check the **Combine Records** option. For one-to-many or many-to-many joins, leave **Combine Records** unchecked.

   Important: The **Combine Records** option can have unexpected results if you do not understand how it functions.

4. In the **Sources** tab, select a record source and, optionally, a dimension source.

If a component's record index contains dimension values, you must provide a dimension source. Generally, this is only the case if you are caching data that has been previously processed by Forge.

5. In the **Record Index** tab, do the following:
   a) Specify which properties or dimensions you want to use as the record index for this component. Note that the record index you specify for a cache must match the join key that you will specify for that cache in the record assembler.
   b) Indicate whether you want to discard records with duplicate keys.
6. (Optional) In the **Comment** tab, add a comment for the component.
7. Click **OK**.
8. Repeat these steps for all record sources that will be part of the join.

Related Links

Joins that do not require record caches on page 87
There are two join cases that do not require record caches:

Combining equivalent records in record caches on page 88
The **General** tab on the **Record Cache editor** has a **Combine Records** setting. With the setting enabled for record caches, equivalent records in data sources are combined.

Adding a record assembler

Use the **Record Assembler editor** to add and configure a new record assembler for your pipeline.

To add a record assembler to your pipeline:

1. In the **Pipeline Diagram editor**, click **New**, and then choose **Record > Assembler**.
   The **Record Assembler editor** appears.
2. In the **Name** text box, type a unique name for the new record assembler.
3. In the **Sources** tab, do the following:
   a) In the **Record Sources** list, select a record source and click **Add**. Repeat as necessary to add additional record sources.
      With two exceptions, record assemblers must use record caches as their source of record data.
   b) In the **Dimension Source** list, select a dimension source.
      If the key on which a join is performed contains dimension values, you must provide a dimension source. Generally, this is only the case if you are joining data that has already been processed once by Forge.
4. (Optional) In the **Record Index** tab, do the following:
   a) Specify which properties or dimensions you want to use as the record index for this component.
      An assembler's record index does not affect the join, it only affects the order in which downstream components will retrieve records from the assembler.
   b) Indicate whether you want to discard records with duplicate keys.
5. In the **Record Join** tab, configure your joins.
6. (Optional) In the **Comment** tab, add a comment for the component.
7. Click **OK**.

Related Links

Joins that do not require record caches on page 87
There are two join cases that do not require record caches:

Configuring the join on page 81
You can use the **Record Assembler** and **Join Type editors** to choose from and configure the different types of joins.

Configuring the join

You can use the **Record Assembler** and **Join Type editors** to choose from and configure the different types of joins.

To configure the join in the record assembler:

1. In the **Record Assembler editor**, click the **Record Join** tab.
2. Use the **Join Type** list to select the kind of join you want to perform.
3. If you are performing a left join, check the **Multi Sub-records** option if the left record can be joined to more than one right record.
4. The join entries list represents the record sources that will participate in the join, as specified on the **Sources** tab. In the **Join Entries** list, define the order of your join entries by selecting an entry and clicking **Up** or **Down**.

   For all joins, properties get processed from join sources in the order in they are in the list. The first entry is the Left entry for a left join.
5. To define the join key for a join entry, select the entry from the **Join Entries** list and click **Edit**.
   The **Join Entry editor** appears.
6. Click **Add**.
   The **Key Component editor** appears.
7. Using the steps below, create a join key that is identical to the record index key for the join entry you selected.
   a) In the **Type** frame, do one of the following:
      - Choose **Custom Property**. Type a name for the property in the **Custom Property** text box.
      - Choose **Dimension**. Select a dimension name from the **Dimension** list.

   b) Click **OK** to return to the **Join Entry editor**.
   c) (Optional) Repeat these steps for each component you want to add to the key.
   d) (Optional) To reorder the components in the key, select a component in the **Join Entry editor** and click **Up** or **Down**.
   e) Click **OK** to close the **Join Entry editor**.
8. Repeat steps 5 through 7 for each record source that is participating in the join.
9. When you are done configuring your join, click **OK** to close the **Record Assembler editor**.

Related Links

About tweaking left joins on page 85
The **Multi Sub-records** setting (on the **Record Assembler editor Record Join** tab) changes the behavior of a left join if a record from the left source has multiple values for the join key. It is used only used with left joins. Enabling this option forces Forge to create multiple keys for such records.

Join keys with multiple properties or dimensions on page 77
You can specify multiple properties or dimensions, called *key components*, for a single join key in order to join records based on more than one characteristic.

Join types on page 68
The following sections describe the join types supported by the Data Foundry. Each section provides a simple example for the join type being discussed. Note that while most of the examples use two

record sources, many of the join types accept more than two sources, while other join types accept only one. Also note that in the examples, `Id` is the name of the join key for all sources.

Chapter 13

Advanced Join Behavior

In some cases, multiple sets of records may use identical join keys, or a single record may include multiple keys (such as a database table with two `Id` columns). These sections cover how joins are handled for such situations.

Records that have multiple values for a join key

A record can have multiple property values for a given property name. For example, a record could have two values for the property `Id`.

If a record is configured to join to another record based on a key that has multiple values in one or both of the records, the join implementation must consider the multiple values in the comparison.

The question is, if the record has the values {A, B} for the property `Id`, should it match to records with value A, value B, or both? The answer is that the record matches to records that have exactly both values. This behavior is different than the semantics of a database join, because tuples in a database have only one value per column. Therefore, you should carefully consider how to handle records that have multiple values per key component.

Note: This section describes how to deal with records that have multiple values per join key. Do not confuse this scenario with one where your join keys incorporate multiple properties/dimensions.

The following example illustrates the effects of joining records that have multiple values for a join key.

Source 1

Id	Id	Name	Color
A	BB	Shirt	Blue
A	CC	Shirt	Red
DD	A	Pants	Black
B		Pants	Tan
B	CC	Sweater	Yellow

Source 2

Id	Id	Retail	Price
A	AA	$30	$20
A	BB	$25	$20
A	CC	$25	$25
A	DD	$25	$25
B	CC	$40	$35

A left join, using Id as the join key, on these two data sources results in the following:

Results of Left Join

Id	Id	Name	Color	Retail	Price
A	BB	Shirt	Blue	$25	$20
A	CC	Shirt	Red	$25	$25
DD	A	Pants	Black	$25	$25
B		Pants	Tan		
B	CC	Sweater	Yellow	$40	$35

The record from Source 1 with join key (Id=A, Id=BB) is combined with a record with the same key from Source 2. Similarly, since both sources have a record with keys (Id=A, Id=CC) and (Id=B, Id=CC), these records are combined appropriately. Finally, the record (Id=DD, Id=A) from Source 1 is combined with the record (Id=A, Id=DD) from Source 2. The order of the property values is not significant.

You can tweak left joins in which the left source has multiple values for a key by telling Forge to create a separate join key based on each value.

Related Links

Join keys with multiple properties or dimensions on page 77
You can specify multiple properties or dimensions, called *key components*, for a single join key in order to join records based on more than one characteristic.

About tweaking left joins on page 85
The **Multi Sub-records** setting (on the **Record Assembler editor Record Join** tab) changes the behavior of a left join if a record from the left source has multiple values for the join key. It is used only used with left joins. Enabling this option forces Forge to create multiple keys for such records.

Sources that have multiple records with the same join key value

This section explains Forge's behavior when joining sources where each source may have more than one record with the same join key value (*higher cardinality joins*).

For example, a record source might process 5 records each with Id=A. This behavior has a database counterpart. It is considered here because the results of the join can be complicated. The result of the join is a Cartesian product of the sets of records, from each source, with the same join key.

Consider performing a left join on the following two data sources, assuming the join key is the property Id. Both sources have records with redundant keys. For example, Source 1 has three records with Id=A and two records with Id=B. Source 2 has three records with Id=A and two records with Id=B.

Source 1

Id	Name	Color
A	Shirt	Blue
A	Shirt	Red
B	Pants	Tan
B	Sweater	Yellow
C	Pants	Black

Source 2

Id	Retail	Price
A	$30	$20
A	$25	$20
A	$25	$25
B	$40	$35
B	$25	$25

The results of a left join on these two data sources look like this:

Results of Left Join

Id	Name	Color	Retail	Price
A	Shirt	Blue	$30	$20
A	Shirt	Blue	$25	$20
A	Shirt	Blue	$25	$25
A	Shirt	Red	$30	$20
A	Shirt	Red	$25	$20
A	Shirt	Red	$25	$25
B	Pants	Tan	$40	$35
B	Pants	Tan	$25	$25
B	Sweater	Yellow	$40	$35
B	Sweater	Yellow	$25	$25
C	Pants	Black		

As discussed above, the join produces a Cartesian product. The first record from Source 1 (Id=A, Name=Shirt, Color=Blue) is combined with each of the three records from Source 2 that have the join key Id=A, producing the first three records shown in the results table. Similarly, the second record from Source 1 (Id=A, Name=shirt, Color=blue) is combined with each of the three records from Source 2 with the join key Id=A to produce the next three records.

For a given join key Id=x, the number of records created by a Cartesian product is the product of the number of records in each source with Id=x. In the example above, Source 1 had two records with Id=A and Source 2 had three. Therefore, the Cartesian product produces six records (2 x 3 = 6). Adding a third source with three records of Id=A would produce 18 records (2 x 3 x 3 = 18). Because the number of records produced can grow quickly, you should take care should to evaluate correctness when dealing with data of this nature. Often, the desired behavior is to combine records with duplicate keys, using a Combine join or the **Combine Records** option on a record cache, from all or several sources.

About tweaking left joins

The **Multi Sub-records** setting (on the **Record Assembler editor Record Join** tab) changes the behavior of a left join if a record from the left source has multiple values for the join key. It is used only used with left joins. Enabling this option forces Forge to create multiple keys for such records.

Note: In the case where a left source's join key consists of a single property/dimension, each value becomes an independent key.

For example, if the join key is Id, a record with the values Id=1, Id=2, Id=3 produces three independent keys, one for each value. The right sources are searched for each of these keys. That is, each right source is queried for a match to the join key Id=1, a match to Id=2, and finally a match to Id=3. All records that match any of the keys are combined with the record from the left source, producing the joined record.

Multi sub-records can be extrapolated to join keys with multiple key components by considering the values corresponding to each key component as a set. Performing a Cartesian product of these sets provides the key combinations. For example, given the key components idA and idB and a record from the left source with the values idA=1, idA=2, idB=11, idB=12, the keys produced by the Cartesian product are [{idA=1, idB=11}, {idA=1, idB=12}, {idA=2, idB=11}, {idA=2, idB=12}]. Again, the right sources are searched for each of these keys.

Multi sub-records

A good example that illustrates the use of multi sub-records is one where you have a left table that consists of a CD and the songs on it, and a right table with song details.

In this example, you would perform the join on the SongId, so that each song in the left table is joined appropriately with its counterpart in the right table. Note that in this example, `SongId` is the join key for all sources.

CD Source (Left)

CDId	CDTitle	**SongId**	**SongId**	**SongId**
1	Abbey Road	**1s**	**2s**	**3s**

Song Source

SongId	SongTitle
1s	Come Together
2s	Sun King
3s	Octopus's Garden

Results of Left Join

CDId	CDTitle	**SongId**	**SongId**	**SongId**	SongTitle	SongTitle	SongTitle
1	Abbey Road	**1s**	**2s**	**3s**	Come Together	Sun King	Octopus's Garden

Related Links

Join keys with multiple properties or dimensions on page 77

You can specify multiple properties or dimensions, called *key components*, for a single join key in order to join records based on more than one characteristic.

Chapter 14

Tips and Troubleshooting for Joins

The sections below provide tips and troubleshooting information for joins.

Joins that do not require record caches

There are two join cases that do not require record caches:

- Switch joins do not do record comparisons and, hence, do not require record caches for their data sources. You can use any type of record server component (record adapter, record cache, record assembler, Perl manipulator, and so on) as a source for a switch join.
- For a left join, for which all of the right sources are record caches, the left source does not require a record cache. This special case is useful for optimizing a left join with a large, unsorted data source.

Working with sources that have multiple records with the same join key value

In order to configure a join with the desired behavior, it is important to have a strong understanding of what happens when record assemblers process records that do not have unique values for their join keys (higher cardinality joins).

Related Links

Sources that have multiple records with the same join key value on page 84
This section explains Forge's behavior when joining sources where each source may have more than one record with the same join key value (*higher cardinality joins*).

Best practice for choosing left and right side of joins

A best practice is to keep record sources with the most values per join key on the left side of joins.

When performing joins (such as an outer join), Forge can output records from both sides of the join, except where two records, one from each side, match on the join key, in which case it combines the two records into one. The interesting case is when multiple records on each side have the same value for the join key. For example, if 10 records from the left side and 10 records from the right side each have the same value for the join key, the result of the join is the cross-product of all the records, 100 in total.

Thus, when Forge does joins, it typically streams records from each side, joining where appropriate and outputting records, joining them where appropriate. But in the cross-product case, it cannot stream records from both sides simultaneously. For each record on one side, Forge has to do a separate iteration of the records on the other side. Forge has to pick at least one side of the join for loading all the records with the same join key into memory. Forge's design chooses the right side for that; it always streams records from the left side. On the right side, however, while Forge streams whenever possible, it will load all records with a common join key value into memory.

Thus, a best practice is to keep record sources with the most values per join key on the left side of joins.

Combining equivalent records in record caches

The **General** tab on the **Record Cache editor** has a **Combine Records** setting. With the setting enabled for record caches, equivalent records in data sources are combined.

The setting controls how the cache handles records that have equivalent values for the record index key, and it is turned off by default. Care should be taken if you choose to use it.

Consider performing a left join on the following two data sources, assuming the record index key is the property `Id`. Both sources have records with redundant keys. For example, Source 1 has three records with Id=A and two records with Id=B. Source 2 has three records with Id=A and two records with Id=B.

Source 1

Id	Name	Color
A	Shirt	Blue
A	Shirt	Red
B	Pants	Tan
B	Sweater	Yellow
C	Pants	Black

Source 2

Id	Retail	Price
A	$30	$20
A	$25	$20
A	$25	$25
B	$40	$35
B	$25	$25

Without the **Combine Records** setting enabled, the results of a left join on these two data sources look like this:

Results of left join without Combine records enabled

Id	Name	Color	Retail	Price
A	Shirt	Blue	$30	$20
A	Shirt	Blue	$25	$20
A	Shirt	Blue	$25	$25
A	Shirt	Red	$30	$20
A	Shirt	Red	$25	$20
A	Shirt	Red	$25	$25
B	Pants	Tan	$40	$35
B	Pants	Tan	$25	$25
B	Sweater	Yellow	$40	$35
B	Sweater	Yellow	$25	$25
C	Pants	Black		

With the **Combine Records** setting enabled for the record caches, equivalent records in the data sources would be combined, so the new data sources would look like this:

Source 1, combined

Id	Name	Name	Color	Color
A	Shirt		Blue	Red
B	Pants	Sweater	Tan	Yellow
C	Pants			Black

Source 2, combined

Id	Retail	Retail	Price	Price
A	$30	$25	$20	$25
B	$40	$25	$35	$25

The results of a left join on these two combined data sources would look like this:

Results of left join with "Combine records" enabled

Id	Name	Name	Color	Color	Retail	Retail	Price	Price
A	Shirt		Blue	Red	$30	$25	$20	$25
B	Pants	Sweater	Tan	Yellow	$40	$25	$35	$25
C	Pants			Black				

Forge warnings when combining large numbers of records

When combining a large number of records (via either a Combine join or a record cache with the **Combine Records** setting enabled), Forge will issue a warning that performance may be slow. The default number of records at which this warning is issued is 100.

This threshold can be adjusted with the Forge `--combineWarnCount` command-line flag.

Two messages will be printed:

- The first is an informational message that is printed when the number of records combined reaches the `--combineWarnCount` threshold. The message includes the key of the records being combined. The

intent of this message is to give users an early warning that Forge has just started a potentially long operation and therefore may seem to be stalled, but is actually working.

- The second message is a warning, indicating the total number of records combined, and the value of the key.

Note: Setting the `--combineWarnCount` value to 0 (zero) will disable these messages.

Part 3

Advanced Dimension Features

- *Externally-Created Dimensions*
- *Externally-Managed Taxonomies*

Chapter 15

Externally-Created Dimensions

This section describes how to include and work with an externally-created dimension in a Developer Studio project. This capability allows you to build all or part of a logical hierarchy for your data set outside of Developer Studio and then import that logical hierarchy as an Endeca dimension available for use in search and Guided Navigation.

Overview of externally-created dimensions

An externally-created dimension describes a logical hierarchy of a data set; however, the dimension hierarchy is transformed from its source format to Endeca compatible XML outside of Developer Studio.

The logical hierarchy of an externally-created dimension must conform to Endeca's external interface for describing a data hierarchy (found in `external_dimensions.dtd`) before you import the dimension into your project. Once you import an externally-created dimension, its ownership is wholly transferred to Developer Studio, so that afterwards you can modify the dimension with Developer Studio.

Related Links

External dimensions and external taxonomies on page 93
Externally-managed taxonomies and externally-created dimensions differ in how you include them in a Developer Studio project and how Developer Studio treats them once they are part of a project.

External dimensions and external taxonomies

Externally-managed taxonomies and externally-created dimensions differ in how you include them in a Developer Studio project and how Developer Studio treats them once they are part of a project.

It is important to clarify the difference between an externally-managed taxonomy and an externally-created dimension to determine which feature document is appropriate for your purposes. Use the table below to determine which one you are working with.

The following table compares an externally-managed taxonomy and an externally-created dimension:

Operation	Externally-managed taxonomy	Externally-created dimension
How do you modify or update the	Any changes to the dimension must be made in third-party tool. You then export the taxonomy	You generally do not update the source file for the hierarchy after you import it into your project. If you do update the file and re-import, then any changes you

Operation	Externally-managed taxonomy	Externally-created dimension
hierarchy after it is in the project?	from the tool, and Forge transforms the taxonomy and re-integrates the changes into your project.	made to the dimension using Developer Studio are discarded. After importing the hierarchy, you can modify a dimension just as if you created it manually using Developer Studio.
How does Developer Studio manage the hierarchy?	The third-party tool that created the file retains ownership. The dimension is almost entirely read-only in the project. You cannot add or remove dimension values from the dimension. However, you can modify whether dimension values are inert and collapsible.	After you import the file, Developer Studio takes full ownership of the dimension and its dimension values. You can modify any characteristics of the dimension and its dimension values.
How do you create the XML file?	Created using a third-party tool.	Created either directly in an XML file or created using a third-party tool.
How do you include the file in a Developer Studio project?	Read in to a pipeline using a dimension adapter with **Format** set to **XML - Externally Managed**. Forge transforms the taxonomy file in to a dimension according to the `.xslt` file that you specify on the Transformer tab of the dimension adapter.	By choosing **Import External Dimension** on the **File** menu. During import, Developer Studio creates internal dimensions and dimension values for each node in the file's hierarchy. If you create the file using a third-party tool and any XML transformation is necessary, you must transform the file outside the project before you choose **Import External Dimension** on the **File** menu. The file must conform to `external_dimensions.dtd`.

Related Links

Overview of externally-managed taxonomies on page 99

An externally-managed taxonomy is a logical hierarchy for a data set that is built and managed using a third-party tool. Once you include an externally-managed taxonomy in your project, it becomes a dimension whose hierarchy is managed by the third-party tool that created it.

Including externally-created dimensions in your project

You can use Developer Studio to include an externally-created dimension file in your project, as long as the dimension file conforms to the `external_dimensions.dtd` file.

Ensure you are working with an externally-created dimension, and not an externally-managed taxonomy. Any created dimension files must conform to the Endeca `external_dimensions.dtd` file.

An overview of the process to include an externally-created dimension in a Developer Studio project is as follows:

1. Create a dimension hierarchy. You can do this one of two ways:
   - Create it manually in an XML file.
   - Create a dimension using a third-party tool.

2. Import the XML file for the dimension into Developer Studio, and modify the dimension and dimension values as necessary.

Related Links

External dimensions and external taxonomies on page 93
Externally-managed taxonomies and externally-created dimensions differ in how you include them in a Developer Studio project and how Developer Studio treats them once they are part of a project.

XML requirements on page 95
When you create an external dimension—whether by creating it directly in an XML file or by transforming it from a source file—the dimension must conform to Endeca's `external_dimensions.dtd` file before you import it into your project.

Importing an externally-created dimension on page 97
You add an externally-created dimension to your pipeline by importing it with Developer Studio.

XML requirements

When you create an external dimension—whether by creating it directly in an XML file or by transforming it from a source file—the dimension must conform to Endeca's `external_dimensions.dtd` file before you import it into your project.

The external_dimensions.dtd file defines Endeca-compatible XML used to describe dimension hierarchies in an Endeca system. This file is located in `%ENDECA_ROOT%\conf\dtd` on Windows and `$ENDECA_ROOT/conf/dtd` on UNIX.

Also, an XML declaration that specifies the `external_dimensions.dtd` file is required in an external dimensions file. If you omit specifying the DTD in the XML declaration, none of the DTD's implied values or other default values, such as classification values, are applied to the external dimensions during Endeca ITL processing. Here is an example XML declaration that should appear at the beginning of an external dimension file:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE external_dimensions SYSTEM "external_dimensions.dtd">
```

Here is a very simple example of an external dimension file with the required XML declaration and two dimensions:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE external_dimensions SYSTEM "external_dimensions.dtd">

<external_dimensions>
 <node id="1" name="color" classify="true">
  <node id="2" name="red" classify="true"/>
  <node id="3" name="blue" classify="true"/>
 </node>

 <node id="10" name="size" classify="true">
  <node id="20" name="small" classify="true"/>
  <node id="30" name="med" classify="true"/>
 </node>

</external_dimensions>
```

Related Links

XML syntax to specify dimension hierarchy on page 96

The XML elements available to `external_dimensions.dtd` allow a flexible XML syntax to describe a dimension hierarchy. There are three different syntax approaches you can choose from when building the hierarchy structure of your externally-created dimension.

XML syntax to specify dimension hierarchy

The XML elements available to `external_dimensions.dtd` allow a flexible XML syntax to describe a dimension hierarchy. There are three different syntax approaches you can choose from when building the hierarchy structure of your externally-created dimension.

All three approaches are supported by `external_dimensions.dtd`. Each provides a slightly different syntax structure to define a dimension and express the parent/child relationship among dimensions and dimension values. The three syntax choices are as follows:

- Use nested node elements within node elements.
- Use the parent attribute of a node to reference a parent's node ID.
- Use the child element to reference the child's node ID.

You can use only one of the three approaches to describe a hierarchy within a single XML file. In other words, do not mix different syntax structures within one file. Any node element without a parent node describes a new dimension. You can describe as many dimensions as necessary in a single XML file.

The following examples show each approach to building a dimension hierarchy. The these examples are semantically equivalent: each describes the same dimension and child dimension values.

Example of using nested node elements

This example shows nested dimension values `red` and `blue` within the dimension `color`:

```
<node name="color" id="1">
 <node name="red" id="2"/>
 <node name="blue" id="3"/>
</node>
```

Example of using parent attributes

This example shows the `red` and `blue` dimension values using the parent attribute. The value of the parent attribute references the ID for the dimension `color`:

```
<node name="color" id="1"/>
<node id="2" name="red" parent="1"/>
<node id="3" name="blue" parent="1"/>
```

Example of using child elements

This example uses child elements to indicate that `red` and `blue` are dimension values of the `color` dimension. The ID of each child element references the ID of the `red` and `blue` nodes:

```
<node name="color" id="1">
 <child id="2"/>
 <child id="3"/>
</node>
<node name="red" id="2"/>
<node name="blue" id="3"/>
```

Node ID requirements

Each `node` element in your dimension hierarchy must have an `id` attribute. Depending on your requirements, you may choose to provide any of the following values for the id attribute:

- **Name** — If the name of a dimension value is what determines its identity, then provide the `id` attribute with the name.
- **Path** — If the path from the root node to the dimension value determines its identity, then provide a value representing the path in the `id` attribute.
- **Existing identifier** — If a node already has an identifier, then that identifier can be used in the `id` attribute.

The id value must be unique. If you are including multiple XML files, the identifier must be unique across the files.

There is one scenario where an id attribute is optional. It is optional only if you are using an externally-created dimension and also defining your dimension hierarchy using nested node sub-elements (rather than using parent or child ID referencing).

Importing an externally-created dimension

You add an externally-created dimension to your pipeline by importing it with Developer Studio.

Once you import the XML file, the dimension appears in the **Dimensions** view, and Developer Studio has full read-write ownership of the dimension. You can modify any aspects of a dimension and its dimension values as if you created it in Developer Studio.

To import an externally-created dimension:

Note: Unlike the procedure to import an externally-managed taxonomy, you do not need to run a baseline update to import an externally-created dimension.

1. Select **File > Import External Dimensions**.
   The **Import External Dimensions** dialog box displays.
2. Specify the XML file that defines the dimensions.
3. Chose a dimension adapter from the **Dimension adapter to receive imported dimensions** drop-down list.
4. Click **OK**.
   The dimensions appear in the **Dimensions editor** for you to configure as necessary.
5. Save the project.

Related Links

Including externally-managed taxonomies in your project on page 99

You can use Developer Studio to include an externally-managed taxonomy into your project, but you cannot alter the taxonomy within Developer Studio, even after importing it.

Chapter 16

Externally-Managed Taxonomies

This section describes how to work with an externally-managed taxonomy in a Developer Studio project. This capability allows you to build all or part of a logical hierarchy for your data set outside of Developer Studio and use Developer Studio to transform that logical hierarchy into Endeca dimensions and dimension values for use in search and Guided Navigation.

Overview of externally-managed taxonomies

An externally-managed taxonomy is a logical hierarchy for a data set that is built and managed using a third-party tool. Once you include an externally-managed taxonomy in your project, it becomes a dimension whose hierarchy is managed by the third-party tool that created it.

In Developer Studio, you cannot add or remove dimension values from an externally-managed taxonomy. If you want to modify a dimension or its dimension values, you have to edit the taxonomy using the third-party tool and then update the taxonomy in your project.

It is important to clarify the difference between an externally-managed taxonomy and an externally-created dimension to determine which feature document is appropriate for your purposes. The two concepts are similar yet have two important key differences: externally-managed taxonomies and externally-created dimensions differ in how you include them in a Developer Studio project and how Developer Studio treats them once they are part of a project.

Related Links

External dimensions and external taxonomies on page 93
Externally-managed taxonomies and externally-created dimensions differ in how you include them in a Developer Studio project and how Developer Studio treats them once they are part of a project.

Including externally-managed taxonomies in your project

You can use Developer Studio to include an externally-managed taxonomy into your project, but you cannot alter the taxonomy within Developer Studio, even after importing it.

Ensure you are working with an externally-managed taxonomy, and not an externally-created dimension.

An overview of the process to include an externally-managed taxonomy in a Developer Studio project is as follows:

1. You build an externally-managed taxonomy using a third-party tool. This guide does not describe any third-party tools or procedures that you might use to perform this task.
2. You create an XSLT style sheet that instructs Forge how to transform the taxonomy into Endeca XML that conforms to `external_dimensions.dtd`.

   Important: When creating your XSLT stylesheet, you should use the Xerces XML parser to ensure that the output is compatible with Forge.

3. You configure your Developer Studio pipeline to perform the following tasks:
   a) Describe the location of an externally-managed taxonomy and an XSLT style sheet with a dimension adapter.
   b) Transform an externally-managed taxonomy into an externally-managed dimension by running a baseline update.
   c) Add an externally-managed dimension to the **Dimensions** view and the **Dimension Values** view.

After you finish the tasks listed above, you can perform additional pipeline configuration that uses the externally-managed dimension, and then run a second baseline update to process and tag your Endeca records.

Related Links

External dimensions and external taxonomies on page 93
Externally-managed taxonomies and externally-created dimensions differ in how you include them in a Developer Studio project and how Developer Studio treats them once they are part of a project.

XSLT and XML requirements on page 100
To transform an externally-managed taxonomy into an externally-managed dimension, you have to create an XSLT style sheet that instructs Forge how to map the taxonomy XML to Endeca XML. The mapping in your XSLT style sheet and your resulting hierarchy must conform to the Endeca `external_dimensions.dtd` file.

XSLT and XML requirements

To transform an externally-managed taxonomy into an externally-managed dimension, you have to create an XSLT style sheet that instructs Forge how to map the taxonomy XML to Endeca XML. The mapping in your XSLT style sheet and your resulting hierarchy must conform to the Endeca `external_dimensions.dtd` file.

About XSLT mapping

In order for Developer Studio to process the XML from your externally-managed taxonomy, you have to create an XSLT style sheet that instructs Forge how to map the XML elements in an externally-managed taxonomy to Endeca-compatible XML.

This requires configuring the **Transformer** tab of a dimension adapter with the path to the XSLT style sheet and the path to the taxonomy XML file, and then running a baseline update to transform the external taxonomy into an Endeca dimension.

Important: When creating your XSLT stylesheet, you should use the Xerces XML parser to ensure that the output is compatible with Forge.

The `external_dimensions.dtd` defines Endeca-compatible XML to describe dimension hierarchies. This file is located in `%ENDECA_ROOT%\conf\dtd` on Windows and `$ENDECA_ROOT/conf/dtd` on UNIX.

XML syntax to specify dimension hierarchy

The XML elements available to `external_dimensions.dtd` allow a flexible XML syntax to describe dimension hierarchy. There are three different syntax approaches you can choose from when building the hierarchy structure of your externally-managed dimension.

All three options are supported by `external_dimensions.dtd`. Each approach provides a slightly different syntax structure to define a dimension and express the parent/child relationship among dimensions and dimension values. The three syntax choices are as follows:

- Use nested node elements within node elements.
- Use the parent attribute of a node to reference a parent's node ID.
- Use the child element to reference the child's node ID.

You can use only one of the three approaches to describe a hierarchy within a single XML file. In other words, do not mix different syntax structures within one file. Any node element without a parent node describes a new dimension. You can describe as many dimensions as necessary in a single XML file.

The following examples show each approach to building a dimension hierarchy. These examples are semantically equivalent: each describes the same dimension and child dimension values.

Example of using nested node elements

This example shows nested dimension values `red` and `blue` within the dimension `color`:

```
<node name="color" id="1">
 <node name="red" id="2"/>
 <node name="blue" id="3"/>
</node>
```

Example of using parent attributes

This example shows the `red` and `blue` dimension values using the parent attribute. The value of the parent attribute references the ID for the dimension `color`.

```
<node name="color" id="1"/>
<node name="red" id="2" parent="1"/>
<node name="blue" id="3" parent="1"/>
```

Example of using child elements

This example uses child elements to indicate that `red` and `blue` are dimension values of the `color` dimension. The ID of each child element references the ID of the `red` and `blue` nodes.

```
<node name="color" id="1">
 <child id="2"/>
 <child id="3"/>
</node>
<node name="red" id="2"/>
<node name="blue" id="3"/>
```

Node ID requirements and identifier management in Forge

When you transform the hierarchy structure from an external taxonomy, each node element in your dimension hierarchy must have an id attribute. Forge ensures that each identifier is unique across an Endeca implementation by creating a mapping between a node's ID and an internal identifier that Forge creates.

This internal mapping ensures that Forge assigns the same identifier to a node from an external taxonomy each time the taxonomy is processed. For example, if you provide updated versions of a taxonomy file, Forge determines which dimension values map to dimension values from a previous version of the file according to the internal identifier. However, there is a scenario where Forge does not preserve the mapping between the id attribute and the internal identifier that Forge creates for the dimension value. This occurs if you reorganize a dimension value to become a child of a different parent dimension. Reorganizing a dimension value within the same parent dimension does not affect the id mapping when Forge reprocesses updated files.

Depending on your requirements, you may choose to provide any of the following values for the id attribute:

- **Name** — If the name of a dimension value is what determines its identity, then the XSLT style sheet should fill the id attribute with the name.
- **Path** — If the path from the root node to the dimension value determines its identity, then the XSLT style sheet should put a value representing the path in the id attribute.
- **Existing identifier** — If a node already has an identifier, then that identifier can be used in the id attribute.

You can provide an arbitrary ID as long as the value is unique. If you are including multiple XML files, the identifier must be unique across all files. As described above, Forge ensures that identifiers are unique across the system.

Pipeline configuration

These sections describe the pipeline configuration requirements to incorporate an externally-managed taxonomy into your Developer Studio project.

Integrating an externally-managed taxonomy

You use a dimension adapter to read in XML from an externally-managed taxonomy and transform it to an externally-managed Endeca dimension. If necessary, you can import and transform multiple taxonomies by using a different dimension adapter for each taxonomy file.

To perform the taxonomy transformation, you configure a dimension adapter with the XML file of the taxonomy and the XSLT style sheet that Forge uses to transform the taxonomy file's XML elements. You then build the rest of your pipeline, set the instance configuration, and run a baseline update. When the update runs, Forge transforms the taxonomy into a dimension that you can load and examine in the **Dimensions** view.

To integrate an externally-managed taxonomy:

1. In the **Project** tab of Developer Studio, double-click **Pipeline Diagram**.
2. In the **Pipeline Diagram editor**, choose **New > Dimension > Adapter**.
   The **Dimension Adapter editor** displays.
3. In the **Dimension Adapter Name** text box, enter a unique name for the dimension adapter.
4. In the **General** tab, do the following:
   a) In the **Direction** frame, select **Input**.
   b) In the **Format** field, select **XML - Externally Managed**.

c) In the **URL** field, enter the path to the source taxonomy file. This path can be absolute or relative to the location of your project's `Pipeline.epx` file.
d) Check **Require Data** if you want Forge to generate an error if the file does not exist or is empty.

5. In the **Transformer** tab, do the following:
   a) In the **Type** field, enter `XSLT`.
   b) In the **URL** field, specify the path to the XSLT file you created.
6. Click **OK**.
7. Select **File > Save**
8. If necessary, repeat steps 2 through 6 to include additional taxonomies.
9. Create a dimension server to provide a single point of reference for other pipeline components to access dimension information. For more information about dimension servers, see the *Oracle Endeca Developer Studio Help*.

Transforming an externally managed taxonomy

In order to transform your externally-managed taxonomy into an Endeca dimension, you have to set the instance configuration and run a baseline update.

Running the update allows Forge to transform the taxonomy and store a temporary copy of the resulting dimension in the EAC Central Server. After you run the update, you can then create a dimension in the **Dimensions** view.

To transform an externally-managed taxonomy:

Note: To reduce processing time for large source data sets, you may want to run the baseline update using the `-n` flag for Forge. (The `-n` flag controls the number of records processed in a pipeline, for example, `-n 10` processes ten records.) You can specify the flag in the Forge section of the EAC Admin Console page of Endeca Workbench.

1. Add any pipeline components to your pipeline which are required for the update to run.

   You cannot, for example, run the update without a property mapper. However, you can temporarily add a default property mapper and later configure it with property and dimension mapping.
2. Ensure you have sent the latest instance configuration to Endeca Workbench.

   Oracle recommends using the `update_web_studio_config` script to perform this task, as it will not overwrite any configuration files which are maintained by Endeca Workbench. The `update_web_studio_config` script is included in the Endeca Deployment Template.

   The Oracle Endeca Deployment Template is available on the Oracle Software Delivery Cloud. For more information about the Endeca Deployment Template, see the *Oracle Commerce Administrator's Guide*
3. On the EAC Admin Console page of Endeca Workbench, run your baseline update script.

Related Links

Loading an externally-managed dimension on page 104

After you transform an external taxonomy into an Endeca dimension, you can then load the dimension in the **Dimensions** view and add its dimension values to the **Dimension Values** view.

Uploading post-Forge dimensions to Endeca Workbench

If you are using the baseline update script provided with the Deployment Template, it will automatically upload the post-Forge dimensions to Endeca Workbench. If, however, you are not using these scripts, you must use the `emgr_update` utility.

After the baseline update finishes, the latest dimension values generated by the Forge process must be uploaded to Endeca Workbench. This will ensure that any new dimension values (including values for autogen dimensions and external dimensions) are available to Endeca Workbench for use (for example, for merchandizing rule triggers).

If you are not using the baseline update script provided with the Deployment Template, you must use the `emgr_update` utility as follows:

1. Open a command prompt or UNIX shell to run the utility.
2. Enter the following into the command line:`emgr_update --action set_post_forge_dims`
   This will update the Endeca Workbench configuration with the post-Forge dimensions.

For more information on this utility, see the *Oracle Commerce Administrator's Guide*.

Loading an externally-managed dimension

After you transform an external taxonomy into an Endeca dimension, you can then load the dimension in the **Dimensions** view and add its dimension values to the **Dimension Values** view.

Rather than click **New**, as you would to manually create a dimension in Developer Studio, you instead click **Discover** in **Dimensions** view to add an externally-managed dimension. Developer Studio discovers the dimension by reading in the dimension's file that Forge created when you ran the first baseline update. Next, you load the dimension values in the **Dimension Values editor**.

To load a dimension and its dimension values:

Note: Because the dimension values are externally managed, you cannot add or remove dimension values. You can however modify whether dimension values are inert or collapsible.

1. In the **Project** tab of Developer Studio, double-click **Dimensions**.
   The **Dimensions** view displays.
2. Click the **Discover** button to add the externally-managed dimension to the **Dimensions** view.

   Most characteristics of an externally-managed dimension and its dimension values are not modifiable. These characteristics either appear as unavailable or Developer Studio displays a message indicating what actions are possible.

   The dimension appears in the **Dimensions** view with its **Type** column set to **Externally Managed**.
3. In **Dimensions** view, select the externally-managed dimension and click **Values**.
   The **Dimension Values** view appears with the root dimension value of the externally-managed dimension displayed.
4. Select the root dimension value and click **Load**.
   The remaining dimension values display.
5. Repeat steps 3 and 4 for any additional externally-managed taxonomies you integrated in your project.

Related Links

About updating an externally-managed taxonomy in your pipeline on page 105

If you want to modify an externally-managed taxonomy and replace it with a newer version, you have to revise the taxonomy using the third-party tool that created it, and then repeat the process of incorporating the externally-managed taxonomy into your pipeline.

Running a second baseline update

After loading dimension values and building the rest of your pipeline, you must run a second baseline update to process and tag your Endeca records. The second baseline update performs property and dimension mapping that could not be performed in the first baseline update because the externally-managed dimensions had not yet been transformed and available for mapping.

Before running this update, make sure you have transformed and mapped your externally-managed dimensions.

To run a second baseline update:

1. Ensure you have sent the latest instance configuration to Endeca Workbench.

   Oracle recommends using the `update_web_studio_config` script to perform this task, as it will not overwrite any configuration files which are maintained by Endeca Workbench. The `update_web_studio_config` script is included in the Endeca Deployment Template.

   The Oracle Endeca Deployment Template is available on the Oracle Software Delivery Cloud. For more information about the Endeca Deployment Template, see the *Oracle Commerce Administrator's Guide*.

2. On the EAC Admin Console page of Endeca Workbench, run your baseline update script.

About updating an externally-managed taxonomy in your pipeline

If you want to modify an externally-managed taxonomy and replace it with a newer version, you have to revise the taxonomy using the third-party tool that created it, and then repeat the process of incorporating the externally-managed taxonomy into your pipeline.

Related Links

Pipeline configuration on page 102

These sections describe the pipeline configuration requirements to incorporate an externally-managed taxonomy into your Developer Studio project.

Unexpected default-mapping behavior

Under certain circumstances, Forge will default-map dimensions from externally-managed taxonomies even when default dimension mapping is disabled in the Property Mapper. The following two conditions are required for this behavior to occur:

- A dimension mapping exists in the Property Mapper which points to a dimension sourced from an externally-managed taxonomy file.
- The source node for the mapped dimension is not present in the externally-managed taxonomy file (for example, because of a taxonomy change or user error).

When these two conditions are met, Forge dynamically creates an entry for the missing node's dimension in its output dimensions files; this entry has the attribute value `SRC_TYPE="PROPMAPPER"`, as seen on default-mapped dimension entries.

This behavior occurs even when the default dimension mapping functionality is disabled (that is, the **If no mapping is found, map source properties to Endeca dimensions** option on the **Advanced** tab of the **Property Mapper component editor** in Developer Studio is not checked).

The reason for the behavior is that in this case, Forge is handling the pipeline differently than Developer Studio. In Developer Studio, you cannot map a source property to a non-existent dimension. Forge, however, allows for properties to be mapped to undeclared dimensions, and when it encounters such a mapping, it creates a new dimension for it using the property mapper.

Part 4

Other Advanced Features

- *The Forge Logging System*
- *The Forge Metrics Web Service*

Chapter 17

The Forge Logging System

This section provides a brief introduction to the Forge logging system. Its command-line interface allows you to focus on the messages that interest you globally and by topic.

Overview of the Forge logging system

The Forge logging system provides a logging interface to Forge components. With this system, you can specify the logging level for a component globally or by topic.

The logging level allows you to filter logging messages so you can monitor elements of interest at the appropriate granularity without being overwhelmed by messages that are not relevant.

A simple command-line interface makes it easy to adjust your logging strategy to respond to your needs. During development, you might be interested in feedback on only the feature you are working on, while in production, you would typically focus on warnings and errors.

Log levels reference

The log levels used by Forge logging are as follows:

Log level	Description
FATAL	Indicates a problem so severe that you have to shut down.
ERROR	Non-fatal error messages.
WARN	Alerts you to any peculiarities the system notes. You may want to address these.
INFO	Provides status messages, even if everything is working correctly.
DEBUG	Provides all information of interest to a user.

About logging topics

All log messages are flagged with one or more topics. There are different types for different components, all logically related to some aspect of the component.

In Forge, you can specify individual logging levels for each of the following topics:

- baseline
- update
- config
- webservice
- metrics

The command line interface

You access logging on Forge with the `--logLevel` option. Its usage is as follows: `--logLevel (<topicName>=)<logLevel>`

By selecting a level you are requesting all feedback at of that level of severity and greater. For example, by specifying the `WARN` level, you receive `WARN`, `ERROR`, and `FATAL` messages.

The `--logLevel` option sets either the default log level, the topic log level, or both:

- The default log level provides global logging for the component:
  ```
  forge --logLevel WARN
  ```
  This example logs all `WARN` level or higher messages.

  Note: Forge defaults to log all `INFO` or higher level messages if a default level is not specified.

- The topic log level provides logging at the specified level for just the specified topic:
  ```
  forge --logLevel baseline=DEBUG
  ```
  This example overrides the default log level and logs all `DEBUG` messages and higher in the baseline topic.

- If two different log levels are specified, either globally or to the same topic, the finer-grained level is used:
  ```
  forge –logLevel INFO –logLevel WARN
  ```
  In the case of this example, all `INFO` level messages and higher are printed out.

It is possible to specify both default and topic level logging in the same command to filter the feedback that you receive. For example:

```
forge --logLevel WARN --logLevel config=INFO --logLevel update=DEBUG
```

This command works as follows:

- It logs all `WARN` or higher messages, regardless of topic.
- It logs any message flagged with the config topic if it is `INFO` level or higher.
- It logs any message flagged with the update topic if it is `DEBUG` level or higher.

Aliasing existing -v levels

The Forge `-v` logging option is still supported, but has been changed to alias the `--logLevel` option as follows: `-v[f|e|w|i|d]`. The following table maps the relationships and indicates the status of the arguments in this release (supported or deprecated).

Argument	Status in 11.0.0	Pre-6.x meaning	11.0.0 log level
`-v`	Supported	Defaults to v (verbose) or the EDF_LOG_LEVEL environment variable.	`DEBUG`
`-vv`	Deprecated	Verbose (all messages).	`DEBUG`
`-vi`	Supported	Info (info, stat, warnings, and errors).	`INFO`
`-va`	Deprecated	Stat (stat, warnings, and errors).	`INFO`
`-vw`	Supported	Warnings and errors.	`WARN`
`-ve`	Supported	Errors.	`ERROR`
`-vq`	Deprecated	Quiet mode (errors).	`ERROR`
`-vs`	Deprecated	Silent mode (fatal errors).	`FATAL`
`-vd`	Supported	n/a	`DEBUG`
`-vf`	Supported	n/a	`FATAL`
`-vt`	Deprecated	Printed out the timestamp when using `--legacyLogFormat`.	Has no effect. The timestamp is always printed now.

About logging output to a file

In Forge, the `-o` flag defines a location for the logging output file. If you do not specify a location, it logs to standard error.

The following snippet shows the start of an output file:

```
INFO 01/25/07 15:15:50.791 UTC FORGE {config}: forge <version> ("i86pc-win32")

INFO 01/25/07 15:15:50.791 UTC FORGE {config}: Copyright 2001-2007 Endeca Tech¬
nologies, Inc.

INFO 01/25/07 15:15:50.791 UTC FORGE {config}: Command Line: i86pc-
win32\bin\forge.exe

INFO 01/25/07 15:15:50.791 UTC FORGE {config}: Initialized cURL, version:
libcurl/7.15.5 OpenSSL/0.9.8

ERROR 01/25/07 15:15:50.791 UTC FORGE {config}: A file name is required!
```

Changes to the EDF_LOG_LEVEL environment variable

The `EDF_LOG_LEVEL` environment variable continues to be supported. If used, it should be set to one of the new log level names.

The `EDF_LOG_LEVEL` environment variable sets the default Forge log level.

If you choose to use `EDF_LOG_LEVEL`, the variable should be set to one of the new log level names, such as `WARN` or `ERROR`. Just as in previous versions of logging, the value set in `EDF_LOG_LEVEL` may be overridden by any command line argument that changes the global log level.

Chapter 18

The Forge Metrics Web Service

You can query a running Forge component for performance metrics using the Forge Metrics Web service. This makes Forge easier to integrate into your management and monitoring framework.

About the Forge Metrics Web service

The Forge Metrics Web service provides progress and performance metrics for Forge. You can use the output of this Web service in the monitoring tool or interface of your choice.

A running instance of Forge hosts a WSDL interface, `metrics.wsdl`. Using this WSDL interface, you can query Forge for specific information about its performance.

Metrics are hierarchical, with parent-child relationships indicated by their location in the tree. You can either give the service a full path to precisely the information you are seeking, or get the full tree and traverse it to find what you want.

The following is an example of the kind of information tree returned by the Forge Metrics Web service:

```
(Root)
   Start time: Wed Jan 24 14:34:14 2007
   Percent complete: 41.4%
   Throughput: 871 records/second
   Records processed: 24000
   Components
      IndexerAdapter
         Records processed: 24902
         Total processing time: 2.331 seconds
      PropDimMapper
         Records processed: 24902
         Total processing time: 6.983 seconds
      LoadMainData
         Records processed: 24903
        Total processing time: 8.19 seconds
```

Each metric can be one of three types:

- **Metric** — serves as a parent category for child metrics, without containing any data of its own.
- **Attribute metric** — stores attributes, such as the start time of the Forge being queried.

  For each attribute metric you request, you receive ID, Name, and Attribute Value (a string).

- **Measurement metric** — contains quantatative data, such as:

- Estimated percent complete.
- Overall throughput.
- Number of records processed.
- Per-component throughput.

For each measurement metric you request, you receive ID, Name, Measurement Units (a string), and Measurement Value (a number).

Note:

The Forge Metrics Web service does not tell you what step Forge is on or its estimated time to completion.

The service is not long-lived; it exits when Forge does. For this reason, you cannot use this service to find out how long the Forge run took.

The Forge Metrics Web service does not work in conjunction with parallel Forge.

About enabling Forge metrics

Before you can generate Forge metrics, you have tell Forge the port on which to set up the Forge Metrics Web service. By doing so, you also turn Forge metrics on.

In the Endeca Application Controller, you set the `web-service-port` when you provision the Forge component. You can do this three ways:

- In Endeca Workbench, on the **EAC Administration** page.
- In a provisioning file used with the eaccmd tool (for details on provisioning a Forge component, see the *Oracle Endeca Application Controller Guide*.
- Programmatically, via the `webServicePort` on the `ForgeComponentType` object. For details, see the *Oracle Endeca Application Controller Guide*.

Outside of the Application Controller environment, you can also set or change the Web service port (and thus turn on Forge metrics) at the Forge commandline. The commandline argument for setting the metrics port is `--wsport <port-number>` .

About enabling SSL security

You can enable SSL on the Forge component to make the Forge Metrics Web service secure.

For information on enabling SSL on the Forge component programmatically or while provisioning with eaccmd, see the *Oracle Endeca Application Controller Guide*.

Note: The Web services port disregards the `cipher` sub-element of the `ssl-configuration` element.

About using Forge metrics

Assuming Forge's `web-service-port` is set, when you start Forge, it serves up the Metrics Web service. You can then use any Web services interface to talk to it and request metrics.

You can request global information on the parent node, or request on a component-by-component basis. (Each pipeline component has corresponding metrics.) If you request "/" , the Metrics Web service returns the root and all of its children. To refine your request, you give the Web service the path to the node you are interested in.

The MetricsService API

The MetricsService interface includes the methods and classes described below.

The metrics schema is defined in `metrics.wsdl`, which is located in the `$ENDECA_ROOT/lib/services` directory on UNIX and `%ENDECA_ROOT%\lib\services` on Windows.

Methods

Name	Purpose	Parameters	Exception	Returns
`getMetric (MetricInputType getMetricInput)`	Lists the collection of metrics in an application.	`getMetricInput` is a `MetricInputType` object consisting of a path to the node you want to query and a Boolean setting that allows you to exclude that node's children from the query.	`MetricFault` is the error message returned when the method fails.	`getMetricOutput`, a string collection of `metrics`.

Classes

Name	Purpose	Properties
`MetricType`	A class that describes a metric.	`id` is a unique string identifier for the metric. `displayName` is the name for the metric, as it appears in the output file. `children` is a collection of metric objects.
`MetricListType`	A class that describes a list of metrics.	`metric` is a collection of metrics comprising this `MetricListType` object.
`MetricInputType`	A class that describes the input to the `getMetric` method.	`path` is the path to the node you want to query. Null indicates top level, returning the whole tree. `excludeChildren` lets you indicate if you want just the metrics of the node specified in `path` or those of its children too.
`MetricResultType`	A class that describes the output returned by the `getMetric` method.	`metric` is an object of type `MetricType`.
`AttributeType`	An extension of `MetricType`, the `AttributeType` class describes an attribute metric.	`value` is a string describing the attribute.

Name	Purpose	Properties
`MeasurementType`	An extension of `MetricType`, the `MeasurementType` class describes a measurement metric.	`value` is a double representing the value of the measurement metric. `units` is a string describing the unit of measure used by the metric.

Appendix A

Forge Flag Reference

This reference provides a description of the options (flags) used by the Forge program.

Forge flag options reference

The included table lists the different flag options that Forge takes.

The usage of Forge is as follows:

`forge [-bcdinov] [--options] <Pipeline-XML-File>`

<Pipeline-XML-File> can be a relative path or use the `file://[hostname]/` protocol.

Forge takes the following options:

Important: All flags are case-sensitive.

Option	Description
`-b <cache-num>`	Specify the maximum number of records that the record caches should buffer. This may be set individually in the **Maximum Records** field of the **Record Cache editor** in Developer Studio.

`-c <name=value>`	Forge has a set of XML entity definitions whose values can be overridden at the command line, such as `current_date`, `current_time`, and `end_of_line`. You can specify a replacement string for the default entity values using the `-c` option, or in an `.ini` file specified with `-i` (described below).
	The format is:
	`<configValName=configVal>`
	For example:
	`end_of_line="\n"`
	which would be specified on the command line with:
	`-c end_of_line="\n"`

Option	Description
	or included as a line in an `.ini` file specified with `-i`. This allows you to assign pipeline values to Forge at the command line. In the above example, you would specify `&end_of_line;` in your pipeline file instead of hard-coding "\n", then invoke Forge with the `-c` option shown above. Forge would substitute "\n" whenever it encountered `&end_of_line;`. For a complete list of entities and their default values, see the ENTITY definitions in Endeca_Root/conf/dtd/common.dtd.
`-d <dtd-path>`	Specify the directory containing DTDs (overrides the `DOCTYPE` directive in XML).
`-i <ini-filename>`	Specify an `.ini` file that contains XML entity string replacements. Each line must be in this form: `<configValName=configVal>` See the description of the `-c` option for details.
`-n <parse-num>`	Specify the number of records to pull through the pipeline. This option is ignored by the record cache component.
`-o <filename>`	Specify an output file for messages.
`-v[f|e|w|i|d]`	Set the global log level. See `--logLevel` for corresponding information. If the `-v` option is omitted, the global log level defaults to `d` (`DEBUG`) or the value set in the *EDF_LOG_LEVEL* environment variable. If the `-v` option is used without a level, it defaults to `d` (`DEBUG`). `f` = `FATAL` messages only. `e` = `ERROR` and `FATAL` messages. `w` = `WARNING`, `ERROR`, and `FATAL` messages. `i` = `INFO`, `WARNING`, `ERROR`, and `FATAL` messages. `d` = `DEBUG`, `INFO`, `WARNING`, `ERROR`, and `FATAL` messages. **Note:** Options -v[a|q|s|t|v] have been deprecated.
`--client <server:port>`	Run as a client and connect to a Forge server in a Parallel Forge environment.
`--clientNum <num>`	Direct a Forge server to use *<num>* instead of assigning a client number. Useful when the client number must remain consistent (that is, it must start from zero and be sequential for all clients). Requires the `--client` option.

Option	Description
`--combineWarnCount <num>`	Specify the number of records that can be combined (via a Combine join or a record cache with the **Combine Records** setting enabled) before issuing a warning that performance may be slow. The default is `100`, while `0` will disable the warnings.
`--compression <num> | off`	Instruct Forge to compress the output to a level of <*num*>, which is `0` to `9` (where `0` = minimum, `9` = maximum). Specify `off` to turn off compression.
`--connectRetries <num>`	Specify the number of retries (`-1` to `100`) when connecting to the server. The default is `12` while `-1` = retry forever. Requires the `--client` option.
`--encryptKey [user:]<password>`	*Deprecated*. Encrypt a key pair so that only Forge can read it.
`--help [option]`	Print full help if used with no options. Prints specific help with these options (option names and arguments are case sensitive): • `expression` = Prints help on expression syntax. • `expression:TYPE` = Prints help on the syntax for a specific expression type, which can be `DVAL`, `FLOAT`, `INTEGER`, `PROPERTY`, `STREAM`, `STRING`, or `VOID`. • `config` = Prints help on configuration options.
`--idxCompression [<num> | off]`	Set the compression of the IndexerAdapter output Forge to a level of <*num*>, which is `0` to `9` (where `0` = minimum, `9` = maximum). Specify `off` to turn off compression.
`--ignoreState`	Instruct Forge to ignore any state files on startup. The state files are ignored only during the startup process. After start up, Forge creates state files during an update and overwrites the existing state files.
`--indexConfigDir <path>`	Instruct Forge to copy index configuration files from the specified directory to its output directory.
`--inputDir <path>`	Instruct Forge to load input data from this directory.<*path*> must be an absolute path and will be used as a base path for the pipeline. Any relative paths in the pipeline will be relative to this base path.  **Note:** If the pipeline uses absolute paths, Forge ignores this flag.
`--input-encoding <encoding>`	*Deprecated*. Specify the encoding of non-XML input files.

Option	Description
`--javaArgument <java_arg>`	Prepend the given Java option to the Java command line used to start a Java virtual machine (JVM).
`--javaClasspath <classpath>`	Override the value of the **Class path** field on the **General** tab of the Record adapter, if one is specified. If the Record adapter has a **Format** setting with **JDBC** selected, then **Class path** indicates the JDBC driver. If the Record adapter has a **Format** setting with **Java Adapter** selected, then **Class path** indicates the absolute path to the custom record adapter's `.jar` file.
`--javaHome <java_home>`	Specifies the location of the Java runtime engine (JRE). This option overrides the value of the **Java home** field on the **General** tab of a Record adapter, if one is specified. The `--javaHome` setting requires Java 2 Platform Standard Edition 5.0 (aka JDK 1.5.0) or later.
`--logDir <path>`	Instructs Forge to write logs to this directory, overriding any directories specified in the pipeline.
`--logLevel (<topicName>=) <logLevel>`	Set the global log level and/or topic-specific log level. If this option is omitted, the value defaults to `INFO` or to that set in the `EDF_LOG_LEVEL` environment variable. For corresponding information, see the `-v` option. Possible log levels are: • `FATAL` = `FATAL` messages only. • `ERROR` = `ERROR` and `FATAL` messages. • `WARN` = `WARN`, `ERROR`, and `FATAL` messages. • `INFO` = `INFO`, `WARN`, `ERROR`, and `FATAL` messages. • `DEBUG` = `DEBUG`, `INFO`, `WARN`, `ERROR`, and `FATAL` messages. Possible topics for Forge are: • baseline • update • config • webservice • metrics
`--noAutoGen`	Do not generate new dimension value IDs (for incremental updates when batch processing is running).

Option	Description
`--numClients <num>`	The number of Parallel Forge clients connecting. Required with `--server` option.
`--numPartitions <num>`	Specify the number of Dgidx instances available to Forge. This number corresponds to the number of Dgraphs, which in turn corresponds to the number of file sets Forge creates. This option overrides the value of the `NUM_IDX` attribute in the `ROLLOVER` element of your project's `Pipeline.epx` file, if one is specified.
`--outputDir <path>`	Instruct Forge to save output data to this directory, overriding any directories specified in the pipeline.
`--outputPrefix <prefix>`	Override the value specified in **Output** prefix field of the **Indexer Adapter** or **Update Adapter editors** in your Developer Studio pipeline.
`--perllib <dir>`	Add <*dir*> to perl's library path. May be repeated.
`--pidfile <pidfile-path>`	File in which to store process ID (`PID`).
`--printRecords [number]`	Print records as they are produced by each pipeline component. If number is specified, start printing after that number of records have been processed.
`--pruneAutoGen`	Instructs Forge to remove from the AutoGen state any dimensions that have been promoted as internal dimensions. When a pipeline developer promotes a dimension that was automatically generated, the dimension is copied into the `dimensions.xml` file and is removed from the AutoGen state file.
`--retryInterval <num>`	Specify the number of seconds (`0` to `60`) to sleep between connection attempts. The default is `5`. Requires the `--client` option.
`--server <portNum>`	Run as a server and listen on port specified Requires the `--numClients` option.
`--spiderThrottle <wait>:<expression_type>:<expression>`	*Deprecated*. During a crawl, throttle the rate at which URLs are fetched by the spider, where: <*wait*> is the fetch interval in seconds. <*expression_type*> specifies the type of regular or host expression to use: • `url-regex` • `url-wildcard` • `host-regex` • `host-wildcard`

Option	Description
	<*expression*> is the corresponding expression. **Example:** `--spiderThrottle 10:url-wildcard:*.html` This would make all URLs that match the wildcard "`*.html`" wait 10 seconds between fetches.
`--sslcafile <CAcertfile-path>`	Specify the path of the eneCA.pem Certificate Authority file that the Forge server and Forge clients will use to authenticate each other.
`--sslcertfile <certfile-path>`	Specify the path of the `eneCert.pem` certificate file that will be used by the Forge server and Forge client for SSL communications.
`--sslcipher <cipher>`	Specify one or more cryptographic algorithms, one of which Dgraph will use during the SSL negotiation. If you omit this setting, the Dgraph chooses a cryptographic algorithm from its internal list of algorithms. See the *Endeca Commerce Security Guide* for more information.  **Note:** This setting is ignored by the `--wsport` flag, even when it uses SSL to secure its communications.
`--stateDir <path>`	Instruct Forge to persist data in this directory, overriding any directories specified in the pipeline.
`--tmpDir <path>`	Instruct Forge to write temporary files in the specified directory, overriding any directories specified by environment variables. The <*path*> value is interpreted as being based in Forge's working directory, **not** in the directory containing `Pipeline.epx`.
`--time <comp>`	Timing statistics (`comp` = time each component).
`--timeout <num>`	Specify the number of seconds (from `-1` to `300`) that the server waits for clients to connect. Default is `60` and `-1` means wait forever. Requires the `--server` option.
`--version`	Print out the current version information.
`--wsport <portNum>`	Start the Forge Metrics Web service, which is off by default. It listens on the port specified.

Appendix B

File Formats Supported by the Document Conversion Module

This section lists the file formats that are supported by the Endeca Document Conversion Module. After installing this module, you can use the `CONVERTTOTEXT` expression in your pipeline to convert any of the supported source document formats. The Endeca Web Crawler and the Endeca CAS Server provide tight integrations with the Document Conversion Module, which means that they can convert binary files as they are being crawled.

Word processing formats

The following table lists supported word processing formats:

Format	Version (if applicable)
Adobe FrameMaker (MIF)	Versions 3.0 - 6.0
Adobe Illustrator Postscript	Level 2
Ami	
Ami Pro for OS2	
Ami Pro for Windows	2.0, 3.0
DEC DX	Through 4.0
DEC DX Plus	4.0, 4.1
Enable Word Processor	3.0 - 4.5
First Choice WP	1.0, 3.0
Framework WP	3.0
Hangul	97 - 2007
IBM DCA/FFT	
IBM DisplayWrite	2.0 - 5.0
IBM Writing Assistant	1.01
Ichitaro	5.0, 6.0, 8.0 - 13.0, 2004

Format	Version (if applicable)
JustWrite	Through 3.0
Kingsoft WPS Writer	2010
Legacy	1.1
Lotus Manuscript	Through 2.0
Lotus WordPro	9.7, 96, - Millennium 9.6
Lotus WordPro (non-Win32)	97 - Millennium 9.6
MacWrite II	1.1
Mass 11	All versions through 8.0
Microsoft Publisher (File ID only)	2003 - 2007
Microsoft Word for DOS	4.0 - 6.0
Microsoft Word for Macintosh	4.0 - 6.0, 98 - 2008
Microsoft Word for Windows	1.0 - 2007
Microsoft Word for Windows	98-J
Microsoft WordPad	
Microsoft Works WP for DOS	2.0
Microsoft Works WP for Macintosh	2.0
Microsoft Works WP for Windows	3.0, 4.0
Microsoft Write for Windows	1.0 - 3.0
MultiMate	Through 4.0
MultiMate Advantage	2.0
Navy DIF	
Nota Bene	3.0
Novell Perfect Works	2.0
Office Writer	4.0 - 6.0
OpenOffice Writer	1.1 - 3.0
Oracle Open Office Writer	3.x
PC File Doc	5.0
PFS:Write	Versions A, B
Professional Write (DOS)	1.0, 2.0
Professional Write Plus (Windows)	1.0
Q&A Write (Windows)	2.0, 3.0
Samna Word IV	1.0 - 3.0
Smna Work IV+	
Samsung JungUm Global (File ID only)	

Format	Version (if applicable)
Signature	1.0
SmartWare II WP	1.02
Sprint	1.0
StarOffice Writer	5.2 - 9.0
Total Word	1.2
Wang PC (IWP)	Versions through 2.6
WordMarc Composer	
WordMarc Composer+	
WordMarc Word Processor	
WordPerfect for DOS	4.2
WordPerfect for Macintosh	1.02 - 3.1
WordPerfect for Windows	5.1 - X4
WordStar 2000 for DOS	1.0 - 3.0
WordStar 2000 for DOS	2.0, 3.0
WordStar for DOS	3.0 - 7.0
WordStar for Windows	1.0
XyWrite	Through III+

Text and markup formats

The following table lists supported text and markup formats:

Notes:

- The Document Conversion Module supports converting XML content contained in both PCDATA and CDATA elements.
- In the case of XHTML, "file ID only" means that the conversion process produces an Endeca property representing the file format type but nothing else.

Format	Version (if applicable)
ANSI Text	7 bit and 8 bit
ASCII Text	7 bit and 8 bit
DOS character set	
EBCDIC	
HTML (CSS rendering not supported)	1.0 - 4.0
IBM DCA/RFT	
Macintosh character set	

Format	Version (if applicable)
Rich Text Format (RTF)	
Unicode Text	3.0, 4.0
UTF-8	
Wireless Markup Language	1.0
XML	text only
XHTML (file ID only)	1.0

Spreadsheet formats

The following table lists supported spreadsheet formats:

Format	Version
Enable Spreadsheet	3.0 - 4.5
First Choice SS	Through 3.0
Framework SS	3.0
IBM Lotus Symphony Spreadsheets	1.x
Kingsoft WPS Spreadsheets	2010
Lotus 1-2-3	Through Millennium 9.6
Lotus 1-2-3 Charts (DOS and Windows)	Through 5.0
Lotus 1-2-3 (OS/2)	2.0
Microsoft Excel Charts	2.x - 2007
Microsoft Excel for Macintosh	98 - 2008
Microsoft Excel for Windows	3.0 - 2010
Microsoft Excel for Windows (xslb)	2007 - 2010 Binary
Microsoft Multiplan	4.0
Microsoft SS Works for DOS	2.0
Microsoft Works for Macintosh	2.0
Microsoft SS Works for Windows	3.0, 4.0
Novell PerfectWorks	2.0
OpenOffice Calc	1.1 - 3.0
Oracle Open Office Calc	3.x
PFS: Professional Plan	1.0
Quattro for DOS	Through 5.0
QuattroPro for Windows	Through X4

Format	Version
SmartWare Spreadsheet	
SmartWare II SS	1.02
StarOffice Calc	5.2 - 9.0
SuperCalc	5.0
Symphony	Through 2.0
VP Planner	1.0

Vector image formats

The following table lists supported vector image formats:

Format	Version (if applicable)
Adobe Illustrator	4.0 - 7.0, 9.0
Adobe Illustrator (XMP only)	11 - 13 (CS 1 - 3)
Adobe InDesign (XMP only)	3.0 - 5.0 (CS 1 - 3)
Adobe InDesign Interchange (XMP only)	
Adobe Photoshop (XMP only)	8.0 -10.0 (CS 1 - 3)
Adobe PDF	1.0 - 1.7 (Acrobat 1 - 9)
Adobe PDF Package	1.7 (Acrobat 8 - 9)
Adobe PDF Portfolio	1.7 (Acrobat 8 - 9)
Adobe Photoshop	4.0
Ami Draw	SDW
AutoCAD Drawing	2.5, 2.6
AutoCAD Drawing	9.0 - 14.0
AutoCAD Drawing	2000i - 2007
AutoShade Rendering	2.0
Corel Draw	2.0 - 9.0
Corel Draw Clipart	5.0, 7.0
Enhanced Metafile (EMF)	
Escher graphics	
FrameMaker Vector and Raster Graphics (FMV)	3.0 - 5.0
Gem File (Vector)	
Harvard Graphics Chart (DOS)	2.0 - 3.0

Format	Version (if applicable)
Harvard Graphics for Windows	
HP Graphics Language	2.0
Initial Graphics Exchange Specification (IGES) Drawing	5.1 - 5.3
Micrografx Designer	Through 3.1
Micrografx Designer	6.0
Micrografx Draw	Through 4.0
Microsoft XPS (Text only)	
Novell PerfectWorks Draw	2.0
OpenOffice Draw	1.1 - 3.0
Oracle Open Office Draw	3.x
Visio (Page Preview mode only WMF/EMF)	4
Visio	5.0 - 2007
Visio XML VSX (File ID only)	2007
Windows Metafile	

Notes on Adobe PDF text extraction

The CAS Document Conversion Module works as follows when processing Adobe PDF files with security settings:

- The CAS Document Conversion Module will respect the no-copy option of a PDF. That is, if a PDF publishing application has a no-copy option (which prohibits the copying or extraction of text within the PDF), the Document Conversion Module will not extract text from that PDF.
- The CAS Document Conversion Module does not support text extraction from password-protected files.
- The CAS Document Conversion Module does not support text extraction from PDFs with encrypted content.

To extract the text from these types of PDFs, you must re-create them without setting the appropriate security option.

In addition, text added with the Sticky Note tool is not extracted.

Raster image formats

The following table lists supported raster image formats:

Format	Version
CALS Raster (GP4)	Type I and Type II
Computer Graphics Metafile	ANSI, CALS, NIST
Encapsulated PostScript (EPS)	TIFF header only
GEM Image (Bitmap)	

Format	Version
Graphics Interchange Format (GIF)	
IBM Graphics Data Format (GDF)	1.0
IBM Picture Interchange Format (PIF)	1.0
JBIG2	graphic embeddings in PDF files
JFIF (JPEG not in TIFF format)	
JPEG	
JPEG 2000	JP2
Kodak Flash Pix	
Kodak Photo CD	1.0
Lotus PIC	
Lotus Snapshot	
Macintosh PICT1 and PICT2	BMP only
MacPaint	
Microsoft Windows Bitmap	
Microsoft Windows Cursor	
Microsoft Windows Icon	
OS/2 Bitmap	
OS/2 Warp Bitmap	
Paint Shop Pro (Win32 only)	5.0, 6.0
PC Paintbrush (PCX)	
PC Paintbrush DCX (multi-page PCX)	
Portable Bitmap (PBM)	
Portable Graymap (PGM)	
Portable Network Graphics (PNG)	
Portable Pixmap (PPM)	
Progressive JPEG	
StarOffice Draw	6.x - 9.0
Sun Raster	
TIFF	Group 5 and Group 6
TIFF CCITT Fax	Group 3 and Group 4
Truevision TGA (Targa)	2.0
WBMP wireless graphics format	
Word Perfect Graphics	1.0
X-Windows Bitmap	x10 compatible

Format	Version
X-Windows Dump	x10 compatible
X-Windows Pixmap	x10 compatible
WordPerfect Graphics	2.0, 7.0, 8.0, 9.0, 10.0

Presentation formats

The following table lists supported presentation formats:

Format	Version (if applicable)
Corel Presentations	6.0 - X3
Harvard Graphics (DOS)	3.0
IBM Lotus Symphony Presentations	1.x
Kingsoft WPS Presentation	2010
Lotus Freelance	1.0 - Millennium 9.6
Lotus Freelance (OS/3)	2.0
Lotus Freelance for Windows	95, 97
Microsoft PowerPoint for Macintosh	4.0 - 2008
Microsoft PowerPoint for Windows	3.0 - 2010
Microsoft PowerPoint for Windows Slideshow	2007 - 2010
Microsoft PowerPoint for Windows Template	2007 - 2010
Novell Presentations	3.0, 7.0
OpenOffice Impress	1.1, 3.0
Oracle Open Office Impress	3.x
StarOffice Impress	5.2 - 9.0
WordPerfect Presentations	5.1 - X4

Archive formats

The following table lists supported archive formats:

Format	Version (if applicable)
LZA Self Extracting Compress	
LZH Compress	
Microsoft Binder	95, 97
RAR	1.5, 2.0, 2.9

Format	Version (if applicable)
Self-extracting .exe	
UNIX Compress	
UNIX GZip	
UNIX TAR	
Uuencode	
ZIP	PKZip
ZIP	WinZip

Database formats

The following table lists supported database formats:

Format	Version
DataEase	4.x
DBase	III, IV, and V
First Choice DB	Through 3.0
Framework DB	3.0
Microsoft Access	1.0, 2.0
Microsoft Access Report Snapshot (File ID only)	2000 - 2003
Microsoft Works DB for DOS	1.0, 2.0
Microsoft Works DB for Macintosh	2.0
Microsoft Works DB for Windows	3.0, 4.0
Paradox (DOS)	2.0 - 4.0
Paradox (Windows)	1.0
Q & A	Through 2.0
R:Base	R:Base 5000 and R:Base System V
Reflex	2.0
SmartWare II	1.02

E-mail formats

The following table lists supported e-mail formats:

Format	Version
Encoded mail messages	MHT

Format	Version
Encoded mail messages	Multi Part Alternative
Encoded mail messages	Multi Part Digest
Encoded mail messages	Multi Part Mixed
Encoded mail messages	Multi Part News Group
Encoded mail messages	Multi Part Signed
Encoded mail messages	TNEF
IBM Lotus Notes Domino XML Language DXL	8.5
IBM Lotus Notes NSF (File ID only)	7.x, 8.x
IBM Lotus Notes NSF (Windows only with Notes client or Domino Server)	8.x
Microsoft Outlook MSG	97 - 2007
Microsoft Outlook Express (EML)	
Microsoft Outlook Forms Template (OFT)	97 - 2007
Microsoft Outlook OST	97 - 2007
Microsoft Outlook PST	97 - 2007
Microsoft Outlook PST (Mac)	2001

Other formats

The following table lists other supported formats:

Format	Version (if applicable)
Microsoft InfoPath (file ID only)	2007
Microsoft OneNote (file ID only)	2007
Microsoft Project (text only)	98 - 2003
Microsoft Project (file ID only)	2007
Microsoft Windows DLL	
Microsoft Windows Executable	
vCard	2.1
vCalendar	2.1
Yahoo! Messenger	6.x - 8.0

Appendix C

Advanced JDBC Column Handler

The Advanced JDBC Column Handler is an extension to the standard JDBC record adapter. It provides support for obtaining data from database column types that are not supported by the standard JDBC record adapter, such as CLOBs and BLOBs.

About the Advanced JDBC Column Handler

The Endeca Data Transformation Layer provides a Java-based database adapter for use with any database that has a JDBC driver. This JDBC adapter can be used as a record adapter type to retrieve Endeca records as rows of a SQL query result. Not all database column types are supported by this out-of-the-box Endeca JDBC adapter. See the JDBC Input Format section of Endeca Developer Studio Help for a list of supported database column types.

The Advanced JDBC Column Handler extends the set of supported database column types by providing handlers for the following column types:

- LONGVARCHAR
- CLOB
- BLOB
- LONGVARBINARY
- BINARY
- VARBINARY

JDBC driver

The JDBC Driver for the specific database type is required for using the JDBC record adapter and should be located somewhere on the file system.

JDBC configuration options

Create a properties file named `columnHandler.properties`.

Place this file in the `ENDECA_ROOT\lib\java` directory. The following table lists the configuration options used in the `columnHandler.properties` file:

`binaryDataChunkSize`	The size (in bytes) of the incremental data chunk read and written while processing binary columns. This is a

	performance optimization setting; changing this value will not affect the values ultimately returned by the Advanced JDBC Column Handler. Default setting: `1024`
`charDataChunkSize`	The size (in bytes) of the incremental data chunk read and written while processing character-data columns. This is a performance optimization setting; changing this value will not affect the values ultimately returned by the Advanced JDBC Column Handler. Default setting: `1024`
`outputDataDir`	The directory to which data files will be written (see File System Output). Relative paths to an output directory will be relative to forge's working directory. Default setting: `../incoming`
`charDataToDisk`	If this setting is `true` , character data will be written out to the file system (encoded according to the value of the `charDataOut¬putEnc` property), and the path to the output file will be returned as the property's value. If `false`, the character data will be returned as the property's value. Set this option to `true` for very large `char` columns. Default setting: `false`
`charDataOutputEnc`	Output encoding to use when writing character data to disk. A valid Java charset name must be specified. Some common encoding charsets are listed below: • US-ASCII • ISO-8859-1 • UTF-8 • UTF-16BE • UTF-16LE • UTF-16 This setting is only used if `charDataToDisk` is *true*. Default setting: `UTF-8`

Here is a sample `columnHandler.properties` file:

```
binaryDataChunkSize=1024
charDataChunkSize=1024
outputDataDir=C:\\Endeca\\Apps\\Discover\\data\\forge_output
charDataToDisk=true
charDataOutputEnc=UTF-8
```

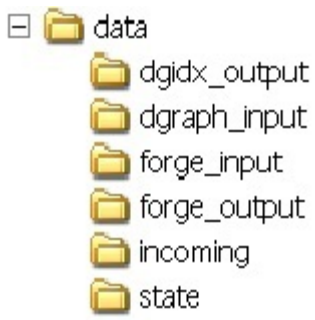
If Forge is triggered in standalone mode then this configuration file can be placed in the same directory as the `adapter.jar` file. The `outputDataDir` in that case should point to the current working directory.

During a baseline update using the JDBC Column Handler, if this file is not present at `$ENDECA_ROOT\lib\java`, there is a mismatch between the output folders during execution and the output is not what is expected.

Storing data on disk

If binary columns will be used, or if character data columns will be output to the file system, you must be sure the output directory exists.

Unless overridden by the `outputDataDir` setting described previously, the default output directory will be the incoming directory parallel to Forge's working directory. For example, the following directory tree shows this incoming directory, assuming `forge_input` is configured as the Forge working directory:



Note that the incoming directory is accessible at `../incoming` relative to the pipeline files within `forge_input/`.

On initialization, the Advanced JDBC Column Handler will check to see if the `outputDataDir` is writable. If it is not, the Advanced JDBC Column Handler will throw a warning into the Forge log, similar to the following:

```
WARN    09/24/14 14:00:49.958 UTC  FORGE      {forge,baseline}: (com.ende¬
ca.soleng.itl.jdbc.AdvancedJDBCColumnHandler): outputDataDir ../incoming not
writable
```

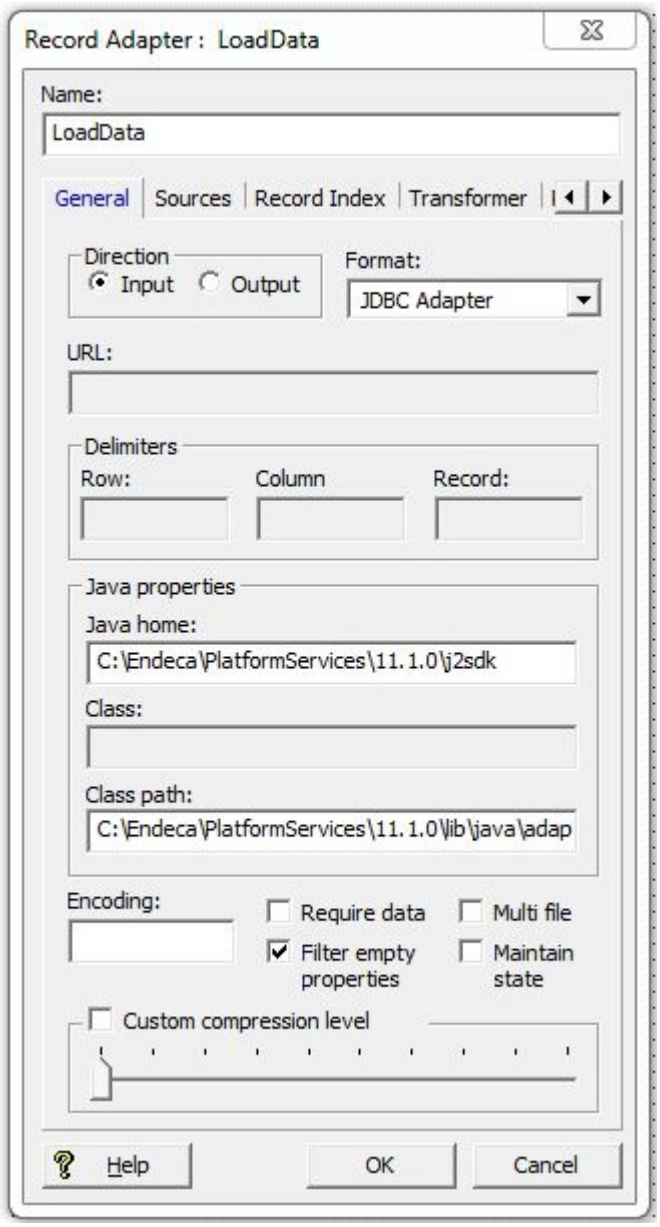
Because the Advanced JDBC Column Handler does not require a writable directory if only character columns are used and character data is not spooled to disk, Forge will continue processing when it encounters a non-writable `outputDataDir`. However, if binary columns are used or if character data is explicitly spooled to disk, Forge will not return any valid data for those columns, and will log additional warnings to the Forge log for each row and column in the database it is not able to process. These warnings will appear similar to the following:

```
WARN    09/24/06 14:02:34.828 UTC  FORGE      {forge,baseline}: (com.ende¬
ca.soleng.itl.jdbc.AdvancedJDBCColumnHandler): IOException while dumping MyBina¬
ryData column
```

Using the Advanced JDBC Column Handler

Create a JDBC record adapter as usual, with PASS_THROUGH values like DB_DRIVER_CLASS, DB_URL, and SQL. See the JDBC section in *Developer Studio Help* for information about how to create a JDBC record adapter.

The following is an example of Record Adapter in Developer Studio:

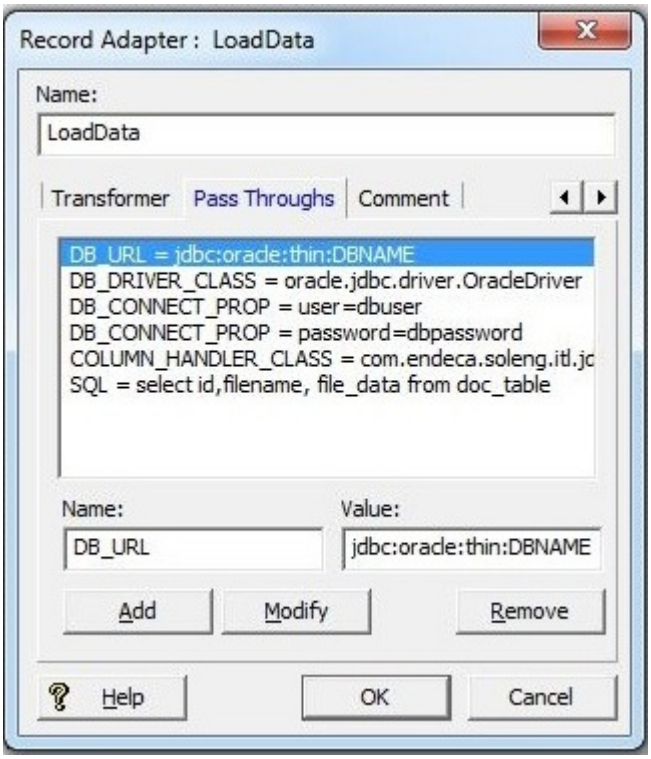


Java home must be set as `$ENDECA_ROOT\j2sdk`

The class path must contain paths to:

- The `adapter.jar` location, for example, `$ENDECA_ROOT\lib\java\adapter.jar`.
- The JDBC driver, for example, `$ENDECA_ROOT\lib\java\ojdbc.jar`.
- The directory where `columnHandler.properties` is located, for example, `C:\Endeca\`.

Add one new PASS_THROUGH with name COLUMN_HANDLER_CLASS and value `com.ende¬ca.soleng.itl.jdbc.AdvancedJDBCColumnHandler`. The following is an example of a complete JDBC record adapter with this additional pass through:



Whether Forge is run on the command line or from a control script, the `adapter.jar` file must be added to the `--javaClasspath` argument. Also, if the `columnHandler.properties` configuration file will be used to override any default options the directory containing the `columnHandler.properties` needs to be added to the `--javaClasspath` argument as well. The following example illustrate the use of this argument when Forge is run from the command line:

```
forge --javaClasspath /path/to/adapter.jar:/path/to/JDBCDriver.jar:/path/to/props¬
file_directory /path/to/Pipeline.epx
```

If your project requires additional JAR files, such as those required by the JDBC driver, be sure to include those references as well. Also note that Unix systems use a colon to delimit multiple JARs, while Windows uses a semicolon. For information about how to set the classpath, refer to *Developer Studio Help*.

Note: Instead of specifying clear text credentials, you can use Oracle Credentials Store (OCS) to specify database credentials information. In which case instead of specifying the username and password information along with DB_URL or DB_CONNECT_PROP, you need to use the passthrough CREDENTIALS_KEY and provide the key name that should be used to retrieve the credentials from Oracle Credentials Store. For information about how to use the OCS, refer to the *Oracle Commerce Guided Search Security Guide*.

Output

File system output

The Advanced JDBC Column handler optionally writes character data (`CLOB`, `LONGVARCHAR` types) to the file system, but by default these character values are returned inline as the Endeca property's value. If configured for output to the file system using the `charDataToDisk` option mentioned above, the files will be created in

the `outputDataDir` directory (also configurable) and would have filenames of the form `clob_dataN.tmp`. *N*, in this case, is a random number suffix to keep these temporary files distinct.

The binary column type handlers always write their data to the file system, in the `outputDataDir` directory. These file names are of the form `blob_dataN.tmp`. The relative path to each file is returned as the Endeca property's value. For example `../incoming/blob_data22607.tmp`. The pipeline must then read this file in a subsequent record manipulator.

Importing character data with IMPORT_PROP

If the `charDataToDisk` option is enabled, character data will be written to the file system. One typical way to acquire the data in these files is to build a record manipulator that uses the IMPORT_PROP expression to read in the character data. The following is an example of such a record manipulator:

```
<RECORD_MANIPULATOR FRC_PVAL_IDX="TRUE" NAME="BLOB Manip.">
<RECORD_SOURCE>Records In</RECORD_SOURCE>
<EXPRESSION LABEL="" NAME="IF" TYPE="VOID" URL="">

<COMMENT>if a reference to a CLOB file exists...</COMMENT>
<EXPRESSION LABEL="" NAME="PROP_EXISTS" TYPE="INTEGER" URL="">
 <EXPRNODE NAME="PROP_NAME" VALUE="CLOB_COL_NAME"/>
</EXPRESSION>

<EXPRESSION LABEL="" NAME="IMPORT_PROP" TYPE="VOID" URL="">
<COMMENT>pull in the char data and remove the file</COMMENT>
 <EXPRNODE NAME="PROP_NAME" VALUE="CLOB_COL_NAME"/>
 <EXPRNODE NAME="REMOVE_FILES" VALUE="TRUE"/>
 <EXPRNODE NAME="ENCODING" VALUE="UTF-8"/>
</EXPRESSION>

</EXPRESSION>
</RECORD_MANIPULATOR>
```

Processing binary data with the Document Converter

One typical usage scenario for binary column data is to read in documents like PDFs or Word files from the database. In this case, the Advanced JDBC Column Handler would write out this binary column data to the temporary files mentioned above. Then the pipeline would invoke the Document Converter to convert these binary-formatted files into plaintext Endeca properties indexed for search. The following example pipeline component could be used to do this conversion:

```
<RECORD_MANIPULATOR FRC_PVAL_IDX="TRUE" NAME="BLOB Manip.">
<RECORD_SOURCE>Records In</RECORD_SOURCE>
<EXPRESSION LABEL="" NAME="IF" TYPE="VOID" URL="">

<COMMENT>if a reference to a BLOB file exists...</COMMENT>
<EXPRESSION LABEL="" NAME="PROP_EXISTS" TYPE="INTEGER" URL="">
 <EXPRNODE NAME="PROP_NAME" VALUE="BLOB_COL_NAME"/>
</EXPRESSION>

<EXPRESSION LABEL="" NAME="RENAME" TYPE="VOID" URL="">
<COMMENT>… rename the BLOB property,</COMMENT>
 <EXPRNODE NAME="OLD_NAME" VALUE="BLOB_COL_NAME"/>
 <EXPRNODE NAME="NEW_NAME" VALUE="Endeca.Document.Body"/>
</EXPRESSION>

<EXPRESSION LABEL="" NAME="CONVERTTOTEXT" TYPE="VOID" URL="">
<COMMENT>extract the searchable text from the file,</COMMENT>
 <EXPRNODE NAME="RESPONSE_TIMEOUT" VALUE="300"/>
</EXPRESSION>
```

```
<EXPRESSION TYPE="VOID" NAME="REMOVE_EXPORTED_PROP">
<COMMENT>and then remove the file from the filesystem.</COMMENT>
 <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Document.Body"/>
 <EXPRNODE NAME="REMOVE_PROPS" VALUE="TRUE"/>
</EXPRESSION>

</EXPRESSION>
</RECORD_MANIPULATOR>
```

Note that the binary column's property name should be renamed to Endeca.Document.Body, since this is the property sought by the Document converter module. After this manipulator processes a record, it will create properties like Endeca.Document.Text, which contains the converted document text and Endeca.Document.Encoding, which reflects the binary file format detected. For more information on the Document converter module, see the VOID CONVERTTOTEXT section of the Data Foundry Expression Reference.

Troubleshooting

Logging output

Logging output will be directed to the Forge log. Non-fatal warning messages may be seen here. For example, if a column handler encounters some sort of stream-reading error, the following log message would appear in the forge log file:

```
WARN    09/22/14 14:00:49.958 UTC  FORGE      {forge,baseline}: (com.ende¬
ca.soleng.itl.jdbc.AdvancedJDBCColumnHandler): Could not read data from LONGVARCHAR
 column “text”: out of memory
```

The record returned for this row will have a null value for the property in question. Processing will continue with the next row in the SQL query result. The only fatal errors which stop Forge from running will occur if an unsupported column type is encountered. Unsupported Java SQL column types include the following:

- ARRAY
- DISTINCT
- JAVA_OBJECT
- OTHER
- REF
- STRUCT

If any of these column types are returned in the SQL result, Forge will produce an error message like the following:

```
ERROR   07/31/14 13:29:07.903 UTC FORGE {forge,baseline}: (AdapterRunner): Unsup¬
ported Java SQL column type ARRAY
```

JDBC driver

The Advanced JDBC Column Handler processes data retrieved by the JDBC database driver. If you encounter any problems, the first step is to ensure that the database driver is performing correctly. Testing the database driver outside of forge will verify this; a good first step is to test the driver using a standalone Java program.

If you experience errors using the column handler with Oracle's JDBC drivers, you should check the following guidelines:

- If you are using Oracle 9i or later, make sure that you are using the JDBC driver implementation that came with your Oracle server installation. If you have a patched revision of the Oracle server (for example.

9.2.0.6), it is likely that the patch contains an updated JDBC driver. Check to make sure that you are using the patched JDBC driver.

- If you are using earlier versions of Oracle 9i, try using the OCI driver instead of the thin driver. Later versions of the Oracle 9i thin driver have full support for CLOBs and BLOBs, but earlier Oracle 9i drivers do not.
- If you are using Oracle 8i or earlier, contact Oracle Customer Support.

Index

A

adding components to a pipeline 44
Advanced JDBC Column Handler 133
Auto Generate mode
    described 36
    saving state information for 49

B

basic pipeline
    dimension adapter 49
    dimension server 49
    indexer adapter 51
    property mapper 50
    record adapter 48
    testing 53

C

combine joins 73
Combine Records setting in record caches 88
component names as used in a pipeline 44

D

data processing
    general workflow 21
    in detail 21
    loading raw data 22
    mapping source properties to dimensions 24
    standardizing properties 23
    writing out finished data 24
default mappings
    enabling 40
    overriding with null mappings 33
Default Maximum Length 41
    override 41
Developer Studio 18
Dgidx
    introduced 14
    running 17
Dgraph, running the 17
dimension adapter 49
dimension groups 60
dimension hierarchy 15
    configuring in Developer Studio 18
dimension mapping 24, 35
    advanced techniques 39
    Auto Generate mode 36
    behavior when no mapping is found 40
    default mapping 40
    example 37
dimension mapping *(continued)*
    implicit mapping 39
    Must Match mode 36
    Normal match mode 35
    priority order for advanced techniques 30
    source properties to like-named dimensions 39
    synonyms 57
    viewing existing 32
dimension search configured in Developer Studio 18
dimension server
    for persisting auto-generated dimensions 49
    overview 49
dimension values
    auto generating 36
    mapping to source property values 24
    specifying the order of 59
dimensions
    assigning multiple mappings to 34
    creating 56
    mapping to source properties 24
    specifying the order of 59
directory structure for the Endeca Application Controller 43
disjunct joins 70
Document Conversion module
    other supported formats 132
    supported compressed formats 130
    supported database formats 131
    supported e-mail formats 131
    supported presentation formats 130
    supported raster image formats 128
    supported text and markup formats 125
    supported vector image formats 127
    supported word processing formats 123
dynamic business rules 60
    configuring in Developer Studio 18
    configuring in Oracle Endeca Workbench 18

E

emgr_update utility 104
Endeca Application Controller
    architecture 19
    communicating with 20
    communicating with Oracle Endeca Workbench 20
    directory structure 43
    introduced 19
Endeca CAS 13
Endeca Crawler
    Document Conversion module
        supported spreadsheet formats 126
Endeca Developer Studio
    creating a basic pipeline project 47
    creating and mapping dimensions 56

Endeca Developer Studio *(continued)*
creating and mapping Endeca properties 56
specifying index configuration options 60
using to add and edit pipeline components 44, 59
Endeca ITL
architecture 14
Data Foundry programs 14
data processing with 13
indexing
about 25
indexing with 13, 14, 25
introduced 13
loading raw data 22
mapping source properties to dimensions 24
standardizing source properties 23
writing out tagged data 24
Endeca properties
assigning multiple mappings to 34
creating 56
Endeca Tools setup information 19
Endeca tools suite 18
explicit mapping
creating 32
described 28
externally created dimensions
compared to externally managed taxonomies 93
Developer Studio configuration 94
importing 97
introduced 93
XML requirements 95
externally managed taxonomies
Developer Studio configuration 99
integrating 102
introduced 99
loading 104
node ID requirements 102
pipeline configuration 102
transforming 103
XML syntax 101
XSLT mapping 100

F

filtering unknown properties 28
first record joins 72
Forge
flags 117
introduced 14
running 17
Forge logging system 109
Forge metrics
enabling 114
using 115
Forge Metrics Web service 113
API 115
enabling SSL 114

H

higher cardinality joins 84

I

implicit mapping
described 29
enabling 39
overriding with null mappings 33
importing externally created dimensions 97
index configuration 15, 60
indexer adapters 51
inner joins 69
input components 22
instance configuration
creating 17
described 14

J

Java manipulators, about 59
JDBC Column Handler 133
join keys for data sources 66
joins
adding a record assembler 80
adding a record cache 79
cases where record caches are not required 87
choosing left and right 87
combine 73
combining equivalent records 88
configuring in a record assembler 81
creating record indexes 75
disjunct 70
first record 72
higher cardinality 84
implementing 79
inner 69
left 68
multiple keys in left joins 85
multiple values for join key 83
outer 69
overview 65
performing in a database 66
record index keys 67
sort switch 71
switch 70

L

left joins
described 68
multiple keys for records 85
loading source data 22
logging
aliasing v-levels 111
command line interface 110
EDF_LOG_LEVEL settings 111
levels 109

logging *(continued)*
    logLevel 110
    output file 111
    topics 110

M

mapping
    explicit 29
    source properties to dimensions 35
    source properties to like-named dimensions 39
match modes
    Auto Generate 36
    Must Match 36
    Normal 35
Multi Sub-records option for record assembler 85
multiple values for a join key 83
Must Match mode 36

N

node ID requirements for externally managed taxonomies 102
Normal match mode 35
null mapping
    described 29
    overriding implicit and default mappings 33

O

Oracle Endeca Workbench 18
outer joins 69

P

Perl assembler 59
pipeline 22
    adding components to 44
    creating a data flow for 44
    creating using the Basic Pipeline template 47
    described 14
    editing components in 44
    fundamentals 43
    placement of property mapper 30
    running 53
    sequential record processing 22
    URLs in 45
    using only one property mapper in 27
precedence rules
    introduced 18
    specifying in Developer Studio 18, 60
priority order of source property mapping 30
property mapper
    creating 31
    described 50
    minimum configuration 28
    placement in pipeline 30
    using only one per pipeline 27, 50
property mapper *(continued)*
    using the Mappings editor 32

R

record adapter
    overview 48
    record index 48
record assembler
    adding for joins 80
    configuring joins in 81
    creating join keys 76
    described 59, 65
    join keys with multiple properties 77
    Multi Sub-records option 85
record cache
    adding for joins 79
    Combine Records setting 88
    creating record indexes 75
    described 59
record index keys for joins 48, 67
record search configured in Developer Studio 18
record specifier property, creating 58
reference implementation, UI 53

S

search characters 60
search configuration 18
search interfaces, about 60
sort switch join 71
source data
    in delimited format 48
    loading 22
source properties
    assigning multiple mappings to 34
    mapping 55
    removing unknown 28
    specifying null mappings for 58
    standardizing 23
source property mapping
    described 27
    priority order 30
    types 29
    viewing existing 32
source property values
    defining maximum length for importing 41
    mapping to dimension values 35
source records 22
spiders 59
standardizing source properties 23
stemming 60
stop words 60
switch joins 70
system operations 19
system provisioning 18

T

tagging Endeca records 17
thesaurus entries
    configuring in Oracle Endeca Workbench 18
    introduced 60

U

UI reference implementation, using 53
unknown source properties, removing 28

W

Web service, Forge Metrics 113

X

XML syntax for dimension hierarchy 96