Endeca® Platform Services

Forge Guide

Version 6.0.3 • June 2012

ORACLE®

ENDECA

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....................25
About source property mapping....................25
About using a single property mapper....................25
About using explicit mapping....................25
Minimum configuration....................26
About mapping unwanted properties....................26
About removing source properties after mapping....................26
Types of source property mapping....................27
About adding a property mapper....................28
The Mappings editor....................30

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....................50
Property mapper....................51
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

Chapter 17: Stratify....107
About integrating Stratify taxonomies....107
How Endeca and Stratify classify unstructured documents....109
Overview of the integration process....110

Required Stratify tools....111
About preparing to integrate Stratify....112
About integrating a Stratify taxonomy....118

Part IV: Other Advanced Features....125

Chapter 18: The Forge Logging System....127
Overview of the Forge logging system....127
Log levels reference....127
About logging topics....128
The command line interface....128

Chapter 19: The Forge Metrics Web Service....131
About the Forge Metrics Web service....131
About enabling Forge metrics....132
About using Forge metrics....133
The MetricsService API....133

Appendix A: Forge Flag Reference....135
Forge flag options reference....135

Appendix B: File Formats Supported by the Document Conversion Module.143
Word processing formats....143
Spreadsheet formats....146
Graphics formats....148
Presentation formats....151
Compressed formats....152
Database formats....153
E-mail formats....154
Other formats....155

Appendix C: The Endeca Crawler....157
Overview of the Endeca Crawler....157
About installing the Endeca Crawler....157
Source documentation and Endeca records....157
Crawling errors....160
Endeca Crawler operational details....161
Security information and the Endeca Crawler....161
Full crawls vs differential crawls....161
How redundant URLs are handled....161
URL and record processing....162
Generated record properties syntax reference....163
Viewing properties created by the Endeca Crawler....170
The full crawling pipeline....170
About creating a full crawling pipeline....170
Components that support the Endeca Crawler....170
Implementing a full crawling pipeline....171
Creating a record adapter to read documents....172
Creating a record manipulator....173
Adding a RETRIEVE_URL expression....174
About converting documents to text....175
Removing document body properties....177
About modifying records with a Perl manipulator....178
Creating a spider....178
Specifying root URLS to crawl....180
Configuring URL extraction settings....181
Example syntax of URL filters....182
Specifying a record source for the spider....183
Specifying timeouts....183
About specifying proxy servers....184
About removing unnecessary records after a crawl....185
About configuring authentication....187

About configuring basic authentication.....187
About configuring HTTPS authentication.....189
About configuring client authentication.....190
About authenticating with a Microsoft Exchange server.....192
About authenticating with a proxy server.....192
About using Forge to encrypt keys and pass phrases.....192

Appendix D: Differential Crawling.....195

Overview of a differential crawling pipeline.....195
About enabling differential crawling for the spider.....195
About joining previously-crawled data.....196
About removing invalid content.....197
How the Endeca.Document.IsUnchanged property is set.....197
Caveats for differential crawling.....198
Sample differential crawl pipeline.....198
Record adapters.....200
Setting the DifferentialCrawl input record adapter.....200
Setting the PreviousCrawl input record adapter.....201
Setting the WriteRawRecords output record adapter.....201
Differential spider.....201
Setting Record Caches.....202
Record assembler.....203
Record manipulators.....204
RemoveUnchanged record manipulator.....204
RemoveFailed record manipulator.....206

Copyright and disclaimer

Preface

Oracle Endeca's Web commerce solution enables your company to deliver a personalized, consistent customer buying experience across all channels — online, in-store, mobile, or social. Whenever and wherever customers engage with your business, the Oracle Endeca Web commerce solution delivers, analyzes, and targets just the right content to just the right customer to encourage clicks and drive business results.

Oracle Endeca Commerce 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, Oracle Endeca Commerce enables businesses to help guide and influence customers in each step of their search experience. At the core of Oracle Endeca Commerce is the MDEX Engine,™ a hybrid search-analytical database specifically designed for high-performance exploration and discovery. The Endeca 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. Endeca Assembler dynamically assembles content from any resource and seamlessly combines it with results from the MDEX Engine.

Oracle Endeca Experience Manager is a single, flexible solution that enables you to create, deliver, and manage content-rich, cross-channel customer experiences. It also enables non-technical business users to deliver targeted, user-centric online experiences in a scalable way — creating always-relevant customer interactions that increase conversion rates and accelerate cross-channel sales. Non-technical users can control how, where, when, and what type of content is presented in response to any search, category selection, or facet refinement.

These components — along with additional modules for SEO, Social, and Mobile channel support — make up the core of Oracle Endeca Experience Manager, a customer experience management platform focused on delivering the most relevant, targeted, and optimized experience for every customer, at every step, across all customer touch points.

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 *Endeca 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 the Endeca Information Access Platform.

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 important information regarding Oracle Endeca software, implementation questions, product and solution help, as well as overall news and updates.

You can contact Oracle Support through Oracle's Support portal, My Oracle Support 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 the Endeca Information Access Platform. 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:

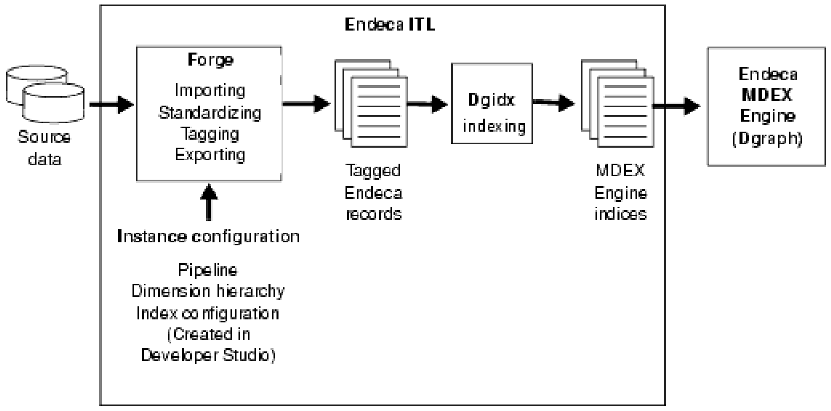
- Imports your source data
- Tags it with the dimension values used for navigating and Endeca properties used for display.
- Stores the tagged data—along with your dimension specifications and any configuration rules—as Endeca records that are ready for indexing.

- 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 provide a means for 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. Endeca 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.

At a high level, Endeca ITL development looks like this:

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 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.

Endeca Workbench

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), Aggregated MDEX Engine, 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 *Endeca Workbench Administrator's Guide* provides in-depth information about tool setup and configuration.
- The *Endeca Developer Studio Help* and the *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

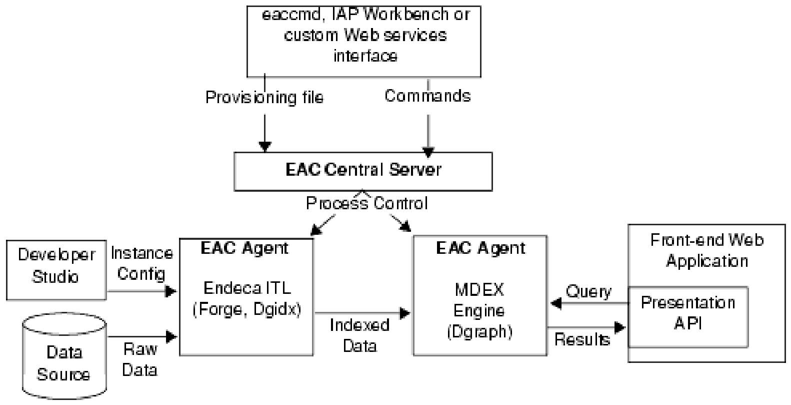
The Endeca Application Controller is the interface you use to control, manage, and monitor your Endeca implementations.

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 *Endeca EAC 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.

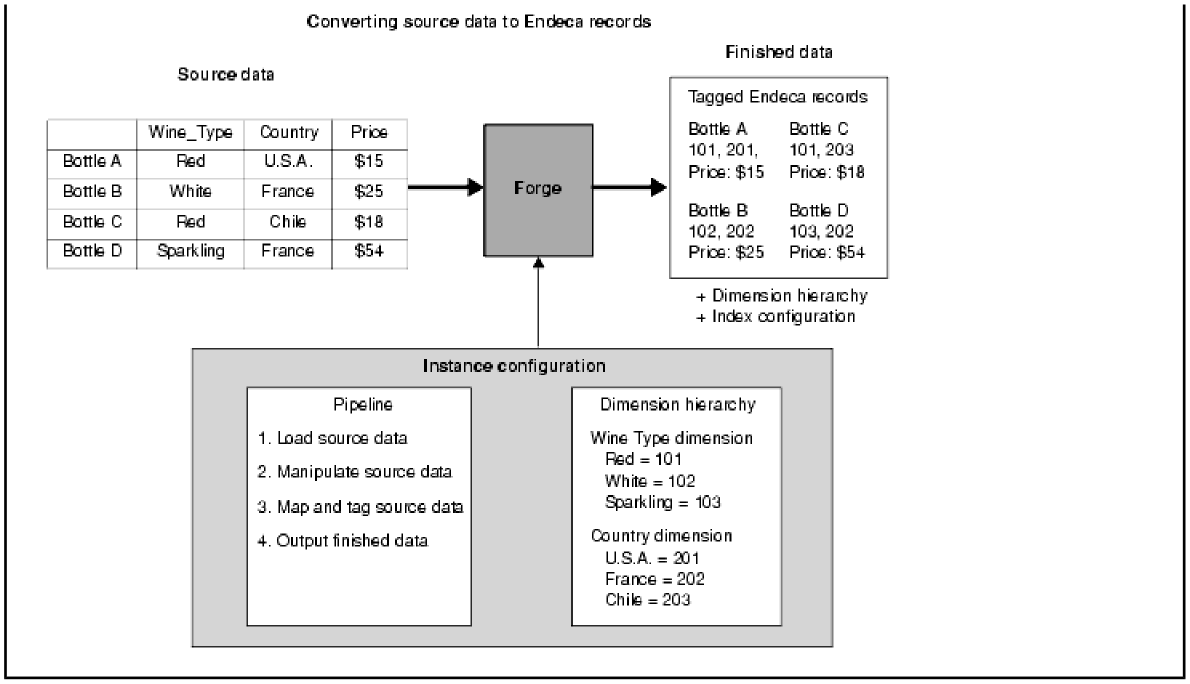
The Forge and Dgidx programs do the actual data processing, but the components you have defined in the pipeline dictate which tasks are performed and when. The Data Foundry attempts to utilize all of the hardware resources available to it, both by processing records in multiple components simultaneously,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



	Wine_Type	Country	Price
Bottle A	Red	U.S.A.	$15
Bottle B	White	France	$25
Bottle C	Red	Chile	$18
Bottle D	Sparkling	France	$54

Source data

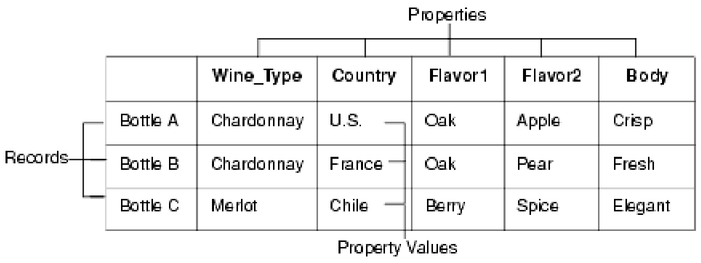
You can load source data from a variety of formats using the Content Acquisition System components.

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

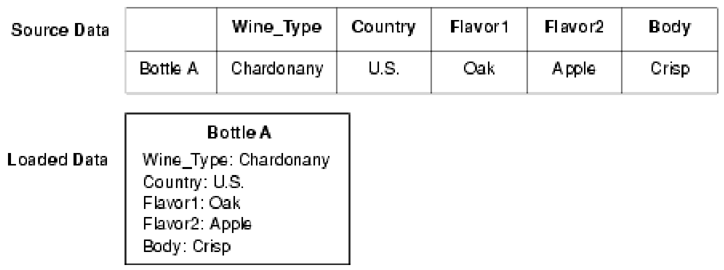
Source data may be loaded into the Data Foundry from a variety of formats. The easiest format to use is a two-dimensional format similar to the tables found in database management systems.

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.



	Wine_Type	Country	Flavor1	Flavor2	Body
Bottle A	Chardonnay	U.S.	Oak	Apple	Crisp
Bottle B	Chardonnay	France	Oak	Pear	Fresh
Bottle C	Merlot	Chile	Berry	Spice	Elegant

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:

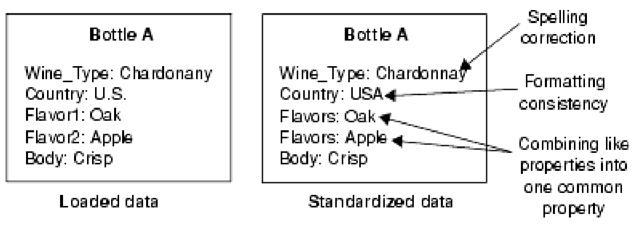
Note: The functionality described below supports limited data cleansing. If you have an existing data cleansing infrastructure, it may be more advantageous to use that facility instead.

1. Fix misspellings in your source properties and property values.
2. Edit source property values to use a consistent format (for example, USA instead of United States or U.S.).
3. 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).
4. Remove unsupported binary characters.

   Property and dimension values are not allowed to contain binary characters from the range 0x00 through 0x1F, with the exceptions of 0x09 (tab), 0x0A (newline), 0x0D (carriage return), and 0x1B (escape). For example, records sourced from 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).

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 25

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, Endeca 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 Endeca 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 27
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. Endeca 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.

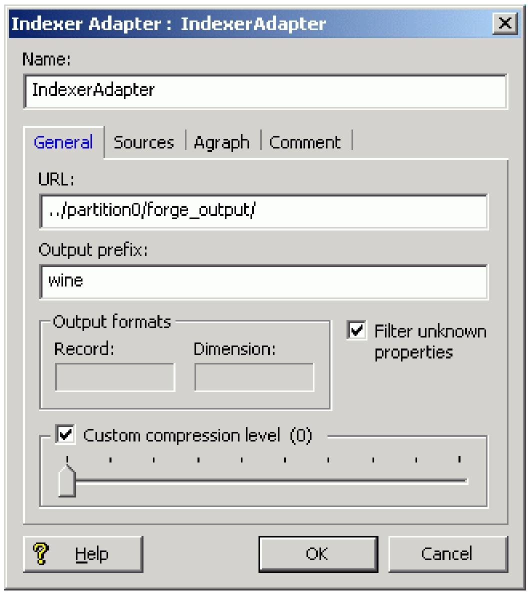
Related Links

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:


Indexer Adapter : IndexerAdapter

Name: IndexerAdapter

General | Sources | Agraph | Comment

URL: ../partition0/forge_output/

Output prefix: wine

Output formats — Record: Dimension:

☑ Filter unknown properties

☑ Custom compression level (0)

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 Endeca 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

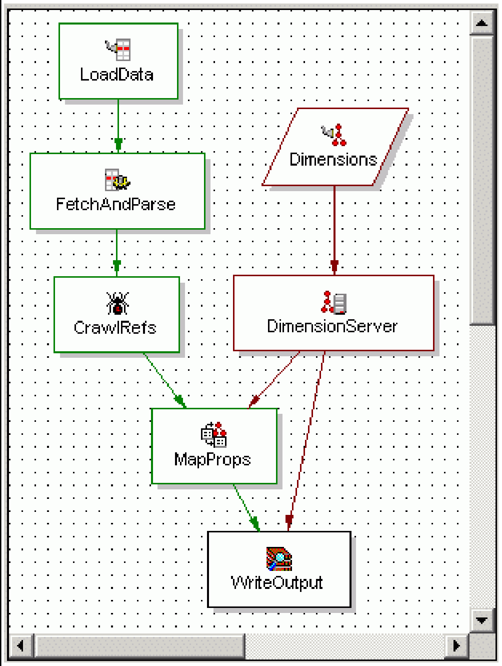
The fundamental requirements for the placement of a property mapper in the pipeline are:

- 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.

Crawler Pipeline

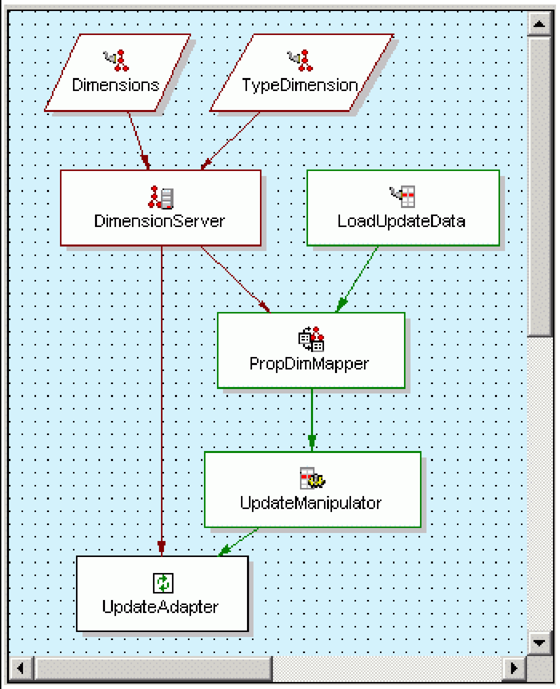
A more complicated pipeline may use a record manipulator (to clean and prepare source properties) and a spider (to crawl file systems and Web sites for documents), as in this Pipeline Diagram example:



In this crawler pipeline, the property mapper (named MapProps) uses the spider (named CrawlRefs) as its record source and the dimension server as its dimension 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:



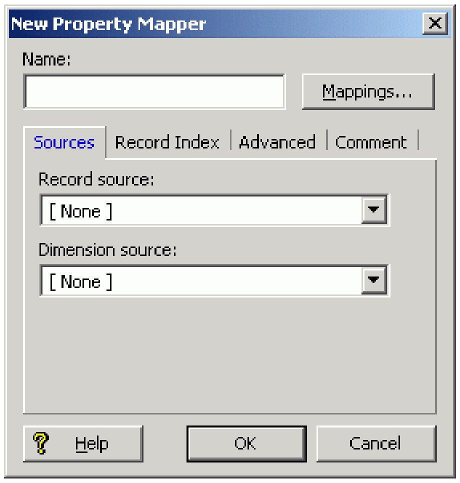
In this partial update pipeline, the property mapper (PropDimMapper) uses the record adapter (LoadUpdateData) as its record source and the dimension server as its dimension source. The record manipulator (UpdateManipulator) uses the property mapper as its record source.

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.



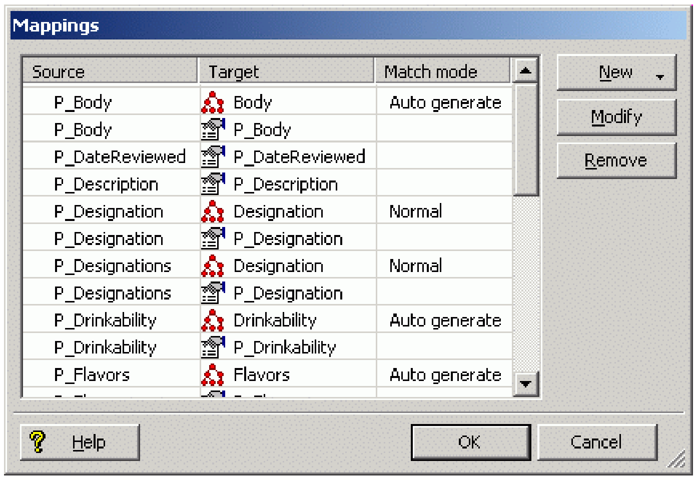
3. Enter a name for the property mapper, a record source, and a dimension source. You can leave the other settings at their defaults while developing your initial working pipeline.
4. To add the property mapper, click **OK**.

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:



Source	Target	Match mode
P_Body	Body	Auto generate
P_Body	P_Body	
P_DateReviewed	P_DateReviewed	
P_Description	P_Description	
P_Designation	Designation	Normal
P_Designation	P_Designation	
P_Designations	Designation	Normal
P_Designations	P_Designation	
P_Drinkability	Drinkability	Auto generate
P_Drinkability	P_Drinkability	
P_Flavors	Flavors	Auto generate

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

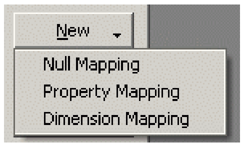
In Developer Studio, you set the type of dimension value handling, on a per mapping basis, by selecting a mode from the **Match mode** list in the **Dimension Mapping** editor, as illustrated below:

Creating new source mappings

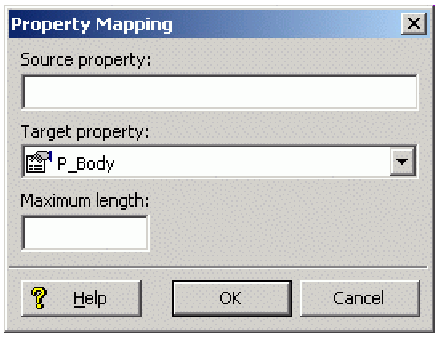
The **New** button lets you create a new source property mapping.

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 *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

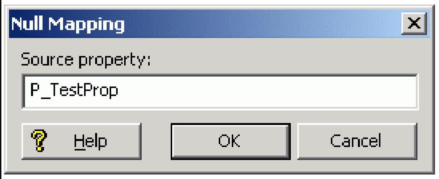
Explicit null mappings provide a means to prevent an implicit or default mapping from being formed for a particular source property. In other words, you can enable either implicit or default mapping, and then turn off mapping altogether for selected source properties using explicit null mappings.

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:


Source property: P_TestProp

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.

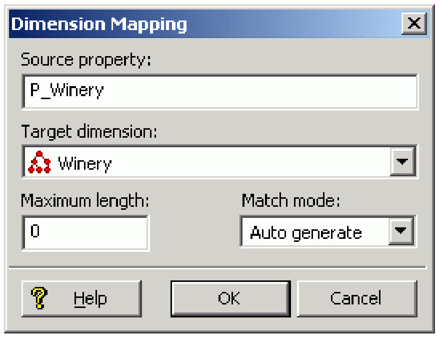
Chapter 4

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:


Dimension Mapping

Source property: P_Winery

Target dimension: Winery

Maximum length: 0

Match mode: Auto generate

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.

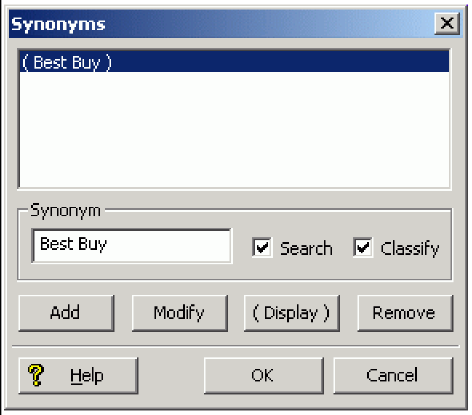
Forge assigns the IDs for any matching dimension values to the Endeca records. Any source property values that do not have matching dimension values in the dimension hierarchy are ignored.

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:


Synonyms

(Best Buy)

Synonym: Best Buy [✓] Search [✓] Classify

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 127
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 50
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

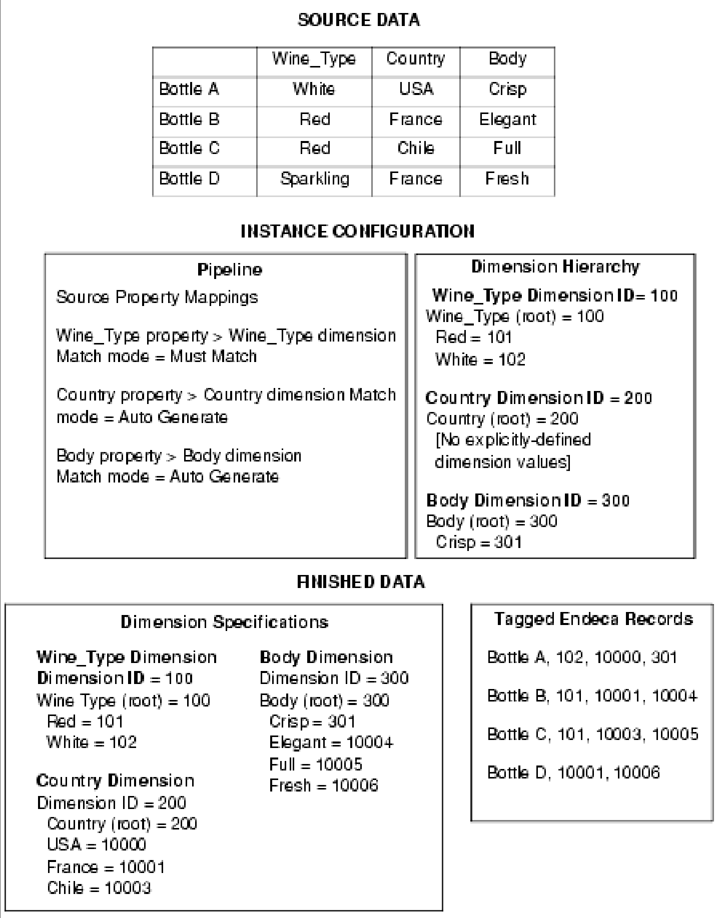
When you choose the match mode to use for generating your dimension values, keep in mind the following two rules of thumb:

- 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



SOURCE DATA

	Wine_Type	Country	Body
Bottle A	White	USA	Crisp
Bottle B	Red	France	Elegant
Bottle C	Red	Chile	Full
Bottle D	Sparkling	France	Fresh

INSTANCE CONFIGURATION

Pipeline

Source Property Mappings

Wine_Type property > Wine_Type dimension
Match mode = Must Match

Country property > Country dimension Match mode = Auto Generate

Body property > Body dimension
Match mode = Auto Generate

Dimension Hierarchy

Wine_Type Dimension ID = 100
Wine_Type (root) = 100
Red = 101
White = 102

Country Dimension ID = 200
Country (root) = 200
[No explicitly-defined dimension values]

Body Dimension ID = 300
Body (root) = 300
Crisp = 301

FINISHED DATA

Dimension Specifications

Wine_Type Dimension
Dimension ID = 100
Wine Type (root) = 100
Red = 101
White = 102

Country Dimension
Dimension ID = 200
Country (root) = 200
USA = 10000
France = 10001
Chile = 10003

Body Dimension
Dimension ID = 300
Body (root) = 300
Crisp = 301
Elegant = 10004
Full = 10005
Fresh = 10006

Tagged Endeca Records

Bottle A, 102, 10000, 301

Bottle B, 101, 10001, 10004

Bottle C, 101, 10003, 10005

Bottle D, 10001, 10006

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.

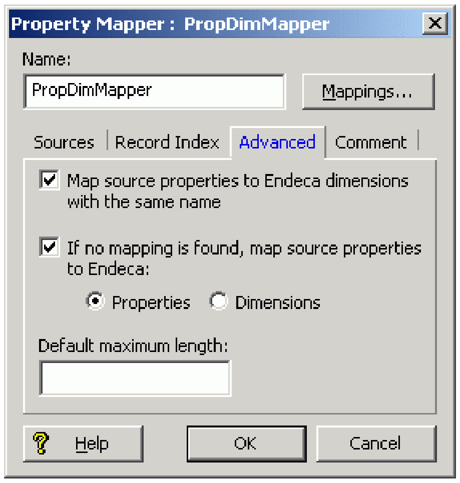
Chapter 5

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.



Name: PropDimMapper

[✓] Map source properties to Endeca dimensions with the same name

[✓] If no mapping is found, map source properties to Endeca: [● Properties] ○ Dimensions

Default maximum length:

The following sections describes these techniques.

Important: Endeca 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 127

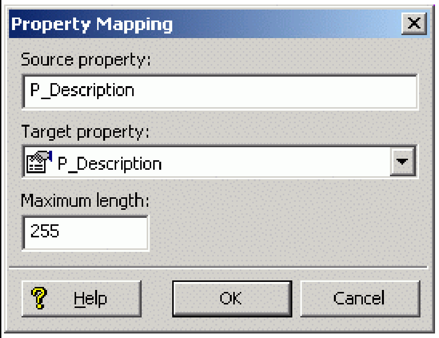
This section provides a brief introduction to the Forge logging system. Its command-line interface allows you to focus on the messages that interest you globally and by topic.

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:


Property Mapping
Source property: P_Description
Target property: P_Description
Maximum length: 255

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.

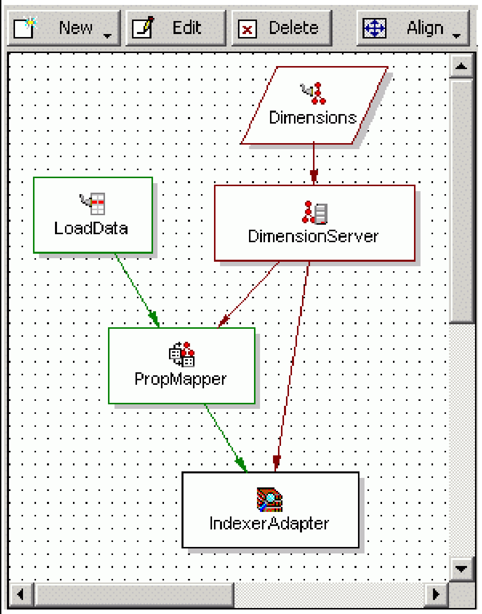
For details on adding and editing pipeline components, see the *Endeca Developer Studio Help*.

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).

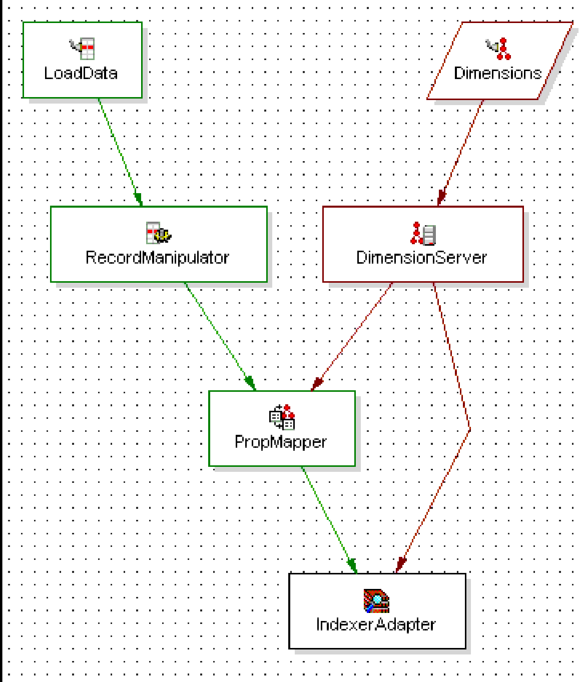
When you specify a data source within a component's editor, you are indicating which of the other components will provide data to that component. Components can have multiple data sources, such as the PropMapper component above, which has both a record source, LoadData, and a dimension source, DimensionServer.

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

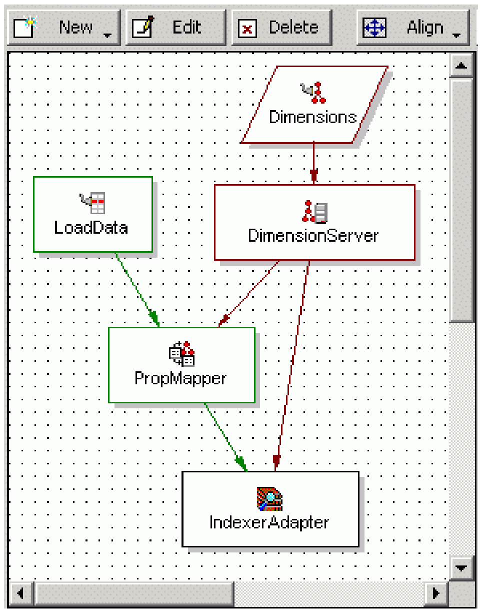
Endeca Developer Studio provides a Basic Pipeline template that helps you get started when building your pipeline from scratch. The goal of the Basic Pipeline template is to get you up and running with a working pipeline as quickly as possible. A working pipeline is defined as a pipeline that can read 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:



Endeca 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 *Endeca Developer Studio Help*.

Record adapters

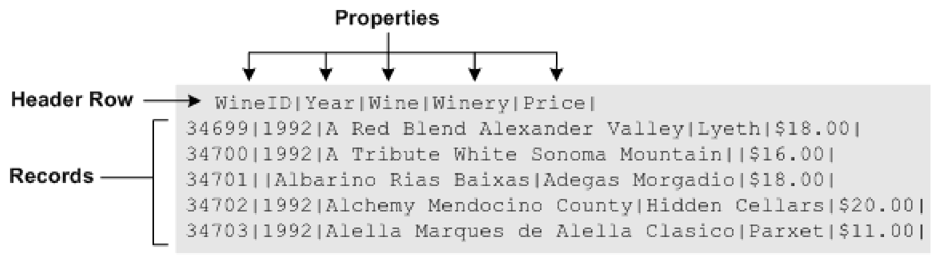
Record adapters load and save records in a variety of formats, including delimited, binary, ODBC (Windows only), JDBC, and Microsoft Exchange. Each record adapter format has its own set of attributes.

This section describes the most common type of record adapter: an input record adapter that loads data stored in delimited format. See the *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.



```
WineID|Year|Wine|Winery|Price|
34699|1992|A Red Blend Alexander Valley|Lyeth|$18.00|
34700|1992|A Tribute White Sonoma Mountain||$16.00|
34701||Albarino Rias Baixas|Adegas Morgadio|$18.00|
34702|1992|Alchemy Mendocino County|Hidden Cellars|$20.00|
34703|1992|Alella Marques de Alella Clasico|Parxet|$11.00|
```

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.

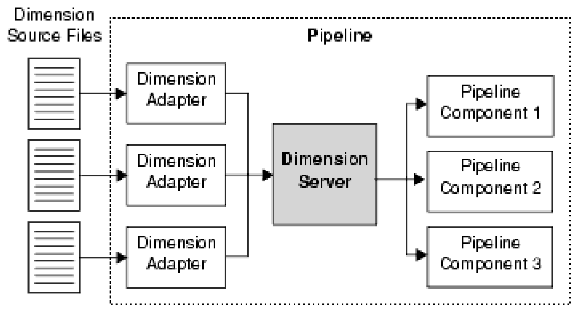
Dimension server

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

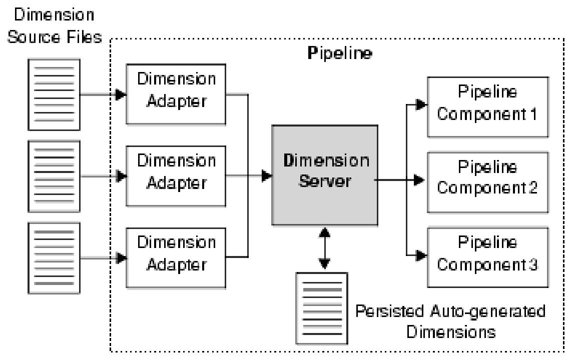
Dimension information typically follows the path outlined below:

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 25

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.

Endeca 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 25

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 *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 *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, Endeca 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, Endeca 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 *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 Endeca recommends that you use.

Related Links

Overview of Source Property Mapping on page 25

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 25

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 27
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 *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 *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 *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.

Component	Description	For More Info
Record manipulators	Provide support, such as URL extraction, for a content acquisition system, such as a crawler implementation.	See the Endeca Crawler section in this guide, and "Record Manipulators and Expressions" in the *Developer Studio Help*.
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 *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 *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*Developer Studio Help*.

Component	Description	For More Info
		See the *Endeca Workbench Help*.
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 *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 *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 *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 *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 *Developer Studio Help*. See "Working with dynamic business rules" in the *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

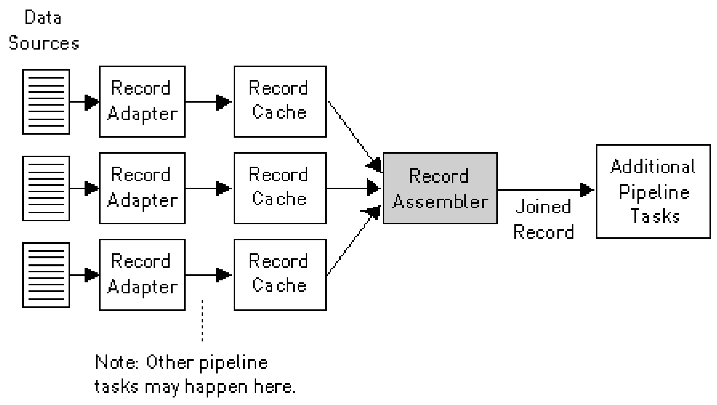
Generally, applications consist of more than one data source. For example, an application used to navigate books would have records that contain both title and author information. If the title and author source data reside in different locations, you would need to join them together to create a single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

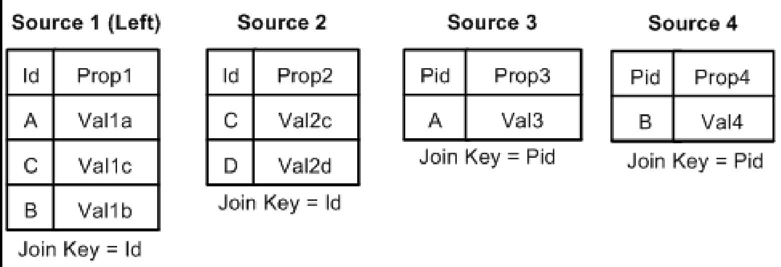
Join keys determine how records are compared by the record assembler. For each data source feeding a join, you designate one or more properties or dimensions to function as the source's join key.

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

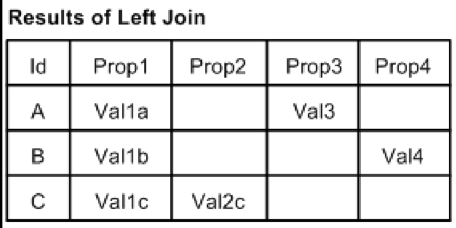
Pid	Prop4
B	Val4

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

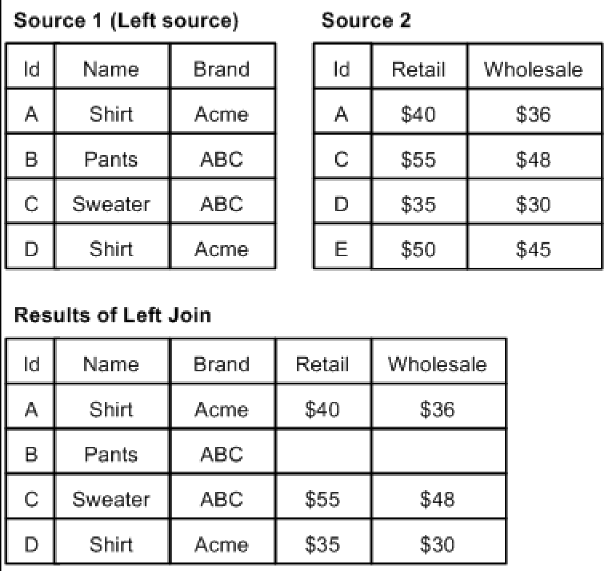
The following sections describe the join types supported by the Data Foundry. Each section provides a simple example for the join type being discussed. Note that while most of the examples use two record sources, many of the join types accept more than two sources, while other join types accept only 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

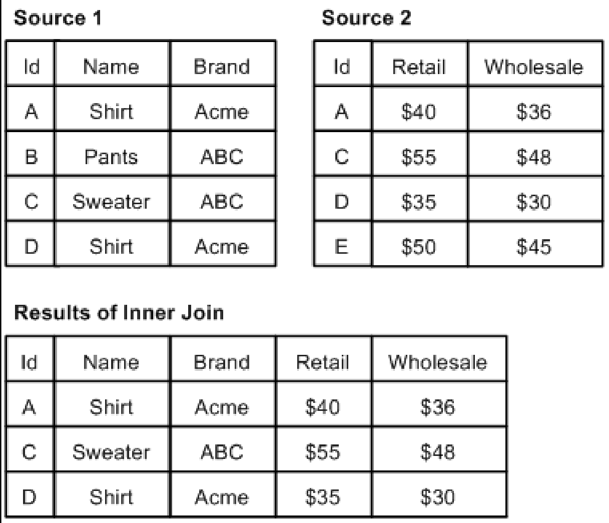
Results of Left Join

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

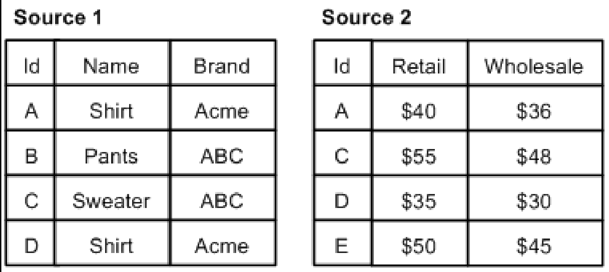
Id	Name	Brand	Retail	Wholesale
A	Shirt	Acme	$40	$36
C	Sweater	ABC	$55	$48
D	Shirt	Acme	$35	$30

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/Whole¬sale` 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

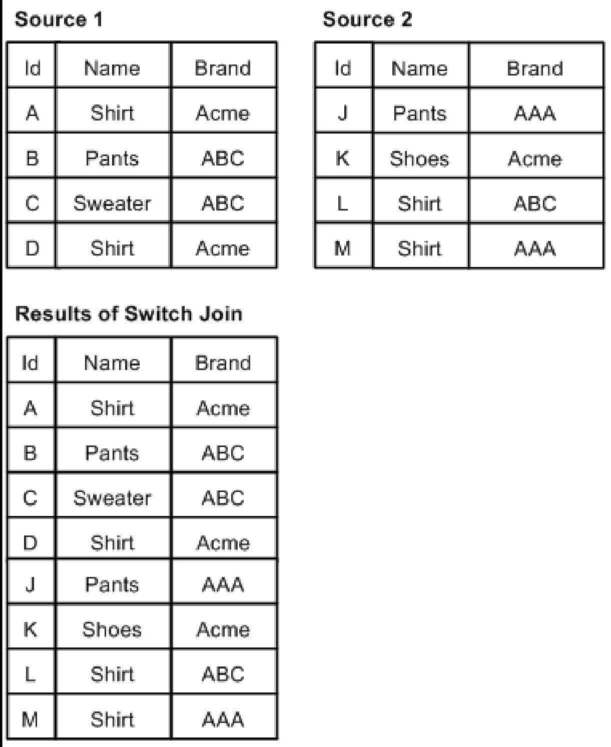
Id	Name	Brand	Retail	Wholesale
B	Pants	ABC		
E			$50	$45

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

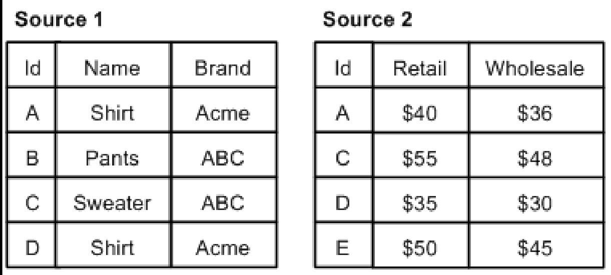
Id	Name	Brand
A	Shirt	Acme
B	Pants	ABC
C	Sweater	ABC
D	Shirt	Acme
J	Pants	AAA
K	Shoes	Acme
L	Shirt	ABC
M	Shirt	AAA

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.

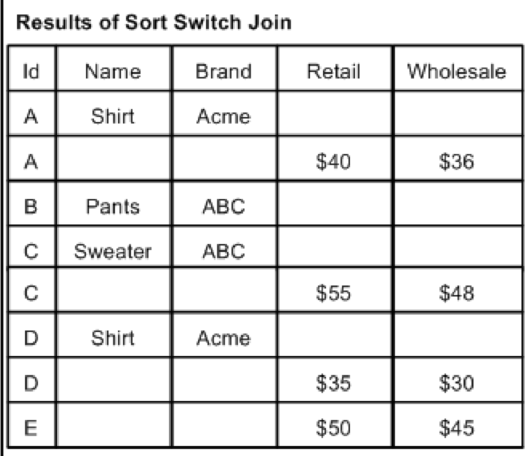


Source 1

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

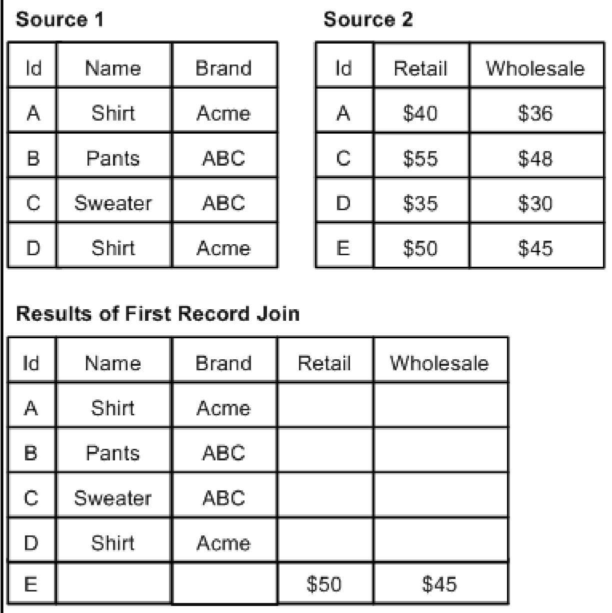
First record join

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

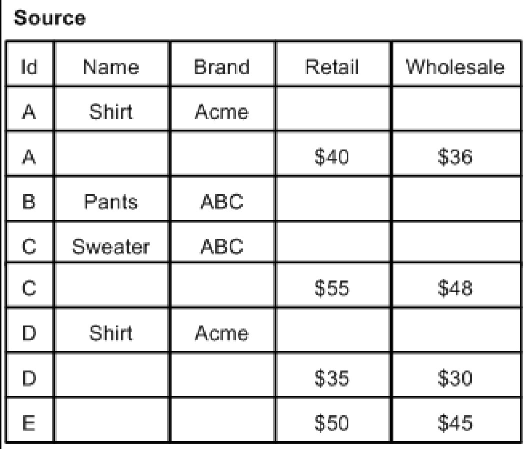
Results of First Record Join

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

Combine join

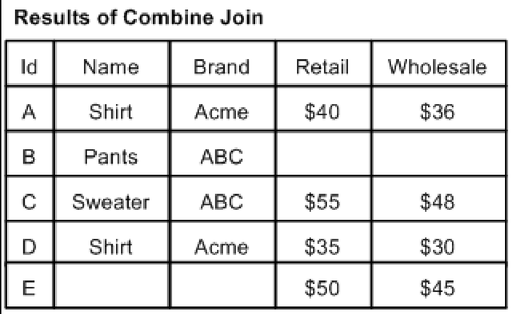
A combine join combines like records from a single data source. Combine is a pseudo-join that operates on a single source.

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.

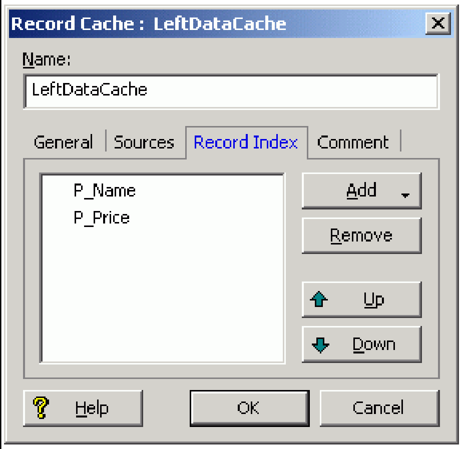
We use a record cache in this example because, with two exceptions, all data sources that feed a join must be record caches.

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`.



Name: LeftDataCache

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.

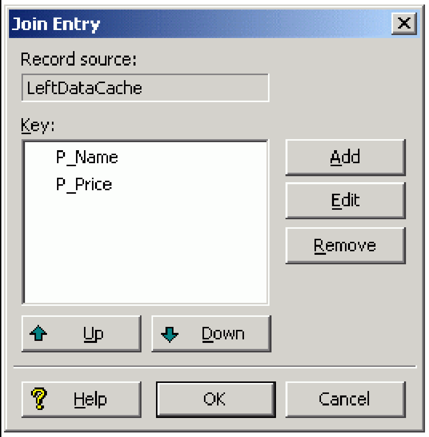
a) In the **Type** frame, do one of the following:
   - Choose **Custom Property**. Type a name for the property in the **Custom Property** text box.
   - Choose **Dimension**. Select a dimension name from the **Dimension** list.

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:


Join Entry

Record source: LeftDataCache

Key:
P_Name
P_Price

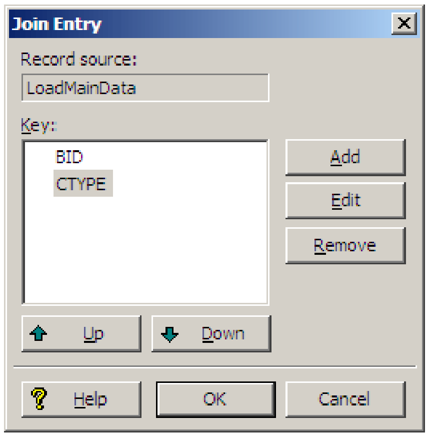
Related Links

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

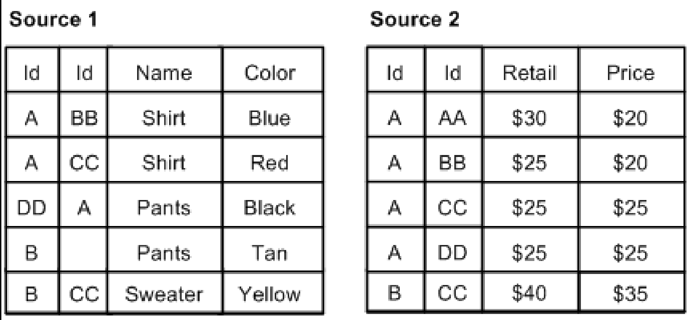
A record can have multiple property values for a given property name. For example, a record could have two values for the property `Id`.

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.



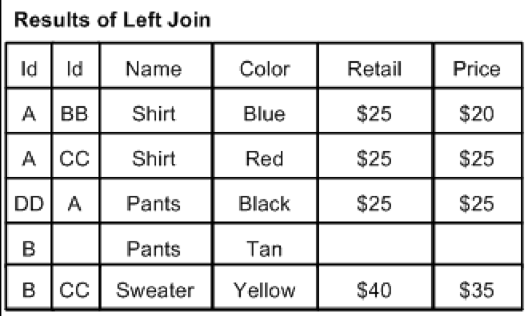
Source 1

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.

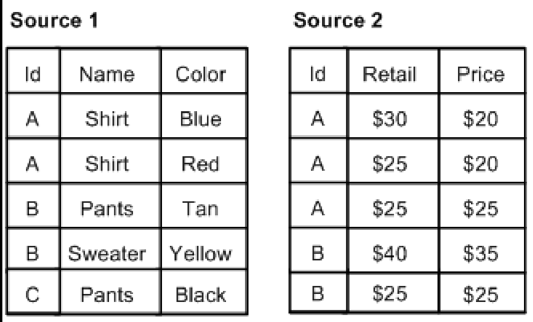
About tweaking left joins on page 85
: The **Multi Sub-records** setting (on the **Record Assembler editor Record Join** tab) changes the behavior of a left join if a record from the left source has multiple values for the join key. It is used only used with left joins. Enabling this option forces Forge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



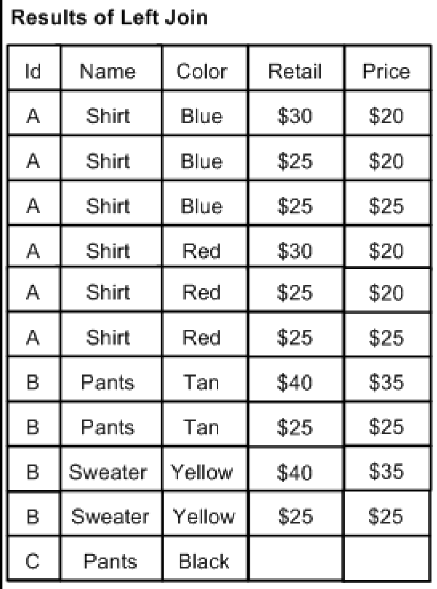
Source 1

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.

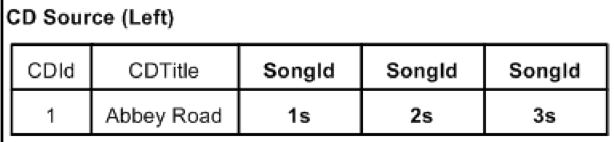
For example, if the join key is Id, a record with the values Id=1, Id=2, Id=3 produces three independent keys, one for each value. The right sources are searched for each of these keys. That is, each right source is queried for a match to the join key Id=1, a match to Id=2, and finally a match to Id=3.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

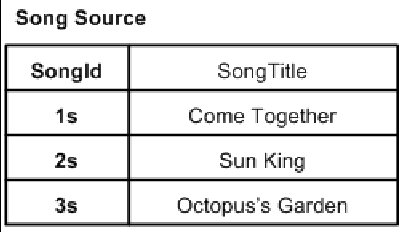
Multi sub-records

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.

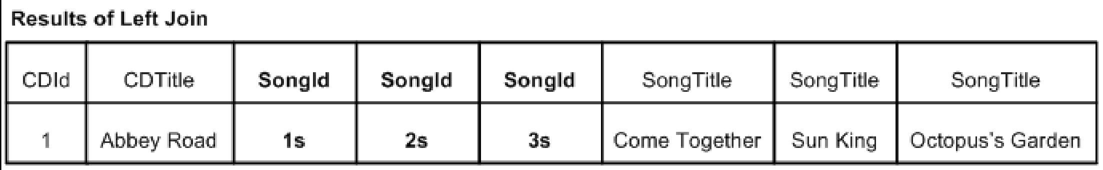
In this example, you would perform the join on the SongId, so that each song in the left table is joined appropriately with its counterpart in the right table. Note that in this example, `SongId` is the join key for all sources.


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.

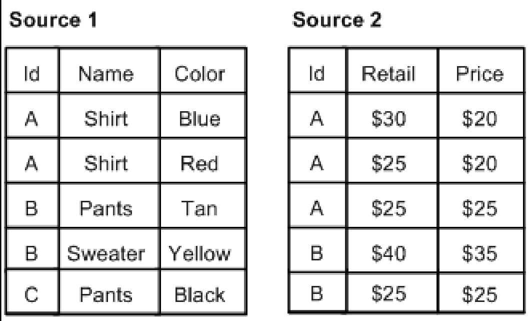
Thus, a best practice is to keep record sources with the most values per join key on the left side of joins.

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*
- *Stratify*

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 hierarchy after it is in the project?	Any changes to the dimension must be made in third-party tool. You then export the taxonomy from the tool, and Forge transforms the taxonomy and re-integrates the changes into your project.	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 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`.
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.

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 *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.

   Endeca 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 Endeca Deployment Template is available as a free download from the Endeca support site. For more information about the Endeca Deployment Template, see the *Endeca Deployment Template Usage Guide*, included in the downloadable ZIP file.
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 *Endeca IAP 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.

   Endeca 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 Endeca Deployment Template is available as a free download from the Endeca support site. For more information about the Endeca Deployment Template, see the *Endeca Deployment Template Usage Guide*, included in the downloadable ZIP file.
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.

Chapter 17

Stratify

Incorporating Stratify into your project allows you to classify unstructured source data, for example, Web pages, PDF documents, and Microsoft® Word documents, for use in Endeca Guided Navigation applications.

About integrating Stratify taxonomies

Stratify taxonomies are used to evaluate unstructured data for cases where information does not exist in a database, character-delimited file, or other easily-classifiable format.

Unstructured source data requires different processing than structured source data. Structured data, such as example, databases, CSV files, character-delimited files, fixed-width files and so on, have name/value pairs that Endeca can translate into dimensions and Endeca properties.

Unstructured data, on the other hand, is not composed of name/value pairs that Endeca can translate into dimensions and Endeca properties. For unstructured data, you have to use tools like the Stratify Discovery System™ to evaluate the content of an unstructured document and assign the document a topic based on classification logic that you configure. In an Endeca pipeline, this topic becomes a property that can be used like any other property associated with an Endeca record; for example, it can be manipulated and mapped to dimensions or Endeca properties.

In the Stratify Discovery System, you use the Stratify Taxonomy Manager™ to build a taxonomy to organize your source data, and you use the Stratify Classification Server™ to classify unstructured source data against that taxonomy. Endeca Developer Studio provides the capability to include a Stratify taxonomy and transform it to an Endeca dimension. Endeca uses the results of document classification performed in Stratify to tag Endeca records, that is your unstructured documents, with classification properties. After the records contain classification properties, you can map the properties to dimension values.

You integrate Stratify into your pipeline by adding a dimension adapter to transform the Stratify taxonomy, an Endeca Crawler to crawl unstructured documents, and a record manipulator with a `STRATIFY` expression to access the Stratify Classification Server. Before integrating Stratify into your project, you will find it helpful to read the *Endeca Forge Guide*.

Related Links

How Endeca and Stratify classify unstructured documents on page 109

Endeca components and Stratify components communicate closely during record processing to classify each unstructured source document. The process is outlined below:

Developing a Stratify taxonomy on page 110

There are two main phases to integrating a Stratify taxonomy and Stratify classification capabilities into a Developer Studio project. In the first phase, you develop a Stratify taxonomy by performing the following tasks:

Frequently used terms and concepts

Endeca concepts and terms for record organization generally correspond to Stratify terms for document organization. Below are some definitions for terms that are commonly used when discussing Stratify.

A Stratify taxonomy contains topics into which source documents are organized; an Endeca dimension contains dimension values into which Endeca records are organized. (Remember that Endeca records are based on source documents.) Both sets of concepts provide a similar framework to describe a way of hierarchically organizing information.

There are several frequently used terms in this document:

- *Structured data* is source data based on name/value pairs. Common example formats that provide this structure include databases, CSV files, character-delimited files, fixed-width files and so on. For example, this pairing might occur in source data as Color/Red, Price/$8.00, and Size/Medium. Endeca can translate these properties into dimensions and Endeca properties for use in search and Guided Navigation.
- *Unstructured data* is source data that does not have name/value pairs. Common examples of unstructured data include MS Word documents, PDF files, Web pages, and so on. If you have unstructured data, applications such as Stratify can examine the document and classify the document into topics based on logic that you configure. Endeca can use the document-to-topic classifications to create name/value properties that can be manipulated and mapped for Guided Navigation.
- *Taxonomies* provide a logical hierarchy to organize data, typically based on a theme. A taxonomy has a root and topics. Topics provide sub-grouping for the theme. For example, in a taxonomy whose root theme is Heath Care, topics may include Physical Health and Emotional Health. Respective sub-topics may include Immunizations and Depression. Source documents are organized in a taxonomy according to a classification model (described below). After you import a taxonomy into your pipeline, it becomes an Endeca dimension.
- *Taxonomy topics* provide sub-grouping organization in a Stratify taxonomy. Think of topics as nodes in a taxonomy structure. Once you import a taxonomy into your pipeline, the topics in a Stratify taxonomy become the dimension values of a single Endeca dimension.
- *Externally managed taxonomies* are logical hierarchies to organize your data that are built and managed by a third-party tool and transformed by Developer Studio into a dimension. A Stratify taxonomy is an example of an external managed taxonomy that you can create to assist in classifying unstructured source documents. The Stratify Classification Server manages a published taxonomy to perform classification tasks. If you want to modify the dimension or dimension values, you have to edit the taxonomy and taxonomy topics in Taxonomy Manager, publish it to the Stratify Classification Server, export the XML, and integrate the XML into Developer Studio.
- *Classification model* describes the classification method to organize source documents into topics. Classification models may use any of the following methods either individually or in combination: statistical, Boolean, keyword, and source rules. The Stratify Classification Server uses both the taxonomy and its classification model to classify unstructured source documents.

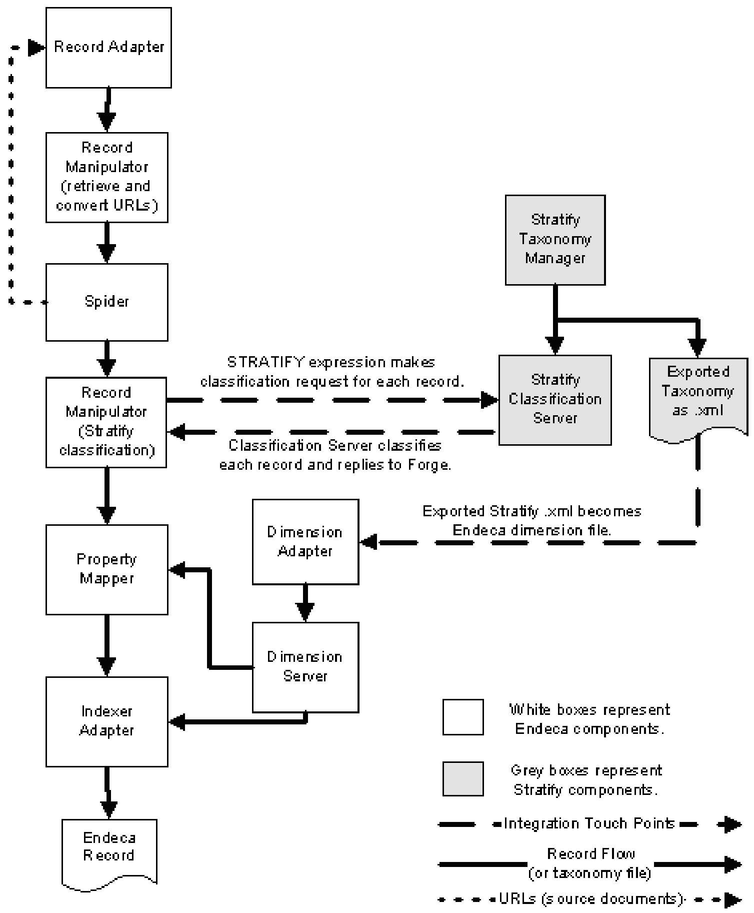
How Endeca and Stratify classify unstructured documents

Endeca components and Stratify components communicate closely during record processing to classify each unstructured source document. The process is outlined below:

A record manipulator using a `STRATIFY` expression interacts with a Stratify Classification Server to classify each record as it is processed through the pipeline. A simplified summary of the interaction is as follows: Forge crawls unstructured source documents, hands them off to Stratify to classify them, and then Forge appends classification properties to the record for the corresponding source document. You map the Stratify properties to a dimension created from the Stratify taxonomy.

The illustration below shows the interaction between Endeca components and Stratify components in greater detail. There are three kinds of flow in the diagram:

- URLs flow from the spider to the record adapter (a record adapter that uses the Document format).
- Documents flow to the indexer adapter and get turned into Endeca records.
- The Stratify taxonomy is published to the Stratify Classification Server, exported from the Taxonomy Manager as XML, and transformed in the pipeline as a dimension. Strictly speaking, this step is not part of the record processing flow. This step must be performed only once before you run a baseline update.


Record Adapter
Record Manipulator (retrieve and convert URLs)
Spider
Record Manipulator (Stratify classification)
Property Mapper
Indexer Adapter
Endeca Record
Stratify Taxonomy Manager
Stratify Classification Server
Exported Taxonomy as .xml
STRATIFY expression makes classification request for each record.
Classification Server classifies each record and replies to Forge.
Exported Stratify .xml becomes Endeca dimension file.
Dimension Adapter
Dimension Server
White boxes represent Endeca components.
Grey boxes represent Stratify components.
Integration Touch Points
Record Flow (or taxonomy file)
URLs (source documents)

When Forge executes as part of a baseline update, the flow of URLs and records is as follows:

1. The terminating component (indexer adapter) requests the next record from its record source (property mapper). At this point, none of the pipeline components between the indexer adapter and the record adapter has records to process yet.
2. When the request for the next record reaches the record adapter, the record adapter asks the spider for the next URL it is to retrieve (the first iteration through the URL processing loop, the URL is the root URL configured on the **Root URL** tab of the **Spider editor**).
3. Based on the URL that the spider provides, the record adapter creates a record containing the URL and a limited set of metadata.
4. The created record then flows down to the first record manipulator where the following takes place:
    - The document associated with the URL is fetched (using the `RETRIEVE_URL` expression).
    - Content (searchable text) is extracted from the document using the `CONVERTTOTEXT` or `PARSE_DOC` expression. Any URLs in the document are also extracted for additional crawling.
5. The record then moves to the spider where additional URLs from the document (those extracted in the record manipulator) are queued for crawling.
6. The created record then flows down to the second record manipulator where the following takes place:
    - The `STRATIFY` expression requests that the Stratify Classification Server classify each document. Forge sends the document as an attachment to a Stratify Classification Server.
    - The Stratify Classification Server examines the document, including the document's structure, and classifies it according to the classification model you developed in the Stratify Taxonomy Manager.
    - The Stratify Classification Server then replies to Forge with a classification response that indicates what properties to append to the record.
7. The property mapper performs source property to dimension and source property to Endeca property mapping.
8. The indexer adapter receives the record and writes it out to disk.

The process repeats until there are no URLs in the URL queue maintained by the spider component.

Related Links

About integrating Stratify taxonomies on page 107

Stratify taxonomies are used to evaluate unstructured data for cases where information does not exist in a database, character-delimited file, or other easily-classifiable format.

Overview of the integration process

There are two main phases to integrating a Stratify taxonomy and Stratify classification capabilities into a Developer Studio project. You perform the first phase using Stratify tools. You perform the second phase using Endeca tools.

Developing a Stratify taxonomy

There are two main phases to integrating a Stratify taxonomy and Stratify classification capabilities into a Developer Studio project. In the first phase, you develop a Stratify taxonomy by performing the following tasks:

1. Build and validate the taxonomy, including its classification model.
2. Publish the taxonomy and classification model to the Stratify Classification Server.

3. Export the taxonomy as XML.

After developing your taxonomy, you must create a pipeline to incorporate it.

Related Links

About developing a Stratify taxonomy on page 112
Before you can build an integrated project you first need to develop your Stratify taxonomy. Development includes building the taxonomy itself, publishing it and its classification model to the Stratify Classification Server, and then exporting the taxonomy as XML.

About integrating Stratify taxonomies on page 107
Stratify taxonomies are used to evaluate unstructured data for cases where information does not exist in a database, character-delimited file, or other easily-classifiable format.

Creating a pipeline to incorporate Stratify

There are two main phases to integrating a Stratify taxonomy and Stratify classification capabilities into a Developer Studio project. In the second phase, you create a Developer Studio pipeline to incorporate the Stratify taxonomy and access the Stratify Classification Server by performing the following tasks:

Before creating your Developer Studio pipeline, you must develop a Stratify taxonomy.

1. Create a pipeline for an Endeca Crawler.
2. Create a record manipulator to classify documents with the Stratify Classification Server.
3. Add a property mapper and indexer adapter.
4. Integrate the Stratify taxonomy.
5. Run a baseline update to transform the taxonomy.
6. Load a dimension and its dimension values.
7. Map the dimension in a property mapper.
8. Run another baseline update.

Related Links

About creating a pipeline to incorporate Stratify on page 114
A Developer Studio project uses the Endeca Crawler to crawl unstructured source documents. Next, a record manipulator accesses the Stratify Classification Sever, which classifies the Endeca record for each source document.

Required Stratify tools

In addition to installing and configuring the Endeca Information Access Platform, Endeca Developer Studio, and the Endeca Application Controller, you must also install and configure a Stratify Discovery System 3.0 before you begin the integration process.

At a minimum, this installation requires the following two components of the Stratify Discovery System:

- Stratify Taxonomy Manager—manages the total taxonomy lifecycle, using a combination of automated technologies, advanced analytics and human review to create, define, test, publish and refine taxonomies.
- Stratify Classification Server—stores the taxonomy and uses it to classify documents in response to requests from Forge.

You do not need to use either Stratify Analytics or Stratify Notification Server as part of the integration.

About preparing to integrate Stratify

In order to run Stratify on your data, you need to set up a taxonomy for it to work with, and create a pipeline that includes Stratify classification abilities along with your taxonomy.

About developing a Stratify taxonomy

Before you can build an integrated project you first need to develop your Stratify taxonomy. Development includes building the taxonomy itself, publishing it and its classification model to the Stratify Classification Server, and then exporting the taxonomy as XML.

You perform all of the steps involved in building and exporting a taxonomy using Stratify tools. This document describes the high-level tasks of building a taxonomy and some of the detailed procedures for exporting a taxonomy. Refer to the *User's Guide for Stratify Discovery System 3.0* for additional details about how to perform each task.

Related Links

Developing a Stratify taxonomy on page 110

There are two main phases to integrating a Stratify taxonomy and Stratify classification capabilities into a Developer Studio project. In the first phase, you develop a Stratify taxonomy by performing the following tasks:

Building a taxonomy

The following steps outline the high-level process to build and publish a Stratify taxonomy using the Taxonomy Manager.

For complete documentation of these steps, see the *User's Guide for Stratify Discovery System 3.0*.

To build and publish your taxonomy:

Note: When creating a Stratify taxonomy, the name you provide becomes the name of the root dimension value and the names of the taxonomy's topics become the names of the dimension values beneath the root. Also, if you are building multiple Stratify taxonomies, the identifier for each Stratify topic must be unique across all taxonomy files.

1. Build a taxonomy using the **Taxonomy Manager**. This step includes creating the taxonomy in any of the following ways:
   - manually
   - generating automatically
   - importing from another external source

   This step may also include editing the taxonomy as necessary during development.
2. Define the classification model for the taxonomy. You can employ any of the four Stratify classification models:
   - statistical
   - keyword
   - source rules

- Boolean

3. Compile the model of the taxonomy, if you created a statistical classification model. This step is not necessary for the other models.
4. Test the taxonomy's classification model.
5. Publish the taxonomy to the Stratify Classification Server and version any other published taxonomy if necessary.

After building and publishing your taxonomy, you must export it as an XML file.

Related Links

Exporting a taxonomy on page 113

After you have built and published a taxonomy to the Stratify Classification Server, you must export the taxonomy as an XML file. You will later include this XML file in Developer Studio and transform it to an Endeca dimension to use during mapping.

Exporting a taxonomy

After you have built and published a taxonomy to the Stratify Classification Server, you must export the taxonomy as an XML file. You will later include this XML file in Developer Studio and transform it to an Endeca dimension to use during mapping.

You must build and publish your taxonomy before exporting it.

To export a taxonomy:

Note: This procedure describes exporting a taxonomy from the Stratify Taxonomy Manager. If you have additional questions about the procedure, see the *User's Guide for Stratify Discovery System 3.0*.

1. Start **Stratify Taxonomy Manager**.
2. From the **View** menu, select **Taxonomies**.
3. In the **Select Taxonomies to View** dialog box, check the taxonomy that you want to export.
4. Click **OK**.
5. In the taxonomy tree, right-click the published taxonomy.
6. Select **Export** from the submenu.
7. In the **Export** dialog box, perform the following steps:
   a) In the **Topic** field, confirm that you selected the correct taxonomy.
   b) In the **To File** field, provide a path and file name for the taxonomy you want to export.
   c) Select the **Export Children** option.
   d) Under **Advanced Options**, select **All**.
8. Click **OK**.
9. If necessary, repeat steps 2 through 8 to export multiple taxonomies.

Related Links

Building a taxonomy on page 112

The following steps outline the high-level process to build and publish a Stratify taxonomy using the Taxonomy Manager.

About creating a pipeline to incorporate Stratify

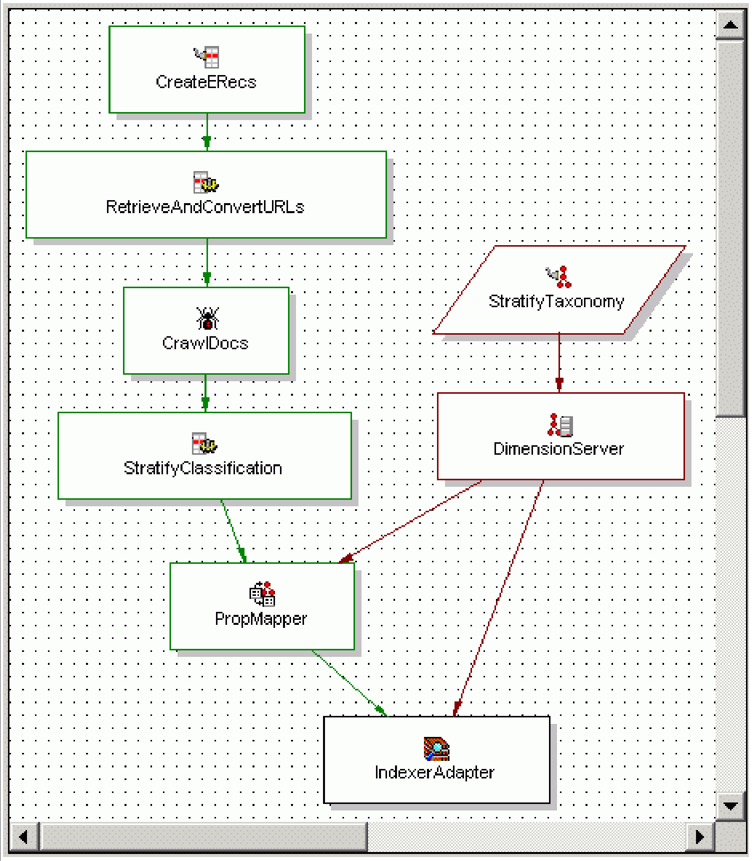
A Developer Studio project uses the Endeca Crawler to crawl unstructured source documents. Next, a record manipulator accesses the Stratify Classification Sever, which classifies the Endeca record for each source document.

There are two specific additions to a typical Endeca Crawler pipeline that allow you to incorporate Stratify taxonomies and classification capabilities:

- A dimension adapter is required to read in and transform the Stratify taxonomy.
- A `STRATIFY` expression is required after the `RETRIEVE_URL` and text extraction expressions (either `PARSE_DOC` or `CONVERTTOTEXT`) to communicate with the Stratify Classification Server and classify documents.

Because of the similarity between an Endeca Crawler pipeline and an Endeca Crawler pipeline with Stratify, this document relies on cross-references to the “Implementing the Endeca Crawler” appendix. In cases where the two crawler pipelines are exactly the same, this document references the crawler procedure. In cases where a typical crawler pipeline differs with a crawler pipeline including Stratify, this document indicates those differences.

This diagram shows an example pipeline similar to the one you create in this document:



Related Links

Creating a pipeline to incorporate Stratify on page 111

There are two main phases to integrating a Stratify taxonomy and Stratify classification capabilities into a Developer Studio project. In the second phase, you create a Developer

Studio pipeline to incorporate the Stratify taxonomy and access the Stratify Classification Server by performing the following tasks:

Creating an Endeca Crawler

Begin your pipeline by creating an Endeca Crawler that crawls your unstructured documents. Most of the steps to create a crawler pipeline that includes Stratify are common to a typical crawler pipeline. The pipeline differs in the components that follow the spider component.

To create a Endeca Crawler pipeline:

1. Create a record adapter to read source documents.

   For details, see "Creating a record adapter to read documents" in the chapter titled "The Endeca Crawler" in this guide.
2. Create a record manipulator.

   For this task and the following bullet items, see "Creating a record manipulator in this guide:
   a) Add a `RETRIEVE_URL` expression.
   b) Convert documents to text.
   c) (Optional) Identify the language of the documents.
   d) (Optional, Recommended) Remove document body properties.
3. Create a spider component.

   See "Creating a spider" in the chapter titled "The Endeca Crawler".

After creating your Endeca Crawler pipeline, proceed to configure your Stratify Classification Server.

Related Links

Configuring a Stratify Classification Server on page 115
By adding the `STRATIFY` expression to a record manipulator, you identify it as a Stratify Classification Server.

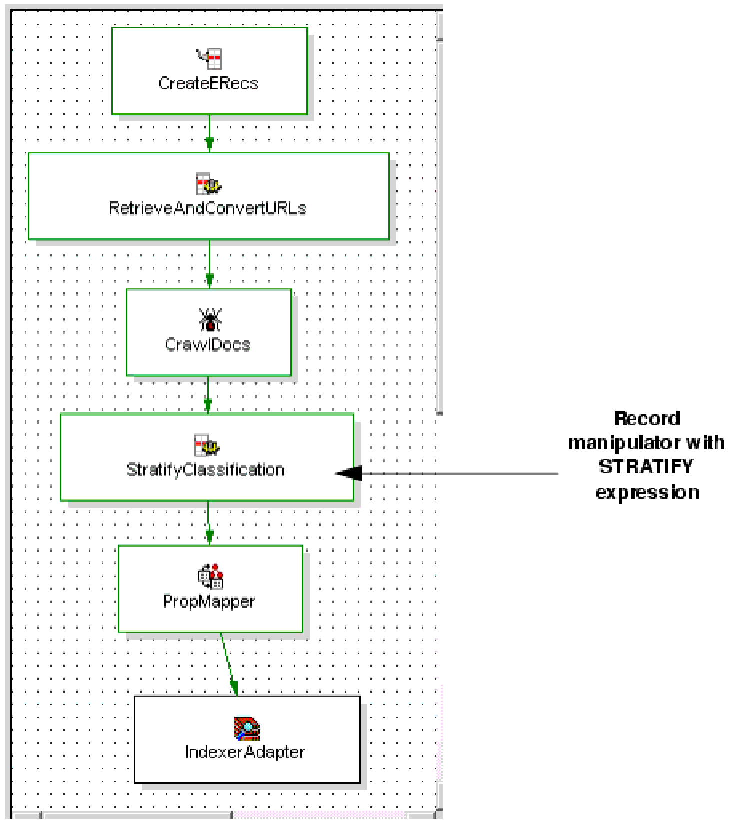
Configuring a Stratify Classification Server

By adding the `STRATIFY` expression to a record manipulator, you identify it as a Stratify Classification Server.

Before setting up your Stratify Classification Server, you need to create an Endeca Crawler pipeline.

A `STRATIFY` expression is required after the `RETRIEVE_URL` and text extraction expressions (either `PARSE_DOC` or `CONVERTTOTEXT`). The `STRATIFY` expression identifies a Stratify Classification Server that classifies the unstructured document associated with an Endeca record.

For the sake of pipeline clarity, Endeca recommends that you add the `STRATIFY` expression in its own record manipulator that follows the spider component. The recommended position of a record manipulator containing the `STRATIFY` expression is after the spider component and before the property mapper:



If you have more than one Stratify Classification Server in your environment, then you need one `STRATIFY` expression to specify the host, port, hierarchy ID, and other information for each server. Typically, a single taxonomy is published to a single Stratify Classification Server.

Note: You can publish multiple taxonomies to a single Stratify Classification Server if you prefer.

To add a `STRATIFY` expression to a record manipulator:

1. In the **Project** tab of Developer Studio, double-click **Pipeline Diagram**.
2. In the **Pipeline Diagram** editor, choose **New** > **Record** > **Manipulator**.
   The **Record Manipulator** editor displays.
3. Click **OK**.
4. Double-click the record manipulator.
5. Add the required `STRATIFY` expressions from the following list:

Expression	Description
`STRATIFY_HOST`	The machine name or IP address of the Stratify Classification Server.
`STRATIFY_PORT`	The port on which the Stratify Classification Server listens for requests from Forge.
`HIERARCHY_ID`	The identifier of a Stratify classification model.
`IDENTIFIER_PROP_NAME`	The Endeca identifier for the record being processed. The default is `Endeca.Identifier`.

Expression	Description
`BODY_PROP_NAME`	The property that the Stratify Classification Server examines to classify the document. The default property is `Endeca.Document.Body`. You can provide either `Endeca.Document.Body` or `Endeca.Document.Text`. However, specifying `Endeca.Document.Body` provides better classification because Forge can send the document to the Stratify Classification Server as an attachment, and the Stratify Classification Server can use the attachment to determine structural information of the document that aids in classification. If you specify `Endeca.Document.Text`, Forge sends the converted text of the document without any of its structural information.

Note: It is not necessary to provide attribute values for the `LABEL` or `URL` attributes.

To determine the `VALUE` of `HIERARCHY_ID`:

a) Navigate to the working directory of the Stratify Classification Server that contains your classification model and taxonomy files. This directory is typically located at `<Stratify Install Directory>\ClassificationServer\ClassificationServer\ClassificationServerWorkDir\Taxonomy-N`, where *N* is the number of the directory that contains the classification model you want to use with your Endeca project. (Your environment may have multiple `\Taxonomy-N` directories each containing different classification model and taxonomy files).

b) Note the number at the end of the of `\Taxonomy-N` directory. This number is the value of `HIERARCHY_ID`. For example, if the classification model you want to use is stored in `...\Taxonomy-2`, then `HIERARCHY_ID` should have `VALUE="2"`. If you published more than one taxonomy to your Stratify Classification Server, include a `HIERARCHY_ID` node for each taxonomy.

For general information about how to create expressions, see the *Endeca Data Foundry Expression Reference*.

6. If necessary, add additional `STRATIFY` expressions for each Stratify Classification Server in your environment.

Related Links

Creating an Endeca Crawler on page 115

Begin your pipeline by creating an Endeca Crawler that crawls your unstructured documents. Most of the steps to create a crawler pipeline that includes Stratify are common to a typical crawler pipeline. The pipeline differs in the components that follow the spider component.

The Stratify classification process

When you run a baseline update, here is how the classification process takes place:

1. For each record that passes through the record manipulator, the `STRATIFY` expression requests that the Stratify Classification Server classify the document indicated by `Endeca.Document.Body`.
2. Forge sends the document as an attachment to the Stratify Classification Server.
3. The Stratify Classification Server examines the document, including the document's structure, and classifies it according to the classification model you developed in Taxonomy Manager. (You indicate the classification model in the `HIERARCHY_ID` expression node).

4. The Classification Server then replies to Forge with a classification response that indicates what property values to append to the record.

The `STRATIFY` expression generates at least the following properties to append to each record:

- `Endeca.Stratify.Topic.HID<`*`hierarchy ID`*`>=<`*`topic ID`*`>`

  This property corresponds to the ID value of a topic in your published Stratify taxonomy. Each topic in your taxonomy has an ID value assigned by Stratify. The value of *<hierarchy ID>* corresponds to your `HIERARCHY_ID` expression node. For example, if an Eating Disorders topic has a topic ID of 2097222 in a health care taxonomy whose hierarchy ID is 15, then the Endeca property is `Endeca.Stratify.Topic.HID15="2097222"`.

- `Endeca.Stratify.Topic.Topic.Name.HID<`*`hierarchy ID`*`>.TID<`*`topic ID`*`>=<`*`topic name`*`>`

  This property corresponds to a topic name from your published Stratify taxonomy for its corresponding topic ID. For example, for the Eating Disorders topic in the health care taxonomy mentioned earlier, this property is `Endeca.Stratify.Topic.Name.HID15.TID2097222="Eat¬ ing Disorders"`.

- `Endeca.Stratify.Topic.Score.HID<`*`hierarchy ID`*`>.TID<`*`topic ID`*`>=<`*`score`*`>`

  This property indicates classification score between an unstructured document and the topic it has been classified into. The value of *<score>* is a percentage expressed as a value between zero and one. Zero indicates the lowest classification score (0%), and one indicates the highest score (100%). You can use this property to remove records from your application that have a low score for classification matching, for example, `Endeca.Stratify.Top¬ ic.Score.HID15.TID2097222="0.719380021095276"`.

Adding a property mapper and indexer adapter

Although the pipeline does not have any external dimensions available for mapping (because you have not yet integrated the Stratify taxonomy), you still need to add a property mapper in the pipeline to successfully run the first baseline update.

You add an empty property mapper to your pipeline after building and publishing your taxonomy and configuring your Stratify Classification Server.

Note: There are no special configuration requirements for the indexer adapter in a pipeline that integrates Stratify. See the *Endeca Developer Studio Help* for more information about creating a property mapper and indexer adapter.

1. Create an empty property mapper and select the upstream record manipulator as its record source.
2. Add an indexer adapter to your project as you would for a basic pipeline.

You will come back to this property mapper later in the process to add dimension mapping for your external taxonomy.

About integrating a Stratify taxonomy

To integrate a Stratify taxonomy into your pipeline, you must create a new dimension adapter and run a baseline update to transform the XML and load the dimension values. A second update is required to process and tag your Endeca records after mapping the source values.

Integrating a stratify taxonomy

You use a dimension adapter to read in the Stratify taxonomy XML and transform it to Endeca-compatible XML. If necessary, you can include and transform multiple Stratify taxonomies by using a single dimension adapter for each taxonomy file.

After building and exporting your Stratify taxonomy and creating a pipeline for it, you can integrate the two.

In the **Dimension Adapter editor**, you specify the XML file of the taxonomy that Forge transforms. To perform the taxonomy transformation, you have to run a baseline update. When the update runs, Forge transforms the taxonomy into an externally managed Endeca dimension that you can load in the **Dimensions** view.

Note: After you integrate the Stratify taxonomy, a newly-transformed dimension is very similar to a dimension created using Developer Studio. However, because the dimension is externally managed by Stratify, you cannot edit the dimension or its dimension values using Developer Studio. If you want to modify an externally managed dimension, you have to repeat the process of integrating Stratify into your pipeline.

To integrate a Stratify 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 **Stratify**.
   c) In the **URL** field, enter the path to the source taxonomy file that you previously exported. This path must be relative to the location of your project's `Pipeline.epx` file.
   d) If you want Forge to generate an error if the file does not exist or is empty, check **Require Data**.
5. Click **OK**.
6. Select **File** > **Save**.
7. If necessary, repeat steps 2 through 6 to include additional Stratify taxonomies.
8. Create a dimension server to provide a single point of reference for other pipeline components to access dimension information from multiple dimension adapters.

   See the *Endeca Developer Studio Help* for more information.

Running the first baseline update

In order to transform the source XML from your Stratify taxonomy into an Endeca dimension, you have to run a baseline update. Running the update allows Forge to transform the taxonomy and generate a `dimensions.xml` file.

After you have created and integrated your Stratify taxonomy, you run an initial baseline update to transform your Stratify taxonomy XML.

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.) You can specify the flag in the **Forge** section of the **EAC Admin Console** page in Endeca Workbench.

To transform the taxonomy:

1. Place a copy of your source taxonomy file into the directory you specified in Endeca Workbench, for the **Incoming Directory** field of the **EAC Admin Console** page.
2. Ensure you have sent the latest instance configuration from Developer Studio to Endeca Workbench.
3. In Endeca Workbench, run your baseline update script from the **EAC Admin Console** page.

After you run the update, you can then load the dimension. When you load the dimension, Developer Studio reads the `dimensions.xml` file and displays the dimension values in the **Dimension Values** view.

Loading a dimension and its dimension values

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

Before loading dimension values, you must transform your taxonomy into an Endeca dimension by running an initial baseline update.

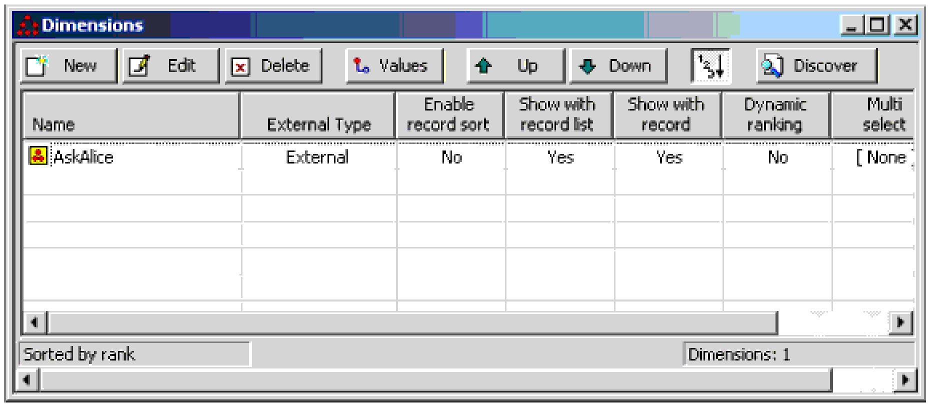
Rather than choose **New** (as you would to create a standard dimension), you instead click **Discover** in the **Dimensions** view to see an externally managed dimension. Developer Studio discovers the dimension by reading in the `dimensions.xml` file created when you ran the first baseline update. Next, you click **Load** in the **Dimension Values** view and Forge reads in dimension value information from the `dimensions.xml` file and presents the dimension values in the **Dimension Values** view.

The core characteristics of an externally managed dimension and its dimension values are not modifiable in the **Dimension Values** view. These features appear as unavailable, or Developer Studio displays a message indicating what actions are possible. You can, however, modify whether dimension values are inert or collapsible.

To load a dimension based on a Stratify taxonomy:

1. In the **Project** tab of Developer Studio, double-click **Dimensions**.
   The **Dimensions** view displays.
2. Click the **Discover** button to display any externally managed dimensions that you imported.
   The dimension or dimensions appear in the **Dimensions view** with the value of their **External Type** column set to `External`.
3. In the **Dimensions** view, select the dimension based on a Stratify taxonomy and click **Values**.
   The **Dimension** view appears with the root dimension displayed.
4. Select the root dimension and click **Load**.
   The dimension values appear in the **Dimension Values** view with the value of their **External Type** column set to `External`.
5. Repeat steps 3-4 for an additional dimensions based on a Stratify taxonomy that you integrated into your project

Here is an example of a **Dimensions** view with one dimension based on a Stratify taxonomy:

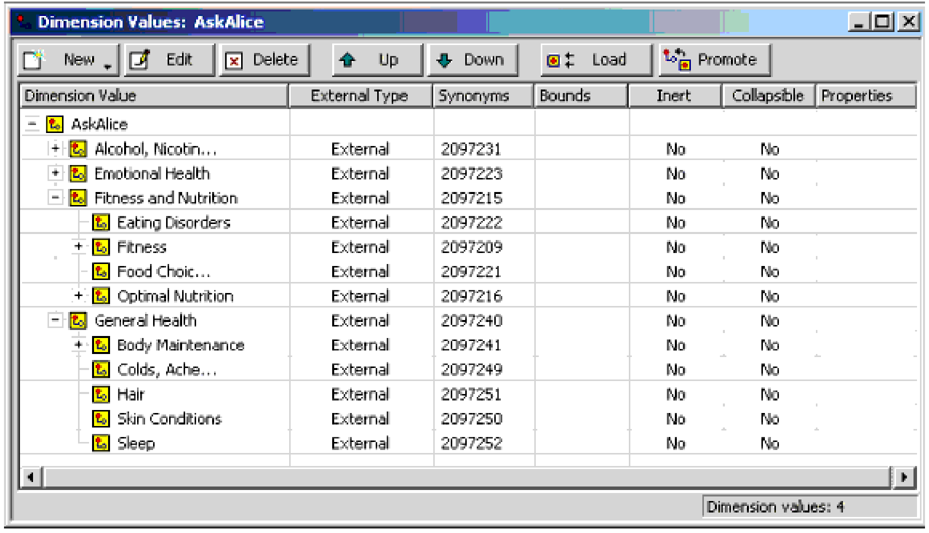


Name	External Type	Enable record sort	Show with record list	Show with record	Dynamic ranking	Multi select
AskAlice	External	No	Yes	Yes	No	[None

Sorted by rank

Dimensions: 1

Here is an example of a **Dimension Values** view with the loaded dimension values:



Dimension Value	External Type	Synonyms	Bounds	Inert	Collapsible	Properties
AskAlice						
Alcohol, Nicotin...	External	2097231		No	No	
Emotional Health	External	2097223		No	No	
Fitness and Nutrition	External	2097215		No	No	
Eating Disorders	External	2097222		No	No	
Fitness	External	2097209		No	No	
Food Choic...	External	2097221		No	No	
Optimal Nutrition	External	2097216		No	No	
General Health	External	2097240		No	No	
Body Maintenance	External	2097241		No	No	
Colds, Ache...	External	2097249		No	No	
Hair	External	2097251		No	No	
Skin Conditions	External	2097250		No	No	
Sleep	External	2097252		No	No	

Dimension values: 4

Related Links

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

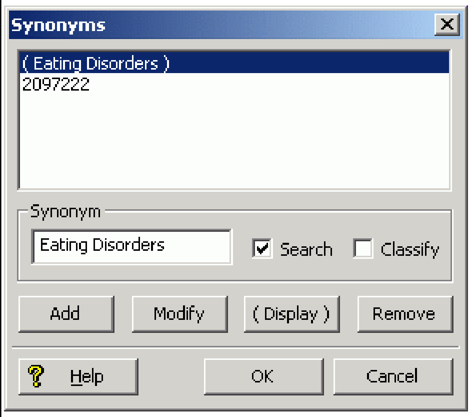
If you want to modify an externally-managed taxonomy and replace it with a newer version, you have to revise the taxonomy using the third-party tool that created it, and then repeat the process of incorporating the externally-managed taxonomy into your pipeline.

About synonym values and dimension values

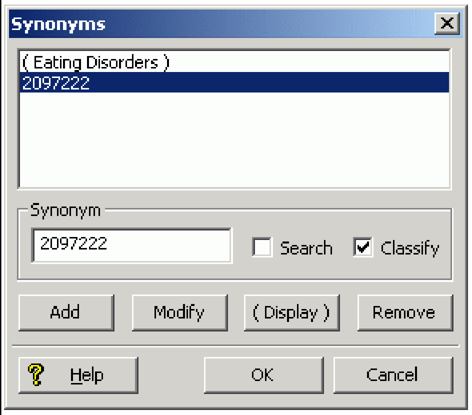
After you load the dimension values, you will notice that each dimension value has two synonyms. One synonym is the name of the dimension value used for display in an application, and the second synonym is an ID required for dimension value mapping.

You should not modify these synonym values.

The synonym with the name of the dimension value has default settings where **Search** is checked and **(Display)** is enabled, as shown in the following example. These settings allow an application user to search and navigate on the `Eating Disorders` dimension value.



The ID synonym has default settings on the **Synonym** editor where **Classify** is checked and **(Display)** is disabled, as shown in the following example. The **Classify** setting instructs Forge to use the ID synonym during record classification.



The ID synonym is based on a topic's `id` element as shown in the Stratify taxonomy. In the example above, here is a portion of the topic element for the `Eating Disorders` topic in the `Ask Alice` taxonomy:

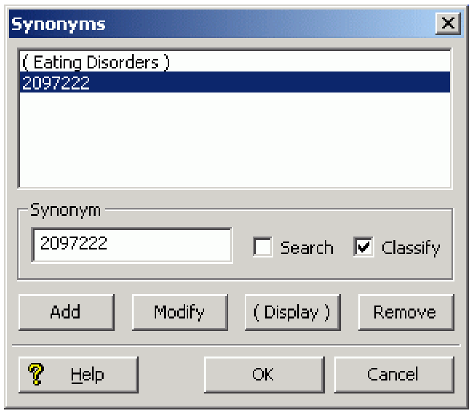
```
<topic id="2097222" name="Eating Disorders" ...</topic>.
```

About ID synonyms

The Endeca Information Access Platform uses the id synonym for dimension value tagging because an integer based on a Stratify id is guaranteed to be unique across other external dimensions, whereas the name of the dimension value is not guaranteed to be unique across other external dimensions.

An ID synonym, based on a Stratify id is not intended to provide an alternative way of describing or searching a dimension value in the same way that synonyms created using Developer Studio are often used.

The following process describes the role of id synonyms in dimension value mapping, using the example shown below:



1. After you transform a taxonomy and load a dimension, each dimension value has an id synonym that comes from the topic id in the Stratify taxonomy.
2. After Stratify classifies the unstructured documents, Forge tags properties to each record that include the Stratify topic id and Stratify topic name. Remember, this topic ID is the same as the id synonym.

   In this example, Forge assigns the properties `Endeca.Stratify.Top¬ic.Name.HID15.TID2097222="Eating Disorders"` and `Endeca.Stratify.Top¬ic.HID15="2097222"` after Stratify classifies an unstructured document into the `Eating Disorders` topic.

3. A property mapper maps a source property named `Endeca.Stratify.Topic.HID<`*`hierarchy id`*`>` to a target dimension for a Stratify taxonomy.

   In this example, `Endeca.Stratify.Topic.HID15` is mapped to the `AskAlice` target dimension.

4. Forge uses the ID synonyms to classify records into dimension values during mapping.

   In this example, any documents that Forge tags with `Endeca.Stratify.Top¬ic.HID15="2097222"` Forge also classifies with the ID synonym `2097222` into the `Eating Disorders` dimension value of the `AskAlice` dimension.

About mapping a dimension based on a Stratify taxonomy

Using **Discover** to add the dimension makes it available for explicit mapping as a target dimension.

You map the source property `Endeca.Stratify.Topic.HID<`*`hierarchy ID`*`>` to your target dimension for the taxonomy. All topics within that hierarchy are mapped as dimension values.

If you are using more than one Stratify taxonomy in your pipeline, create one property to dimension mapping for each taxonomy. For more information about using a property mapper, see the *Endeca Developer Studio Help*.

Running the second baseline update

After discovering and loading the dimension and mapping the dimension to a source property, you must run a second baseline update to process and tag your Endeca records.

You must load dimensions and map them to source properties before running your second update.

The second baseline update performs property and dimension mapping that could not be performed in the first baseline update because the Stratify taxonomy had not yet been transformed into an Endeca dimension and available for mapping.

To run a second baseline update:

Note: If you ran the first baseline update with the Forge `-n` flag, you should delete the flag before running the second baseline update.

1. Ensure you have sent the latest instance configuration from Developer Studio to Endeca Workbench.
2. In Endeca Workbench, run your baseline update script from the **EAC Admin Console** page.

About updating a taxonomy in your pipeline

To update an externally managed dimension in your taxonomy, you have to repeat the process of integrating Stratify into your pipeline.

If you want to modify an externally managed dimension and replace it with a newer version, you have to revise the taxonomy in Stratify Taxonomy Manager, publish it to the Stratify Classification Server, include it into Developer Studio, and make the other necessary changes in Developer Studio. This process is necessary because the Stratify Classification Server must manage all taxonomy changes in order to properly classify documents during Forge's record processing.

Part 4

Other Advanced Features

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

Chapter 18

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 (new, supported, or deprecated).

Argument	Status in 6.0	Old meaning	New 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`	New.	n/a	`DEBUG`
`-vf`	New.	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 6.0.1.52 ("i86pc-win32")

INFO 01/25/07 15:15:50.791 UTC FORGE {config}: Copyright 2001-2007 Endeca
Technologies, 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 19

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 *Application Controller Guide*.
- Programmatically, via the `webServicePort` on the `ForgeComponentType` object. For details, see the *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 *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.

Name	Purpose	Properties
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.
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

Option	Description
	is, it must start from zero and be sequential for all clients). Requires the `--client` option.
`--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. For details, see the *Implementing the Endeca Crawler* appendix.
`--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.

Option	Description
`--input-encoding` `<`*`encoding`*`>`	*Deprecated*. Specify the encoding of non-XML input files.
`--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

Option	Description
`--noAutoGen`	Do not generate new dimension value IDs (for incremental updates when batch processing is running).
`--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.

Option	Description
	<expression_type> specifies the type of regular or host expression to use: • `url-regex` • `url-wildcard` • `host-regex` • `host-wildcard` *<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>`	Set a cipher string (such as RC4-SHA) that specifies the minimum cryptographic algorithm the Forge server/client will use during the SSL negotiation. **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.

Option	**Description**
`--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
Adobe FrameMaker (MIF)	Versions 3.0, 4.0, 5.0, 5.5 and 6.0 and Japanese 3.0, 4.0, 5.0 and 6.0 (text only)
ANSI Text	7 & 8 bit
ASCII Text	7 & 8 bit
DEC WPS Plus (DX)	Versions through 3.1
DEC WPS Plus (WPL)	Versions through 4.1
DisplayWrite 2 & 3 (TXT)	All versions
DisplayWrite 4 & 5	Versions through 2.0
EBCDIC	All versions

Format	Version
Enable	Versions 3.0, 4.0, and 4.5
First Choice	Versions through 3.0
Framework	Version 3.0
Hangul	Version 97 and 2002
IBM FFT	All versions
IBM Revisable Form Text	All versions
IBM Writing Assistant	Version 1.01
JustSystems Ichitaro	Versions 4.x through 6.x, 8.x through 13.x, and 2004
JustWrite	Versions through 3.0
Legacy	Versions through 1.1
Lotus AMI/AMI Professional	Versions through 3.1
Lotus Manuscript	Version 2.0
Lotus WordPro (Windows)	SmartSuite 96, 97 and Millennium and Millennium 9.6
Lotus WordPro (non-Windows)	SmartSuite 97, Millennium, and Millennium 9.6 (text only)
MacWrite II	Versions 1.1
MASS11	Versions through 8.0
Microsoft Rich Text Format (RTF)	All versions
Microsoft Word (DOS)	Versions through 6.0
Microsoft Word (Macintosh)	Versions 4.0 through 2004
Microsoft Word (Windows)	Versions through 2007

Format	Version
Microsoft WordPad	All versions
Microsoft Works (DOS)	Versions through 2.0
Microsoft Works (Macintosh)	Versions through 2.0
Microsoft Works (Windows)	Versions through 4.0
Microsoft Windows Write	Versions through 3.0
MultiMate	Versions through 4.0
Navy DIF	All versions
Nota Bene	Version 3.0
Novell Perfect Works	Version 2.0
Novell/Corel WordPerfect (DOS)	Versions through 6.1
Novell/Corel WordPerfect (Macintosh)	Versions 1.02 through 3.0
Novell/Corel WordPerfect (Windows)	Versions through 12.0
Office Writer	Versions 4.0 through 6.0
OpenOffice Writer (Windows and UNIX)	OpenOffice version 1.1 and 2.0
PC-File Letter	Versions through 5.0
PC-File+ Letter	Versions through 3.0
PFS:Write	Versions A, B, and C
Professional Write (DOS)	Versions through 2.1
Professional Write Plus (Windows)	Version 1.0
Q&A (DOS)	Version 2.0

Format	Version
Q&A Write (Windows)	Version 3.0
Samna Word	Versions through Samna Word IV+
Signature	Version 1.0
SmartWare II	Version 1.02
Sprint	Versions through 1.0
StarOffice Writer	Version 5.2 (text only) and 6.x through 8.x
Total Word	Version 1.2
Unicode Text	All versions
UTF-8	All versions
Volkswriter 3 & 4	Versions through 1.0
Wang PC (IWP)	Versions through 2.6
WordMARC	Versions through Composer Plus
WordStar (DOS)	Versions through 7.0
WordStar (Windows)	Version 1.0
WordStar 2000 (DOS)	Versions through 3.0
XyWrite	Versions through III Plus

Spreadsheet formats

The following table lists supported spreadsheet formats:

Format	Version
Enable	Versions 3.0, 4.0, and 4.5
First Choice	Versions through 3.0
Framework	Version 3.0
Lotus 1-2-3 (DOS and Windows)	Versions through 5.0
Lotus 1-2-3 (OS/2)	Versions through 2.0
Lotus 1-2-3 Charts (DOS and Windows)	Versions through 5.0
Lotus 1-2-3 Charts (OS/2)	Versions through 2.0
Lotus 1-2-3 for SmartSuite	Versions 97 through Millennium 9.6
Lotus Symphony	Versions 1.0, 1.1, and 2.0
Microsoft Excel Charts	Versions 2.x through 7.0
Microsoft Excel (Macintosh)	Versions 3.0 through 4.0, 98, 2001, 2002, 2004, and v.X
Microsoft Excel (Windows)	Versions 2.2 through 2007
Microsoft Multiplan	Version 4.0
Microsoft Works (Windows)	Versions through 4.0
Microsoft Works (DOS)	Versions through 2.0
Microsoft Works (Macintosh)	Versions through 2.0
Mosaic Twin	Version 2.5
Novell Perfect Works	Version 2.0
PFS: Professional Plan	Version 1.0
QuattroPro (DOS)	Versions through 5.0 (text only)

Format	Version
QuattroPro (Windows)	Versions through 12.0 (text only)
SmartWare II	Version 1.02
StarOffice/OpenOffice Calc (Windows and UNIX)	StarOffice Versions 5.2 through 8.x; OpenOffice Version 1.1, 2.0 (text only)
SuperCalc 5	Version 4.0
VP Planner 3D	Version 1.0

Graphics formats

The following table lists supported graphics formats:

Format	Version
Adobe Photoshop (PSD)	All versions
Adobe Illustrator	Versions 7.0 and 9.0
Adobe FrameMaker graphics (FMV)	Vector/raster through Version 5.0
Adobe Acrobat (PDF)	Versions 1.0, 2.1, 3.0, 4.0, 5.0, 6.0, and 7.0 (including Japanese PDF). **Note:** The 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. To extract the text, you must re-create the PDF without setting the no-copy option. **Note:** Text added with the Sticky Note tool will not be extracted.
Ami Draw (SDW)	Ami Draw

Format	Version
AutoCAD Interchange and Native Drawing formats (DXF and DWG)	AutoCAD Drawing Versions 2.5 - 2.6, 9.0 - 14.0, 2000i, and 2002
AutoShade Rendering (RND)	Version 2.0
Binary Group 3 Fax	All versions
Bitmap (BMP, RLE, ICO, CUR, OS/2 DIB and Warp)	All versions
CALS Raster (GP4)	Type I and Type II
Corel Clipart (CMX)	Versions 5 through 6
Corel Draw (CDR)	Versions 3.x through 8.x
Corel Draw (CDR with TIFF header)	Versions 2.x through 9.x
Computer Graphics Metafile (CGM)	ANSI, CALS NIST Version 3.0
Encapsulated PostScript (EPS)	TIFF header only
GEM Paint (IMG)	All versions
Graphics Environment Manager (GEM)	Bitmap and Vector
Graphics Interchange Format (GIF)	All versions
Hewlett Packard Graphics Language (HPGL)	Version 2
IBM Graphics Data Format (GDF)	Version 1.0
IBM Picture Interchange Format (PIF)	Version 1.0
Initial Graphics Exchange Specification (IGES)	Version 5.1
JBIG2	JBIG2 graphic embeddings in PDF files

Format	Version
JFIF (JPEG not in TIFF format)	All versions
JPEG (including EXIF)	All versions
Kodak Flash Pix (FPX)	All versions
Kodak Photo CD (PCD)	Version 1.0
Lotus (PIC)	All versions
Lotus Snapshot	All versions
Macintosh PICT1 and PICT2	Bitmap only
MacPaint (PNTG)	All versions
Micrografx Draw (DRW)	Versions through 4.0
Micrografx Designer (DRW)	Versions through 3.1
Micrografx Designer (DSF)	Windows 95, Version 6.0
Novell Perfect Works (Draw)	Version 2.0
OS/2 PM Metafile (MET)	Version 3.0
Paintshop Pro 6 (PSP)	Windows only, Versions 5.0 - 6.0
PC Paintbrush (PCX and DCX)	All versions
Portable Bitmap (PBM)	All versions
Portable Graymap (PGM)	No specific version
Portable Network Graphics (PNG)	Version 1.0
Portable Pixmap (PPM)	No specific version
Postscript (PS)	Levels 1-2

Format	Version
Progressive JPEG	No specific version
StarOffice/OpenOffice Draw (Windows and UNIX)	StarOffice Versions 5.2 through 8.x; OpenOffice Version 1.1, 2.0 (text only)
Sun Raster(SRS)	No specific version
TIFF	Versions through 6
TIFF CCITT Group 3 and 4	Versions through 6
Truevision TGA (Targa)	Version 2
Visio (preview)	Version 4
Visio	Versions 5, 2000, 2002, and 2003
WBMP	No specific version
Windows Enhanced Metafile (EMF)	No specific version
Windows Metafile (WMF)	No specific version
WordPerfect Graphics (WPG and WPG2)	Versions through 2.0
X-Windows Bitmap (XBM)	x10 compatible
X-Windows Dump (XWD)	x10 compatible
X-Windows Pixmap (XPM)	x10 compatible

Presentation formats

The following table lists supported presentation formats:

Format	Version
Corel/Novell Presentations	Versions through 12.0
Harvard Graphics (DOS)	Versions 2.x and 3.x
Harvard Graphics (Windows)	Windows versions
Freelance (Windows)	Versions through Millennium 9.6
Freelance (OS/2)	Versions through 2.0
Microsoft PowerPoint (Windows)	Versions through 2007
Microsoft PowerPoint (Macintosh)	Versions 4.0 through v.X
StarOffice/OpenOffice Impress (Windows and UNIX)	StarOffice Versions 5.2 (text only) and 6.x through 8.x (full support); OpenOffice Version 1.1 and 2.0 (text only)

Compressed formats

The following table lists supported compressed formats:

Format	Version
GZIP	All versions
LZA Self Extracting Compress	All versions
LZH Compress	All versions
Microsoft Binder (conversion of Binder supported only on Windows)	Version 7.0 - 97
UUEncode	No specific version
UNIX Compress	No specific version
UNIX TAR	No specific version

Format	Version
ZIP	PKWARE versions through 2.04g

Database formats

The following table lists supported database formats:

Format	Version
Access	Versions through 2.0
dBASE	Versions through 5.0
DataEase	Version 4.x
dBXL	Version 1.3
Enable	Versions 3.0, 4.0, and 4.5
First Choice	Versions through 3.0
FoxBase	Version 2.1
Framework	Version 3.0
Microsoft Works (DOS)	Versions through 2.0
Microsoft Works (Macintosh)	Versions through 2.0
Microsoft Works (Windows)	Versions through 4.0
Paradox (DOS)	Versions through 4.0
Paradox (Windows)	Versions through 1.0
Personal R:BASE	Version 1.0
R:BASE 5000	Versions through 3.1

Format	**Version**
R:BASE System V	Version 1.0
Reflex	Version 2.0
Q & A	Versions through 2.0
SmartWare II	Version 1.02

E-mail formats

The following table lists supported e-mail formats:

Note: The following MIME-encoded mail message formats are supported:

- MIME formats, including:
  - EML
  - MHT (Web Archive)
  - NWS (Newsgroup single-part and multi-part)
  - Simple Text Mail (defined in RFC 2822)
- TNEF format
- MIME encodings, including:
  - base64 (defined in RFC 1521)
  - binary (defined in RFC 1521)
  - binhex (defined in RFC 1741)
  - btoa
  - quoted-printable (defined in RFC 1521)
  - utf-7 (defined in RFC 2152)
  - uue
  - xxe
  - yenc

The following encodings for the body of a message are supported:

- Text
- HTML
- RTF
- TNEF
- Text/enriched (defined in RFC1523)
- Text/richtext (defined in RFC1341)
- Embedded mail message (defined in RFC 822). This is handled as a link to a new message.

The attachments of a MIME message can be stored in many formats. Endeca processes all attachment types that our technology supports.

Format	Version
Microsoft Outlook Folder (PST)	Microsoft Outlook Folder and Microsoft Outlook Offline Folder files versions 97, 98, 2000, 2002, and 2003
Microsoft Outlook Message (MSG)	Microsoft Outlook Message and Microsoft Outlook Form Template versions 97, 98, 2000, 2002 and 2003
MIME	MIME-encoded mail messages. See preceding MIME support notes.

Other formats

The following table lists other supported formats:

Format	Version
Executable (EXE, DLL)	No specific version
HTML	Versions through 3.0, with some limitations
Macromedia Flash	Macromedia Flash 6.x, Macromedia Flash 7.x, and Macromedia Flash Lite (text only)
Microsoft Project	Versions 98 - 2002 (text only)
MP3	ID3 Metadata only
vCard, vCalendar	Version 2.1
WML	Version 5.2
XML	Text only
Yahoo! Instant Messenger	Versions 6.x and 7.x

Appendix C

The Endeca Crawler

This section describes how to configure and run the Endeca Crawler.

Overview of the Endeca Crawler

The Endeca Crawler provides the capability to crawl file systems, HTTP, and HTTPS hosts, in order to fetch documents in a variety of formats.

You use a record adapter to read in the documents to a crawler pipeline. Once read into the pipeline, Forge processes the documents and converts them into Endeca records. These records can contain property values, dimension values, and metadata based on each document's content. You can then build an Endeca application to access the records and allow your application users to search and navigate the document contents contained in the records.

Keep in mind that the Endeca IAP can process only HTML and TXT documents. You can gain the ability to process over 200 document types by installing the optional Endeca Document Conversion Module.

Important: The Endeca Crawler is deprecated, and will be removed in a future version of the Endeca Information Access Platform. Therefore, if you are beginning a new project, it is recommended that you use the Endeca Web Crawler, which is a component of the Endeca Content Acquisition System.

About installing the Endeca Crawler

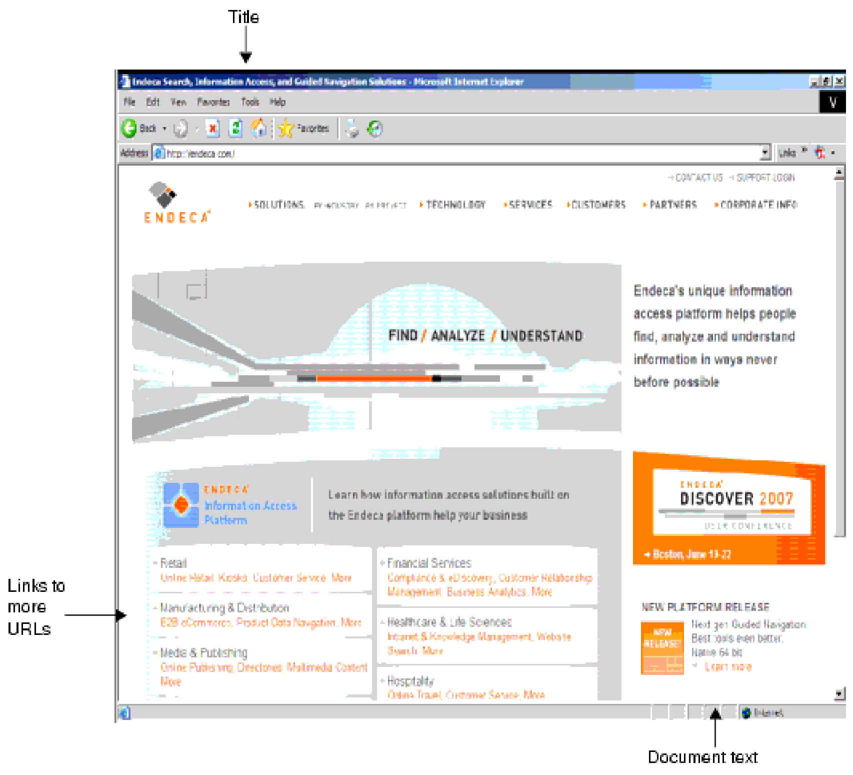
The components required to set up an Endeca Crawler application are included in Developer Studio. Therefore, installing Developer Studio will ensure that you have the necessary software for running a crawl.

Source documentation and Endeca records

In the context of an Endeca Crawler application, Endeca records represent both the data in the source documents and metadata about the source documents.

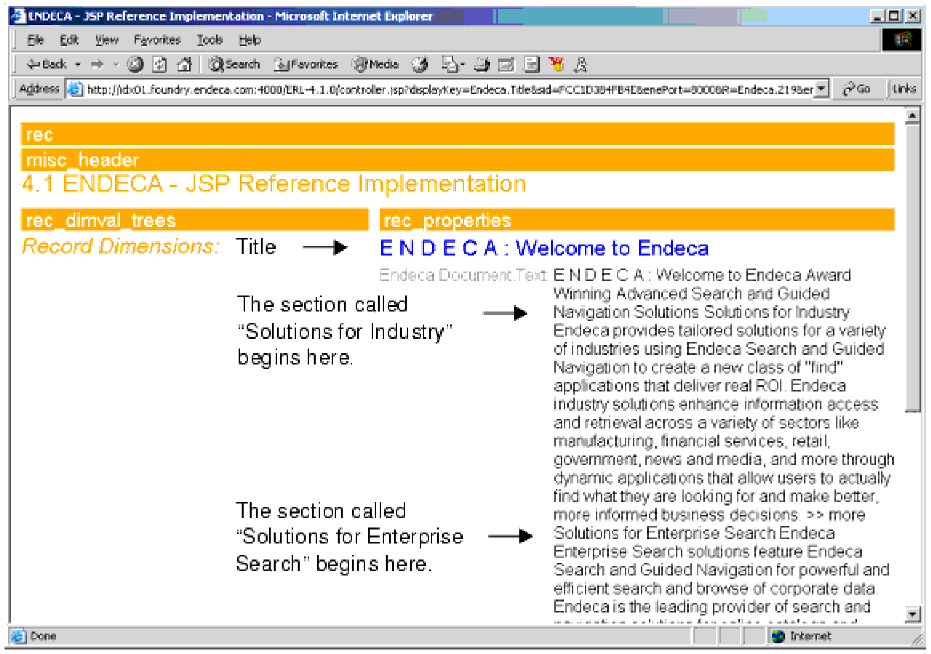
Crawlers provide the means to get records representing the source documents into a pipeline. The source documents themselves may reside on a file system, HTTP, or HTTPS host and be in a wide variety of file formats (common examples include PDF, HTML, DOC, and TXT). As with non-crawler pipelines, the source documents themselves are not modified in any way by the pipeline processing.

In the following example, the Endeca Crawler crawls an HTML source document from the Endeca Web site. The document has a title, text that describes Endeca solutions, and links to other areas of the Web site.


Title
Links to more URLs
Document text
Endeca Search, Information Access, and Guided Navigation Solutions - Microsoft Internet Explorer
http://endeca.com/
ENDECA
SOLUTIONS
TECHNOLOGY
SERVICES
CUSTOMERS
PARTNERS
CORPORATE INFO
FIND / ANALYZE / UNDERSTAND
Endeca's unique information access platform helps people find, analyze and understand information in ways never before possible
Learn how information access solutions built on the Endeca platform help your business
DISCOVER 2007
Boston, June 19-22
NEW PLATFORM RELEASE
Retail
Manufacturing & Distribution
Media & Publishing
Financial Services
Healthcare & Life Sciences
Hospitality

During Data Foundry processing, Forge generates an Endeca record for the document. Among other things, that record contains the body of the source document and the title of the document.

Suppose only the properties for the body of the source document and the title of the document are mapped using the property mapper. The Endeca record for this document looks like this:


ENDECA - JSP Reference Implementation - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://idx01.foundry.endeca.com:4000/ERL-4.1.0/controller.jsp?displayKey=Endeca.Title&sid=FCC1D384FB4E&enePort=8000&R=Endeca.219&er
Go Links
Done
Internet

rec

misc_header

4.1 ENDECA - JSP Reference Implementation

rec_dimval_trees

rec_properties

Record Dimensions: Title → E N D E C A : Welcome to Endeca

Endeca.Document.Text: E N D E C A : Welcome to Endeca Award Winning Advanced Search and Guided Navigation Solutions Solutions for Industry Endeca provides tailored solutions for a variety of industries using Endeca Search and Guided Navigation to create a new class of "find" applications that deliver real ROI. Endeca industry solutions enhance information access and retrieval across a variety of sectors like manufacturing, financial services, retail, government, news and media, and more through dynamic applications that allow users to actually find what they are looking for and make better, more informed business decisions. >> more Solutions for Enterprise Search Endeca Enterprise Search solutions feature Endeca Search and Guided Navigation for powerful and efficient search and browse of corporate data. Endeca is the leading provider of search and

The section called "Solutions for Industry" begins here. →

The section called "Solutions for Enterprise Search" begins here. →

Notice the following correspondence between the source document and the Endeca record:

- The heading in blue is the document's title.
- The first line in the record's `Endeca.Document.Text` property also lists the source document's title and the first line under the documents main graphic. The text begins "Award Winning…"
- The remaining lines correspond to the two sections of the source document called "Solutions for Industry" and "Solutions for Enterprise Search".

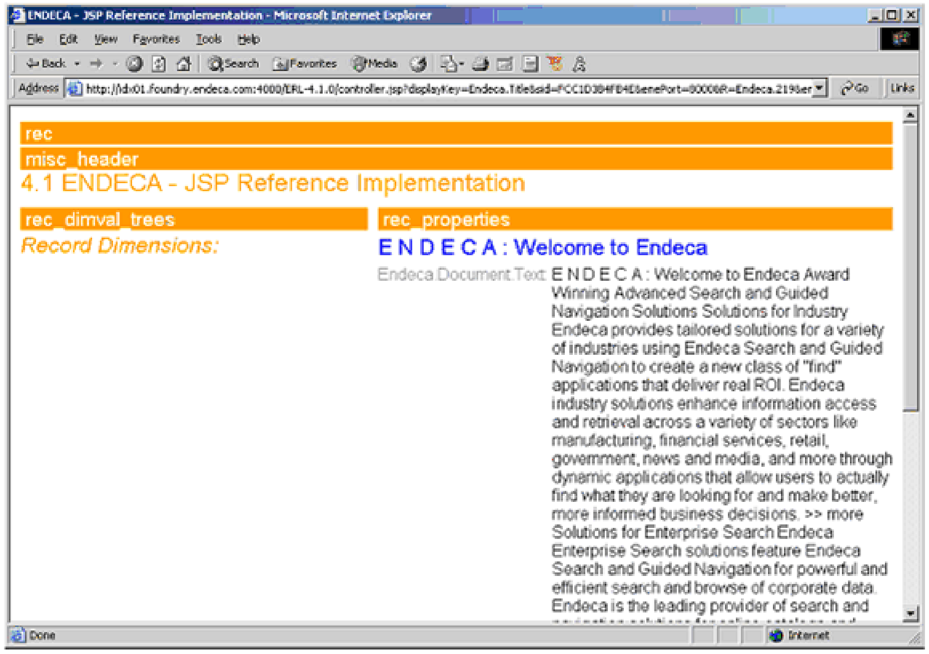
Note: The order in which a source document's data appears in an Endeca record depends upon the structure of the actual source document. The ordering shown here is an example.

Although this example is useful for illustrative purposes, such a record is not very useful to application users. Here, it shows the simplest relationship between a source document and an Endeca record with two properties (title and text). An application for users is not likely to have all of a document's data contained in a single property.

In a more user-oriented application, a crawler pipeline might include Perl code to parse properties from the document text and use those to build Endeca properties and dimensions. Alternatively, the pipeline might build dimensions based on any of the metadata properties that are generated for a record.

Suppose the example mapped several more properties. In addition to title and text, it might also map metadata properties. From the metadata properties available, the pipeline could be set up to expose properties such as encoding, date modified, application type, fetch status, and so, for use as dimensions. These properties would be mapped with a property mapper component to provide both record details and navigation controls in the application.

Re-running a baseline to map all available properties would produce an Endeca record that looks like this:



rec

misc_header

4.1 ENDECA - JSP Reference Implementation

rec_dimval_trees

Record Dimensions:

rec_properties

E N D E C A : Welcome to Endeca

Endeca.Document.Text E N D E C A : Welcome to Endeca Award Winning Advanced Search and Guided Navigation Solutions Solutions for Industry Endeca provides tailored solutions for a variety of industries using Endeca Search and Guided Navigation to create a new class of "find" applications that deliver real ROI. Endeca industry solutions enhance information access and retrieval across a variety of sectors like manufacturing, financial services, retail, government, news and media, and more through dynamic applications that allow users to actually find what they are looking for and make better, more informed business decisions. >> more Solutions for Enterprise Search Endeca Enterprise Search solutions feature Endeca Search and Guided Navigation for powerful and efficient search and browse of corporate data. Endeca is the leading provider of search and

The properties Forge generates are prefixed with `Endeca`. A user-oriented application may not employ all of the properties that Forge generates, but it is useful to see some of them here. Notice the following changes to the revised Endeca record:

- All metadata properties in the source document appear with the record, rather than just `Endeca.Document.Text` and `Endeca.Title` as shown previously.
- There are dimension values based on meta-data properties for encoding, fetch status, and MIME type.

To build a record page that displays all properties, the property mapper must be configured to map all of the source properties to Endeca properties.

Crawling errors

Processing source documents, including retrieving and extracting text can introduce problems. This section lists several common errors and any workarounds, if applicable.

PDF content in `Endeca.Document.Text` displays as binary data — If the Endeca Crawler processes PDF files and the content from those files appears in your application as binary data, the PDF files may contain custom-encoded embedded fonts. It cannot always correctly display content that contains custom-encoded embedded fonts. To solve the issue, a system font is substituted for the custom-encoded font. The substitution succeeds if the encoding in the substituted system font is the same as the custom encoding in the embedded font. When the substitution is not successful, you see binary data in `Endeca.Document.Text`.

Here are several issues related to retrieving documents from HTTP hosts, and an explanation of how the spider handles them:

- **Connection timeout** — The spider retries the request five times. Each timeout is logged in an informational message. After a fifth timeout, an error message is logged, and the record for the offending URL is created with its `Endeca.Document.Status` property set to "Fetch Aborted."
- **URL not found** — The spider logs a warning message that the URL could not be located and creates a record with its `Endeca.Document.Status` property set to “Fetch Failed.”

- **Malformed URL** — The spider logs a warning message that the URL is malformed and creates a record with its `Endeca.Document.Status` property set to “Fetch Failed.”
- **Authentication failure** — The spider logs a warning message that the URL could not be retrieved and creates a record with its `Endeca.Document.Status` property set to “Fetch Failed.”

Endeca Crawler operational details

This section documents some of the operational facets of the Endeca Crawler, including information on how URLs are processed and how Forge generates property names.

Security information and the Endeca Crawler

The Endeca Crawler supports accessing hosts that require basic client authentication and HTTPS authentication.

In addition, the Endeca Crawler supports the Access Control System because it can crawl file systems and generate access control list (ACL) properties for each record. These properties can be used in conjunction with security login modules to limit access to records based on user login profiles. (Login modules are a part of the Access Control System.) For details on gathering security information, see the *Endeca Security Guide*.

Related Links

About configuring HTTPS authentication on page 189

HTTPS configuration is similar to HTTP authentication configuration. Forge supports HTTPS authentication of the server, client authentication with certificates, and secure communication over HTTPS.

Full crawls vs differential crawls

There are two types of crawls a spider can perform:

- **Full crawl** — A crawl in which the spider retrieves all the documents that it is configured to access. A full crawl is analogous to a full update in a basic data pipeline. These sections describe a full crawl.
- **Differential crawl** — A re-crawl in which the spider retrieves only the documents that have changed since the last crawl. The differential crawl URL specified on the **General** tab of the **Spider editor** indicates a file in which Forge stores URLs and metadata about URLs. By reading this file at the beginning of a crawl, the spider can detect source documents that have been modified (updated, added, or otherwise altered) since the last crawl.

Related Links

Differential Crawling on page 195

This section provides an overview of differential crawling.

How redundant URLs are handled

To minimize the possibility of creating redundant records in an application, the Endeca Crawler avoids crawling a URL that has already been retrieved. However, there are some situations where the crawler cannot identify duplicate resources.

The Endeca Crawler can compare URLs to determine if they are equivalent.

Even in cases where flexible URL formatting allows two URLs to be different strings and yet point to the same resource, the crawler can determine that the URLs are equivalent, and will not process both URLs. For example, these two URLs point to the same resource:

```
http://www.endeca.com:80/about/../index.shtml
http://www.endeca.com/index.shtml
```

The crawler recognizes the equivalence and does not process both URLs.

However, there is no foolproof way to determine if two non-equivalent URLs point to the same resource. If the following configurations are present in a crawling environment, it is possible to cause a crawler to queue URLs that point to the same resource:

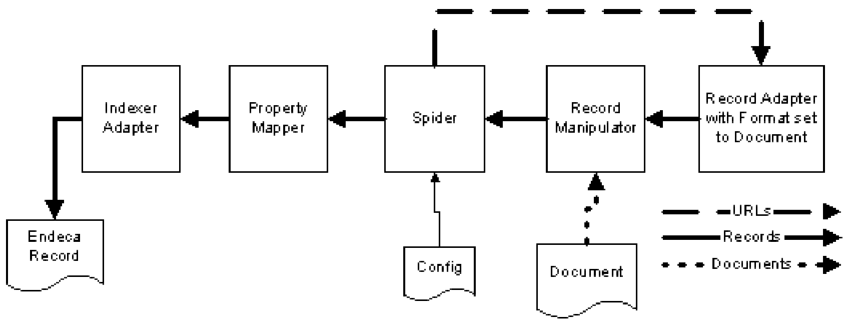
- Symbolic links: For example, if `file:///data.txt` is a symbolic link to `file:///machineA/directory/data.txt`, the crawler queues both URLs and processes `data.txt` twice.
- Virtual hosts: For example, if a crawler was set up to crawl both `http://web.mit.edu/index.html` and `http://www.mit.edu/index.html`, it would crawl the same resource twice, because one host name is an alias for the other.
- IP Address/DNS name equivalence: The crawler does not perform reverse DNS look-ups to determine if two URLs are equivalent. For example, if a crawler is set up to crawl `http://www.mit.edu/index.html` and `http://18.181.0.31/index.html`, it crawls the same resource twice because the IP address in the second URL is the same machine as the host name in the first URL.

URL and record processing

Because Developer Studio exposes crawling and text extraction functionality in the context of a pipeline, it is important to understand how this functionality fits into the Forge processing framework. The following figure shows a diagram of a full crawling pipeline.

There are two kinds of flow in the pipeline:

- URLs flow from the spider to the record adapter (a record adapter that uses the `Document` format)
- Documents flow into the indexer adapter and are transformed into Endeca records.



When Forge executes this pipeline, the flow of URLs and records is as follows:

1. The terminating component (indexer adapter) requests the next record from its record source (property mapper).
2. At this point, the property mapper has no record, so the property mapper asks its record source (spider) for the next record.

3. The spider has no record, so the spider asks its record source (record manipulator) for the next record.
4. The record manipulator also has no record, so it passes the request for the next record upstream to the record adapter (with format type `Document`).
5. The record adapter asks the spider for the next URL it is to retrieve (the first iteration through, this is the root URL configured on the **Root URL** tab of the **Spider editor**).
6. Based on the URL that the spider provides, the record adapter creates a record containing the URL and a limited set of metadata.
7. The created record flows down to the record manipulator where the following takes place:
    a. The document associated with the URL is fetched (using the `RETRIEVE_URL` expression).
    b. Content (searchable text) is extracted from the document (using the `CONVERTTOTEXT` or `PARSE_DOC` expression).
    c. Any URLs in the text are also extracted for additional crawling.
8. The record moves to the spider where additional URLs (those extracted in the record manipulator) are queued for crawling.
9. The property mapper performs property to dimension and source property to Endeca property mapping.
10. The indexer adapter receives the record and writes it out to disk.

The process repeats until there are no URLs in the URL queue maintained by the spider.

Generated record properties syntax reference

During a crawl, Forge produces properties according to a standardized naming scheme. The conventions are outlined below.

The naming scheme is made up of a qualified name with a period (.) to separate qualifier terms. The first term `Endeca` indicates that the property was automatically created by Forge. There may be any number of additional terms depending on the property being described.

Simple properties require only one additional term to fully qualify the property, for example, `Endeca.Identifier` or `Endeca.Title`. Often, a second term describes a property category and a third term fully qualifies the property, for example, `Endeca.Document.Body` or `Endeca.Fetch.Timeout`. Less frequently, properties require additional terms.

The following table provides an overview of naming syntax and includes several common property examples:

Common qualifier terms	Description	Example with fully qualified names

`Document`	Text of the document or metadata the document	`Endeca.Document.Text`
		`Endeca.Document.Revision`
		`Endeca.Document.MimeType`
		`Endeca.Document.XHTML`

`Fetch`	Configuration information about how the document is retrieved	`Endeca.Fetch.ConnectTimeout` `Endeca.Fetch.Proxy`

Common qualifier terms	Description	Example with fully qualified names

`ACL`	Security information (access control list) about the document	`Endeca.ACL.Allow.Read`
		`Endeca.ACL.Allow.Write`
		`Endeca.ACL.Allow.Execute`

`Relation`	Reference information to other documents	`Endeca.Relation.References`

Generated record properties reference

The following table describes all the properties generated by various components in a crawler pipeline that can be included in your Endeca records.

Not all of the properties listed below are likely to appear. The properties that are included depend on the type of source data crawled.

Note: Records created by the Endeca Crawler may contain properties prefixed with the name `Endeca.Fetch` that describe how to access the resource identified by the `Endeca.Identifier` property. The `Endeca.Fetch` property values are created based on the values you provide, if any, in the **Timeout** tab of the **Spider editor**. When the crawler sends a URL to the record adapter, the URL is accompanied by the timeout configuration values that determine how the URL should be retrieved.

Note: In addition to the properties described above, a text extraction expression may create additional properties with arbitrary names that describe metadata about a document, if such metadata exists. For example, if a document contains HTML tags, the expression creates corresponding properties. Or, for example, if you parse MS Word documents that contain an `Author` attribute, the expression creates a corresponding property for `author`.

Property name	Description
`Endeca.ACL.Allow.Read (Write, Execute, Delete, and so on)`	Security information about the document. There can be several dozen access control list (ACL) properties. These properties take their names from system security attributes. For more information, see the *Endeca Security Guide*.
`Endeca.ACL.Deny.Read (Write, Execute, Delete, and so on)`	Security information about the document. For more information, see the *Endeca Security Guide*.
`Endeca.Cookie`	Information about cookies associated with the fetched URL. This information may include the name of the cookie, the Web server's domain, the path to the Web server, fetch dates, and remove dates.

Property name	Description
`Endeca.Document.Body`	The body of the retrieved document. The value of this property is a path to the file that stores the document body on disk. A text extraction expression, such as `CONVERTTO¬TEXT` or `PARSE_DOC`, uses `Endeca.Document.Body` to extract the text of the document.
`Endeca.Document.Encoding`	The encoding of the body of the document, if it can be determined. This property value could be an ISO code or other encoding representation, for example, UTF-8, CP1252, ISO-8859-1. The `RETRIEVE_URL` expression attempts to determine this value based on the Content-type header of the document that a Web server returns to Forge. If no value exists for the Content-type header, then a text extraction expression (for example, `CONVERTTOTTEXT` or `PARSE_DOC`) attempts to determine the encoding value and to generate the property. If the value cannot be determined, Forge logs an error.
`Endeca.Document.IsRedirection`	If the document is a redirection to another document (for example, an HTTP Redirect or a symbolic link file), this property has the value `true`. The document's URL that is redirected to is stored in an `Endeca.Relation.Refer¬ences` property.
`Endeca.Document.IsUnchanged`	If the document is unchanged, either because the fetch was skipped or because fetch determined that the document has not changed, this property has the value `true`. This property is generated in differential crawl pipelines, not full crawl pipelines.
`Endeca.Document.Info.<scheme>.*`	Scheme-specific metadata supplied by each protocol. In this context, the term scheme is synonymous with protocol. For example, crawling HTTP URLs supplies HTTP protocol information in `Endeca.Document.Info.http` and the `FILE` protocol supplies the file type in the `Endeca.Docu¬ment.Info.file.Attribute` property. When crawling HTTP or HTTPS, the value of this property corresponds to either an HTTP response code or an internal product code. The HTTP response codes are defined by W3C. Typical examples are 200 (OK), 404 (Not Found) and so on. Internal product codes are as follows: 390 - Refresh meta tag 391 - Too many redirects

Property name	Description
	392 - Page redirected to self 490 - Malformed URL 491 - Prohibited by robots.txt or metatags 492 - Prohibited by do not crawl blacklist 493 - File is too large to process 494 - Could not find data handler for the file 495 - Failed to parse the HTML 590 - Unknown host. DNS lookup failed. 591 - Host unreachable. Network connection to the Web server is broken 592 - Network error. I/O exception during communication. 593 - Bad HTTP response. Unexpected HTTP interaction.
`Endeca.Document.Language`	The document language code. ISO 639 lists the valid language codes. Common examples include the following:`en` for English, `es` for Spanish, `fr` for French, `de` for German, `ja` for Japanese, and `ko` for Korean.
`Endeca.Document.MimeType`	The MIME Type of the document, if it can be determined. Common examples of this property value include text/html, application/pdf, image/gif, and so on.
`Endeca.Document.NumberOfHops`	The minimum number of hops (link traversals) to get to this document from a root URL during the crawl.
`Endeca.Document.Revision`	The protocol-specific revision information of the document. This value is typically a timestamp of the document's last modified date and may also include revision information about references and metadata. The information contained in the revision varies from protocol to protocol (for example, file vs. http).
`Endeca.Document.Status`	The status of the fetch. This property can have any of the following values: • **Fetch Succeeded** — The document fetch was successful and the document body is in the file indicated by the `Endeca.Document.Body` property. • **Fetch Skipped** — The document was not retrieved because based on the value of `Endeca.Document.Re¬vision`, the document should not yet be considered for re-fetch.

Property name	Description
	• **Fetch Aborted** — The document was not retrieved. This status indicates that a potentially transient phenomenon, for example a timeout, prevented the document from being fetched. Additional information can often be found in scheme-specific properties. • **Fetch Failed** — The document was not retrieved. This status indicates that a non-transient phenomenon, for example an HTTP error like `Document Not Found`, caused the failure. Additional information can often be found in scheme-specific properties.
`Endeca.Document.Text`	The text of the source document. For binary source documents, the `CONVERTTOTEXT` expression converts the text from the file indicated by `Endeca.Document.Body`. For text-based documents, the `PARSE_DOC` expression converts the document body in `Endeca.Document.Body`.
`Endeca.Host`	The host name where the document resides.
`Endeca.Identifier`	The source URL of this document.
`Endeca.Fetch.Timeout`	The maximum time in seconds that Forge should wait to retrieve the resource indicated by `Endeca.Identifier` before aborting the retrieve operation. Forge creates this property by converting Spider configuration settings to properties, and then adds the properties to a record. In particular, the `Endeca.Fetch.*` properties correspond to configuration settings on the **Timeout** tab of the **Spider editor**.
`Endeca.Fetch.ConnectTimeout`	The maximum time in seconds that Forge should wait to establish a connection to the server hosting the resource indicated by `Endeca.Identifier` before aborting the retrieve operation. Forge creates this property by converting Spider configuration settings to properties, and then adds the properties to a record. In particular, the `Endeca.Fetch.*` properties correspond to configuration settings on the **Timeout** tab of the **Spider editor**.
`Endeca.Fetch.Transfer¬ RateLowSpeedLimit`	The minimum transfer rate in bytes per second below which Forge should abort the retrieve operation after the timeout specified in `Endeca.Fetch.TransferRateLowSpeed¬ Time`.

Property name	Description
	Forge creates this property by converting Spider configuration settings to properties, and then adds the properties to a record. In particular, the `Endeca.Fetch.*` properties correspond to configuration settings on the **Timeout** tab of the **Spider editor**.
`Endeca.Fetch.Transfer¬ RateLowSpeedTime`	The maximum time in seconds that Forge should allow the transfer operation to fall below the minimum transfer rate specified in `Endeca.Fetch.Transfer¬ RateLowSpeedLimit` before aborting. Forge creates this property by converting Spider configuration settings to properties, and then adds the properties to a record. In particular, the `Endeca.Fetch.*` properties correspond to configuration settings on the **Timeout** tab of the **Spider editor**.
`Endeca.Fetch.Proxy`	The URI (`HOST:PORT`) of the proxy server that Forge should use to retrieve the resource indicated by `Ende¬ ca.Identifier`. Forge creates this property by converting Spider configuration settings to properties, and then adds the properties to a record. In particular, the `Endeca.Fetch.*` properties correspond to configuration settings on the **Timeout** tab of the **Spider editor**.
`Endeca.Fetch.UserAgent`	The User-Agent HTTP header that Forge should present to the server hosting the resource indicated by `Ende¬ ca.Identifier`. Forge creates this property by converting Spider configuration settings to properties, and then adds the properties to a record. In particular, the `Endeca.Fetch.*` properties correspond to configuration settings on the **Timeout** tab of the **Spider editor**.
`Endeca.Relation.References`	If a document references other documents, each reference is placed into this property. `Endeca.Relation.References` is the default property name produced by either text extraction expression. The property stores any additional URLs to be queued. Property values are either absolute URLs or URLs relative to the record's `Endeca.Identifier` property. For example, crawling a directory overview page that has links to three sub-category pages produces the following `Endeca.Relation.References` properties:

Property name	Description
	`Endeca.Relation.References=http://ende¬ca.com/products/` `Endeca.Relation.References=red/index.html` `Endeca.Relation.References=white/index.html` `Endeca.Relation.References=sparkling/in¬dex.html`
`Endeca.Spider.Depth`	Number of levels from `Endeca.Host` where the source document resides.
`Endeca.Stratify.Topic.HID` *`<hi¬erarchy ID>`*`=` *`<topic ID>`*	*Stratify implementations only*. This property corresponds to the ID value of a topic in your published Stratify taxonomy. Each topic in your taxonomy has an ID value assigned by Stratify. For example, if an `Eating Disorders` topic has an ID of 209722 in a health care taxonomy whose hierarchy ID is 15, then the Endeca property is `Endeca.Stratify.Top¬ic.HID15="209722"`.
`Endeca.Stratify.Topic.Topic Name.HID` *`<hierarchy ID>`*`.TID` *`<topic ID>`*`=`*`<topic name>`*	*Stratify implementations only*. This property corresponds to a topic name from your published Stratify taxonomy for its corresponding topic ID. For example, for the `Eating Disorders` topic in the health care taxonomy mentioned earlier, this property is `Endeca.Stratify.Top¬ic.Name.HID15.TID2097222="Eating Disorders"`.
`Endeca.Stratify.Topic.Score.HID` *`<hierarchy ID>`*`.TID` *`<topic ID>`*`=`*`<score>`*	*Stratify implementations only*. This property indicates classification score between an unstructured document and the topic it has been classified into. The value of *<score>* is a percentage expressed as a value between zero and one. Zero indicates the lowest classification score (0%), and one indicates the highest score (100%). You can use this property to remove records from your application that have a low score for classification matching, for example, `Endeca.Stratify.Top¬ic.Score.HID15.TID2097222="0.719380021095276"`.
`Endeca.Title`	The title of the document.

Related Links

Specifying timeouts on page 183

The spider may be configured with three timeout values specified in the **Timeout** tab. These optional values control connection timeouts and URL retrieval timeouts for each URL that the spider fetches.

Viewing properties created by the Endeca Crawler

Although you might not employ all the properties in your application, it is useful to see which properties are available.

You control which properties are available by modifying the property mapper to map the source properties to Endeca properties. Unmapped properties are not generated.

To view all properties:

1. In the **Project** tab of Developer Studio, double-click **Pipeline Diagram**.
2. Double-click the property mapper.
   The **Property Mapper editor** displays.
3. Click the **Advanced** tab of the **Property Mapper editor**.
4. Check **“If no mapping is found, map source properties to Endeca”** and then click **“Properties”**.

   Note: Remember that enabling this option is helpful to explore your records. A production application should map properties explicitly.

5. Click **OK**.
6. Perform a baseline update.

   See the *Endeca Developer Studio Help* for more information about configuring the property mapper and running full updates.
7. Start an Endeca application (for example, the JSP reference implementation) and view your Endeca records.

The full crawling pipeline

These sections describe how to create and configure a full crawling pipeline using Developer Studio.

About creating a full crawling pipeline

This section describes the pipeline components specific to crawling. Components that are common to non-crawler pipelines (dimension server, property mapper, indexer adapter, etc.) are omitted for simplicity.

This section focuses on the processing loop for a crawling pipeline that is made up of the record adapter, record manipulator, and spider components.

Note: A differential crawl pipeline requires a different design than that of a full crawl.

Related Links

Differential Crawling on page 195
  This section provides an overview of differential crawling.

Components that support the Endeca Crawler

Developer Studio exposes Endeca Crawler functionality using the following components, which form the core of an Endeca Crawler pipeline:

- **A spider component** — Crawls documents starting at the root URLs you specify. In the spider component, you indicate the root URLs from which to begin a crawl, URL filters to determine which documents to crawl, as well as other configuration information that specifies how the crawl should proceed. This information may include timeout values for a crawl, proxy server values, and so on. The spider crawls the URLs and manages a URL queue that feeds the record adapter.
- **A record adapter configured to read documents** — Receives URLs from the spider and creates an Endeca record for each document located at a URL. Each record contains a number of properties, one of which is the record's identifying URL. A downstream record manipulator uses the record identifier to retrieve the document and extract its data.

  Unlike basic pipelines (which use a record adapter to input source data from a variety of formats), an Endeca Crawler pipeline uses a record adapter to input URLs provided by the spider. In a basic pipeline, the format type of a record adapter matches the source data, for example, delimited, XML, fixed-width, or ODBC. In an Endeca Crawler pipeline, the format type of a record adapter must be set to `Document`.
- **A record manipulator incorporating expressions to handle documents** — Contains several Data Foundry expressions that support crawling and document processing tasks. At a minimum, a record manipulator contains one expression to retrieve a URL based on the record's identifier and a second expression to extract and convert the document's content to text. In addition, you can include optional expressions to identify the language of a document, remove temporary properties after processing is complete, or perform a variety of other processing tasks.

Related Links

URL and record processing on page 162

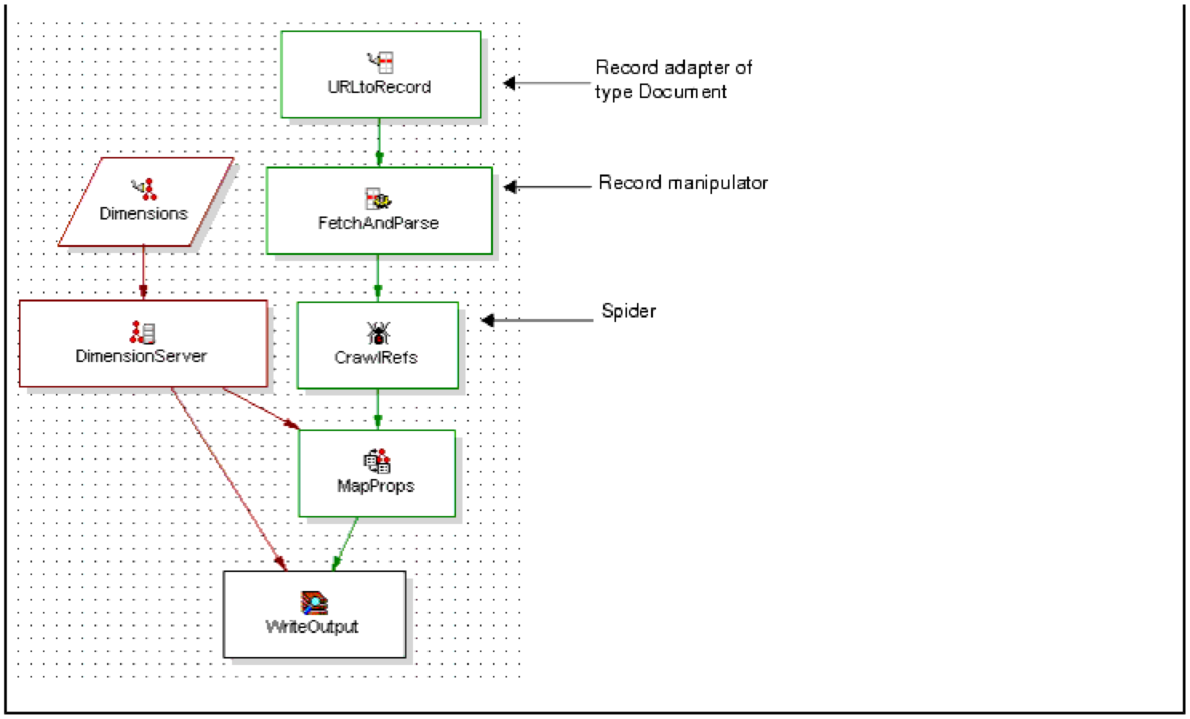
Because Developer Studio exposes crawling and text extraction functionality in the context of a pipeline, it is important to understand how this functionality fits into the Forge processing framework. The following figure shows a diagram of a full crawling pipeline.

Implementing a full crawling pipeline

The high-level overview of a full crawling pipeline is as follows:

1. Create a record adapter to read documents.
2. Create a record manipulator to perform the following tasks.
   a) Retrieve documents from a URL.
   b) Extract and convert document text for each URL.
   c) (Optional) Identify the language of a document.
   d) (Optional) Remove document body properties.
3. (Optional) Modify records with a Perl manipulator.
4. Create a spider to send URLs to the record adapter.
   a) Provide root URLs from which to start a crawl.
   b) Configure URL extraction settings.
   c) Specify a record source for the spider.
   d) (Optional) Specify spider settings such as timeout values and proxy servers.
5. (Optional) Create a record manipulator to remove any unnecessary records after processing.

Here is an example of a pipeline that calls out the core crawler-relevant components and also shows the components common to both basic and crawler pipelines (such as the dimension adapter, property mapper, and indexer adapter):



Related Links

Creating a record adapter to read documents on page 172
: A record adapter reads in the documents associated with the URLs provided by the spider component, and creates a record for each document. As long as the spider has URLs queued, the record adapter creates a record for each URL until all are processed.

Creating a record manipulator on page 173
: Expressions in a record manipulator perform document retrieval, text extraction, language identification, record or property clean up, and other tasks related to crawling. These expressions are evaluated against each record as it flows through the pipeline, and the record is changed as necessary.

About modifying records with a Perl manipulator on page 178

Creating a spider on page 178
: Follow the steps below to set up a spider in your Endeca Crawler pipeline.

Adding the REMOVE_RECORD expression to your pipeline on page 186
: You can remove records after a crawl by adding a `REMOVE_RECORD` expression to your pipeline.

Creating a record adapter to read documents

A record adapter reads in the documents associated with the URLs provided by the spider component, and creates a record for each document. As long as the spider has URLs queued, the record adapter creates a record for each URL until all are processed.

To create a `Document` record adapter:

1. Start Developer Studio.
2. Select **File** > **New Project**.
3. In the **Project** tab of Developer Studio, double-click **Pipeline Diagram**.

4. In the **Pipeline Diagram editor**, click **New**.
5. Select **Record** > **Adapter**. The **Record Adapter editor** displays.
6. In the **Name** text box, type in the name of this record adapter.
7. In the **Direction** frame, make sure the **Input** option is selected.
8. From the **Format** drop-down list, choose **Document**.
9. Leave the **URL** text box empty, and leave **Filter Empty Properties** and **Multi File** unchecked. These settings are ignored by a record adapter configured for `Document` formats.
10. If you know that all of the source documents are of the same encoding type, enter a language encoding in the **Encoding** text box.

    If you do not provide an encoding value, the Endeca ITL automatically attempts to determine the encoding of each document by either requesting that information from the Web server or by examining the document's body.
11. Click the **Pass Throughs** tab of the record adapter.
12. Enter `URL_SOURCE` in the **Name** text box and enter the name of the spider component in the **Value** text box. You will configure the spider later; for now, choose the name of the spider. The URL source is required and must name a spider component.
13. Click **Add**.
14. Click **OK** to add the new record adapter to the project.
15. Select **File** > **Save**.

Related Links

Creating a spider on page 178
Follow the steps below to set up a spider in your Endeca Crawler pipeline.

Generated record properties reference on page 164
The following table describes all the properties generated by various components in a crawler pipeline that can be included in your Endeca records.

Creating a record manipulator

Expressions in a record manipulator perform document retrieval, text extraction, language identification, record or property clean up, and other tasks related to crawling. These expressions are evaluated against each record as it flows through the pipeline, and the record is changed as necessary.

For in-depth information about the expressions that can be used in a record manipulator, see the *Data Foundry Expression Reference*.

At a minimum, a crawler pipeline requires a record manipulator with two expressions: one to retrieve documents (`RETRIEVE_URL`) and another to convert documents to text (`CONVERTTOTEXT` or `PARSE_DOC`). In addition to these expressions, you can include other optional expressions to delete the temporary files created on disk by `RETRIEVE_URL` (using `REMOVE_EXPORTED_PROP`).

To create a record manipulator:

1. In the **Project** tab of Developer Studio, double-click **Pipeline Diagram**.
2. In the **Pipeline Diagram** editor, click **New**.
3. Select **Record** > **Manipulator**.
   The **New Record Manipulator editor** displays.
4. In the **Name** text box, type in the name of this record manipulator.
5. From the **Record source** drop-down list, choose the name of the record adapter.

6. Click **OK** to add the new record manipulator to the project.
7. Select **File** > **Save**.
8. If you are ready to add the expressions described in the sections below, double-click the record manipulator in your pipeline diagram.
   The **Expression editor** displays.

Related Links

Adding a RETRIEVE_URL expression on page 174
: The `RETRIEVE_URL` expression is required to retrieve a document from its URL and store it in a file on disk.

Converting documents to text on page 176
: A record manipulator must contain a text extraction expression (`CONVERTTOTEXT` or `PARSE_DOC`) in order to be capable of converting documents to text. You can use the **Expression editor** to add the necessary expression.

About identifying the document language If your pipeline requires explicitly identifying multiple source documents that may be in multiple languages, you can use the ID_LANGUAGE expression in your record manipulator.

Removing document body properties on page 177
: As a system clean up task, you may want to remove the files indicated by each record's `En¬deca.Document.Body` property. These files are no longer necessary after the text extraction expression runs.

Adding a RETRIEVE_URL expression

The `RETRIEVE_URL` expression is required to retrieve a document from its URL and store it in a file on disk.

A `STRING DIGEST` sub-expression of `RETRIEVE_URL` typically determines the name of the file in which the document is stored.`RETRIEVE_URL` places file's location into the `Endeca.Document.Body` property. Later in pipeline processing, a text extraction expression examines `Endeca.Document.Body` and converts the body content into text stored in `Endeca.Document.Text`.

Forge also places any metadata it can retrieve about the document in properties on the record.

The following properties contain values that are passed as parameters to `RETRIEVE_URL`. The property values configure additional fetching options. The `Endeca.Fetch` properties exist for a record if you provide values on the **Timeout** tab, **Proxy** tab, and **User Agent** text box of the **Spider editor**.

- `Endeca.Fetch.Timeout`
- `Endeca.Fetch.ConnectTimeout`
- `Endeca.Fetch.TransferRateLowSpeedLimit`
- `Endeca.Fetch.TransferRateLowSpeedTime`
- `Endeca.Fetch.Proxy`
- `Endeca.Fetch.UserAgent`

To add `RETRIEVE_URL` to a record manipulator:

1. If the **Expression editor** is not already open, double-click the Pipeline Diagram on the **Project** tab of Developer Studio.
2. Double-click the record manipulator.
   The **Expression editor** displays.

3. Starting at the first line in the **Expression editor**, insert a `RETRIEVE_URL` expression using the example below as a guide.

   The nested sub-expressions within `RETRIEVE_URL` configure how it functions. Here are several important points to consider when configuring `RETRIEVE_URL`:

   - A `STRING` sub-expression is required to name a file created to store the document content for a URL. Typically, you use a `STRING DIGEST` expression create a shorter property identifier (a digest) of the URL indicated by `PROP_NAME`. This digest is necessary because URLs may contain values that are invalid for use as file names.`DIGEST` creates a file name based on the URL but uses only characters a-f and numbers 0-9, so the file name is valid.
   - The `VALUE` expression node in the `CONST` expression specifies the path where the contents of each URL are stored on disk after retrieval.
   - The `PROP_NAME` expression node in the `DIGEST` expression specifies the property that contains the URL to retrieve. The default name of this property is `Endeca.Identifier`.

   Note: It is not necessary to provide attribute values for the `LABEL` or `URL` attributes.

4. Click **Check Syntax** to ensure the expressions are well formed.
5. Click **Commit Changes** and close the **Expression editor**.

The following is an example of a `RETRIEVE_URL` expression:

```
<EXPRESSION LABEL="" NAME="RETRIEVE_URL" TYPE="VOID" URL="">
 <COMMENT>Retrieve the document and store it as a temporary
 file in the state directory, named with the digest (MD5 hash)
 of its URL.</COMMENT>
 <EXPRESSION LABEL="" NAME="CONCAT" TYPE="STRING" URL="">
  <EXPRESSION LABEL="" NAME="CONST" TYPE="STRING" URL="">
   <EXPRNODE NAME="VALUE" VALUE="../partition0/state/"/>
  </EXPRESSION>
  <EXPRESSION TYPE="STRING" NAME="DIGEST">
   <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
    <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Identifier"/>
   </EXPRESSION>
  </EXPRESSION>
 </EXPRESSION>
</EXPRESSION>
```

For additional information on expression configuration, see the *Endeca Data Foundry Expressions Reference*.

About converting documents to text

An expression such as `CONVERTTOTEXT` or `PARSE_DOC` is required to extract document content from the file created by `RETRIEVE_URL` and convert the content into text.

If you are using the Endeca Document Conversion Module, you can use `CONVERTTOTEXT` to convert over 200 document types into text. If you are using Endeca IAP without the optional module, you can use `PARSE_DOC` to convert HTML and TXT documents.

After a record manipulator retrieves a URL and stores a path to the file in `Endeca.Document.Body`, a text extraction expression examines the file indicated by `Endeca.Document.Body`, extracts the document body from the file, and converts the document body into text. The text is stored by default in `Endeca.Document.Text`.

To guide text extraction and conversion, the text extraction expression refers to the `Endeca.Document.MimeType` and `Endeca.Document.Encoding` properties. If no `Endeca.Document.Encoding` exists, Forge attempts to identify the encoding automatically.

As the document body is being extracted from the file and converted to text, the expression examines the document body for any URLs. The text extraction expression adds any URLs it finds as `Endeca.Relation.References` properties to the record.

For example, if a product overview document contains links to ten product detail pages, the Endeca record for the overview document will have ten `Endeca.Relation.References` properties – one for each product detail link. When the record for this document is passed to the downstream spider component, the spider queues the URLs in each `Endeca.Relation.References` property and crawls it. This process continues until the spider component processes all URLs contained in a document.

Related Links

File Formats Supported by the Document Conversion Module on page 143
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.

About identifying the document language If your pipeline requires explicitly identifying multiple source documents that may be in multiple languages, you can use the ID_LANGUAGE expression in your record manipulator.

Generated record properties reference on page 164
The following table describes all the properties generated by various components in a crawler pipeline that can be included in your Endeca records.

Converting documents to text

A record manipulator must contain a text extraction expression (`CONVERTTOTEXT` or `PARSE_DOC`) in order to be capable of converting documents to text. You can use the **Expression editor** to add the necessary expression.

One of following text extraction expressions must be included in an Endeca Crawler pipeline:

- `CONVERTTOTEXT` expression — Extracts documents based on content-type, converts the document body to text, and extracts any URL links contained in the document. This expression uses a document conversion library to convert files from more than 200 different document types into text. The `CONVERT_EMBEDDED` option enables embedded documents to be extracted and converted. Using `CONVERTTOTEXT` subsumes the functionality of `PARSE_DOC` by doing the same extraction and conversion on HTML and TXT documents.`CONVERTTOTEXT` is only available as part of the Endeca Document Conversion Module.
- `PARSE_DOC` expression — Extracts HTML and TXT documents, converts the document body to text, and extracts any URL links contained in the document.`PARSE_DOC` is available as part of the base Endeca IAP.

For additional information on expression configuration, see the *Endeca Data Foundry Expression Reference*.

To add a text extraction expression to a record manipulator:

1. In the Pipeline diagram of Developer Studio, double-click the record manipulator. The **Expression editor** displays.

2. After the RETRIEVE_URL expression, add either the CONVERTTOTEXT or the PARSE_DOC expression using the examples below as a guide. No nested expressions or expression nodes are required, unless an option is specified.

Option	Description

For CONVERTTOTEXT:	`<EXPRESSION NAME="CONVERTTOTEXT" TYPE="VOID">`
	or to specify conversion of embedded documents:
	`<EXPRESSION TYPE="VOID" NAME="CONVERTTOTEXT"> <EXPRNODE NAME="CONVERT_EMBEDDED" VALUE="TRUE"/> </EXPRESSION>`

For PARSE_DOC:	`<EXPRESSION NAME="PARSE_DOC" TYPE="VOID">`

Note: It is not necessary to provide attribute values for the LABEL or URL attributes.

3. Click **Check Syntax** to ensure the expressions are well formed.
4. Click **Commit Changes** and close the **Expression editor**.

Removing document body properties

As a system clean up task, you may want to remove the files indicated by each record's Endeca.Document.Body property. These files are no longer necessary after the text extraction expression runs.

This is an optional task in a crawler pipeline that can occur after a text extraction expression evaluates each record.

As part of a crawler's document processing, the following two steps occur in the record manipulator:

1. RETRIEVE_URL retrieves a URL and automatically exports its contents to a file indicated by Endeca.Document.Body.
2. A text extraction expression (such as CONVERTTOTEXT or PARSE_DOC) examines the file indicated by Endeca.Document.Body, converts the contents of the file to text, and stores the text in Endeca.Document.Text.

After the text extraction expression completes, you can use a REMOVE_EXPORTED_PROP expression to remove the exported file indicated by Endeca.Document.Body, and also the Endeca.Document.Body property if desired.

To add REMOVE_EXPORTED_PROP to a pipeline:

1. In the **Pipeline** view of Developer Studio, double-click the **Record Manipulator**.
   The **Expression Editor** displays.
2. After the text extraction expression, add a REMOVE_EXPORTED_PROP expression. Some important points to consider when configuring this expression are:
   - The PROP_NAME expression node specifies the name of the property that indicates the file to remove. Typically, this is the Endeca.Document.Body property.
   - The URL expression node specifies the URL that files were written to (by RETRIEVE_URL). This value may be either an absolute path or a path relative to the location of the Pipeline.epx file.
   - The PREFIX expression node specifies any prefix used in the file name to remove.

- The REMOVE_PROPS expression node specifies whether to remove the property from the record after deleting the file where the property was stored.TRUE removes the property from the record after removing the corresponding file.FALSE does not remove the property.

Note: It is not necessary to provide attribute values for the LABEL or URL attributes.

3. Click **Check Syntax** to ensure the expressions are well-formed.
4. Click **Commit Changes** and close the **Expression editor**.

About modifying records with a Perl manipulator

Although there is no requirement that a crawler pipeline use a Perl manipulator component, this component is useful to perform more extensive record modification during processing. For example, the component can be used to strip values out of a property such as `Endeca.Document.Text` and add the values back to a record for use in dimension mapping. It can also be used to concatenate properties and add the resulting new property to a record, and so on.

For information about how to add a Perl manipulator component to a pipeline, see the *Endeca Developer Studio Help*. For information about how to implement Perl code in a Perl manipulator, see the *Endeca Forge API Guide for Perl*.

Creating a spider

Follow the steps below to set up a spider in your Endeca Crawler pipeline.

To create a spider:

1. In the **Project** tab of Developer Studio, double-click **Pipeline Diagram**.
2. In the **Pipeline Diagram editor**, choose **New** > **Spider**.
   The **New Spider editor** displays.
3. In the **Name** box, type a unique name for the spider. This should be the same name you specified as the value of URL_SOURCE when you created the record adapter.
4. To limit the number of hops from the root URL (specified on the **Root URLs** tab), enter a value in the **Maximum hops** field.

   The **Maximum hops** value specifies the number of links that may be traversed beginning with the root URL before the spider reaches the document at a target URL. For example, if `http://www.endeca.com` is a root URL and it links to a document at `http://www.endeca.com/news.html`, then `http://www.endeca.com/news.html` is one hop away from the root.
5. To limit the depth of the crawl from the root URL, enter a value in the **Maximum depth** field.

   Maximum depth is based on the number of separators in the path portion of the URL. For example, `http://endeca.com` has a depth of zero (no separators), whereas, `http://endeca.com/products/index.shtml` has a depth of one. The `/products/` portion of the URL constitutes one separator.
6. To specify the User-Agent HTTP header that the spider should present to Web servers, enter the desired value in the **Agent name** field.

   The **Agent name** identifies the name of the spider, as it will be referred to in the **User-agent** field of a Web server's `robots.txt` file. If you provide a name, the spider adheres to the `robots.txt` standard. If you do not provide a name, the spider responds only to rules in a `robots.txt` file where the value of the **User-agent** field is "`*`".

> **Note:** A `robots.txt` file allows Web-server administrators to identify robots, like spiders, and control what URLs a robot may or may not crawl on a Web server. The file specifies a robot's User-agent name and the rules associated with the name. These crawling rules configured in `robots.txt` are often known as the `robots.txt` standard or, more formally, as the Robots Exclusion Standard. For more information on this standard, see .

7. To instruct the spider to ignore the `robots.txt` file on a Web server, check **Ignore robots**.

   By ignoring the file, the spider does not obey the `robots.txt` standard and proceeds with the crawl with the parameters you configure.
8. If you want the spider to reject cookies, check **Disable Cookies**.

   If you leave this unchecked, cookie information is added to the records during the crawl, and the spider also stores and sends cookies to the server as it crawls. (When `RETRIEVE_URL` gets a `Set Cookie` header as part of its HTTP response, `RETRIEVE_URL` can pass this value back to the server, when appropriate, to simulate a session.)
9. For the full crawl described in this section, do not provide any value in the **Differential Crawl** URL box.

Related Links

Differential Crawling on page 195
This section provides an overview of differential crawling.

Spider document processing

The spider component is the core of a Endeca Crawler pipeline. Working in conjunction with a record adapter and a record manipulator, the spider forms a document-processing loop whose function is to get documents into a pipeline.

The primary function of the spider in a loop is to crawl URLs, filter URLs, send URLs to the record adapter, and manage the URL queue until all source documents are processed.

In the **Spider editor**, you can indicate the URLs to crawl, create URL filters to determine which documents to crawl, and specify timeout, proxy, and other configuration information that controls how the crawl proceeds.

Once configured and run, the spider loops through processing documents in a crawler pipeline as described in the steps below. These steps focus only on the spider's document processing loop, not the larger URL and record processing loop:

1. For the first loop of source document processing, the spider crawls the root URL indicated on the **Root URLs** tab of the **Spider editor**.
2. Based on the root URL that the spider crawls, the record adapter creates a record containing the URL, indicated by `Endeca.Identifier`, and a limited set of metadata properties.
3. The newly-created record then flows down to the record manipulator where the following takes place:
   a. The document associated with the URL is fetched (using the `RETRIEVE_URL` expression) and stored in `Endeca.Document.Body`.
   b. Content (searchable text) is extracted from `Endeca.Document.Body` (using the `CONVERTTO¬TEXT` or `PARSE_DOC` expression) and stored in `Endeca.Document.Text`.

   c. Any URLs in `Endeca.Document.Body` are extracted for additional crawling and are stored in `Endeca.Relation.References` by default.

4. The record based on the root URL moves downstream to the spider where additional URLs (those extracted from the root URL and stored in `Endeca.Relation.References`) are queued for crawling.
5. The spider crawls URLs from the record as indicated in the `Endeca.Relation.References` properties. This is the next loop of source document processing.
6. Based on the queued URL that the spider crawls, the record adapter creates a record containing the URL, indicated by `Endeca.Identifier`, and a limited set of metadata properties.
7. Steps 3 through 6 repeat until the spider processes all URLs and the record adapter creates corresponding records.

Related Links

URL and record processing on page 162

Because Developer Studio exposes crawling and text extraction functionality in the context of a pipeline, it is important to understand how this functionality fits into the Forge processing framework. The following figure shows a diagram of a full crawling pipeline.

Specifying root URLS to crawl

On the **Root URLs** tab, you provide the starting points for the spider to crawl.

Each root URL must have a scheme of `FILE`, `HTTP`, or `HTTPS`. The URL must be absolute and well-formed. A useful URL reference is available at .

In addition to starting a crawl from a root URL, you can also start a crawl by posting data to a URL if necessary. You can simulate a form post (the `HTTP POST` protocol) by specifying a root URL with post syntax and values. To construct a `POST` URL, postfix the URL with “?”, add `name=value` pairs delimited by “&”, and then add a “$” followed by the post data.

When you run the pipeline, the spider validates each root URL and checks whether the URL passes the appropriate filters, including the `robots.txt` exclusions (if the **Ignore Robots** checkbox is not set). If a root URL is invalid or does not pass any of the filters an appropriate message is logged.

To specify root URLs:

1. In the **Spider editor**, select the **Root URLs** tab.
2. In the **URL** text box, type the location from which the spider starts crawling. This value can use `FILE`, `HTTP`, `HTTPS`, or form post URLs.
3. Click **Add**.
4. Repeat steps 2 and 3 for additional locations.

For example, given this URL:

```
http-post://web01.qa:8080/qa/post/NavServlet?arg0=foo&arg1=bar$link=1/3
```

the spider executes an HTTP `POST` request to:

```
web01.qa:8080/qa/post/NavServlet
```

with query data:

```
arg0=foo&arg1=bar
```

and post data:

```
link=1/3.
```

Configuring URL extraction settings

On the **URL Configuration** tab, you can provide the name for the properties used to store queued URLs, as well as providing URL filters.

There are two folders:

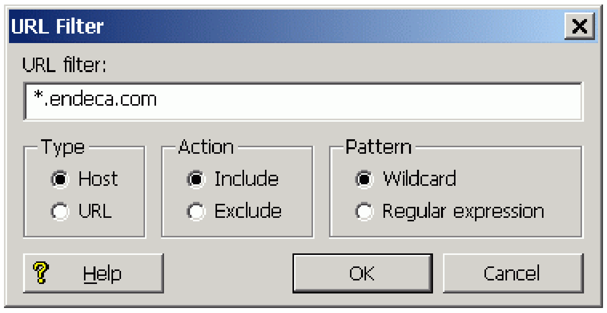
- **Enqueue URLs** — indicates the name of the property that stores links (URLs) to other documents to crawl. When a spider crawls a root URL, the spider component extracts any URLs contained on the root and adds those URLs as properties to the record. The spider queues the URLs, crawls them, and Forge creates additional records until all URLs stored in this property are processed. In a simple crawler application, you only need to specify the `Endeca.Relation.References` property here. This is the default property name produced by either text extraction expression that holds the URLs to be queued.
- **URL Filters** — specifies the filters by which the spider includes or excludes URLs during a crawl. Filters are expressed as wildcards or Perl regular expressions. URL filters are mutually exclusive; that is, URL filter A does not influence URL filter B and vice versa. At least one URL filter is required to allow the spider make additional processing loops over the root URL.

To configure URL extraction settings:

1. In the **Spider editor**, click the **URL Configuration** tab.
2. Right-click the **Enqueue URLs** folder and click **Add**.
   The **Enqueue URL editor** displays.
3. Enter a property name in the **Enqueue URL editor** that designates the property of the record that contains links to queue.
4. (Optional) Select **Remove** if you want to remove the property from the record after its value has been queued.
5. Click **OK**.
6. If necessary, repeat steps 2 through 5 to add additional queue URL properties.
7. Select the **URL Filters** folder and click **Add**.
   The **URL Filter editor** displays.
8. In the **URL Filter** text box, enter either a wildcard filter or regular expression filter.

   Filters can be specified either by using wildcard filters (for example, `*.endeca.com`) or Perl regular expressions (for example `/.*\.html/i`) Generally, you should use wildcard patterns for Host filters and use regular expression patterns for URL filters.

   This example shows a host include filter. It uses a wildcard to include all hosts that are in the `endeca.com` domain:


URL Filter

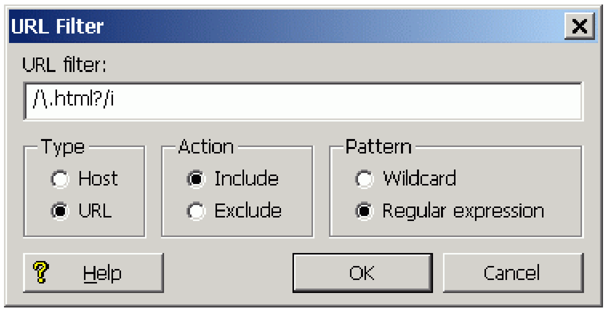
URL filter: *.endeca.com

Type: [✓ Host] URL

Action: [✓ Include] Exclude

Pattern: [✓ Wildcard] Regular expression

This example shows a URL inclusion filter that uses a regular expression filter to include all HTML files, regardless of case:



URL filter: /\.html?/i

Type: Host [✓ URL]

Action: [✓ Include] Exclude

Pattern: Wildcard [✓ Regular expression]

9. In the **Type** frame, select either **Host** or **URL**.

Option	Description
Host	Host filters apply only to the host name portion of a URL.
URL	URL filters are more flexible and can filter URLs based on whether the entire URL matches the specified pattern. For example, the spider may crawl a file system in which a directory named presentations contains PowerPoint documents that, for some reason, should not be crawled. They can be excluded using a URL exclusion filter with the pattern `/.*\/presentations\/.*\.ppt/`.

10. In the **Action** frame, select either **Include** or **Exclude**.

Option	Description
Include	Indicates that the spider crawls documents that match the URL filter.
Exclude	Indicates that the spider excludes documents that match the URL filter.

A URL must pass both inclusion and exclusion filters for the spider to queue it. In other words, a URL must match at least one inclusion filter and a URL also must not match any exclusion filter.

11. In the **Pattern** frame, select either **Wildcard** or **Regular** expression depending on the syntax of the filter you specified in step 5.
12. Repeat the steps 7 through 10 to create additional URL filters as necessary.

    At a minimum, the spider requires one host inclusion filter that corresponds to each root URL you specified on the **Root URL** tab. For example, if you set up a spider to crawl `http://endeca.com`, then the spider needs a host include filter for `endeca.com`. The filter allows the spider to include any links found on the root for additional processing. If you omit this filter, the spider processes the root URL, but not the URLs that the root contains.

Example syntax of URL filters

This section includes examples of common URL filter syntax.

- To crawl only file systems (not HTTP or HTTPS hosts), use a URL inclusion filter with a regular expression pattern of: `/^file/i`
- To crawl only documents with an .htm or .html extension, use a URL inclusion filter with a regular expression pattern of: `/\.html?$/i`
- To crawl the development branch of the Example corporate Web site, use a URL inclusion filter with a regular expression pattern of:

```
/example\.com\/dev\/.*/i
```

This pattern confines the crawler to URLs of the form:

```
example.com/dev/
```

- To restrict a crawler so that it does not crawl URLs on a corporate intranet (for example, those located on host `intranet.foo.com/dev`), use a Host exclusion filter with a regular expression pattern of:

  ```
  /intranet\.example\.com/
  ```

Specifying a record source for the spider

A spider requires an upstream pipeline component to act as its record source.

In most cases, this record source is the record manipulator that contains the `RETRIEVE_URL` and text extraction expressions. The record source could also be a record adapter or another spider.

To specify a record source:

1. In the **Spider editor**, select the **Sources** tab.
2. From the **Record source** list, choose the name of the record manipulator that you created.
3. (Optional) Specify timeouts and proxy server settings as described in the two sections that follow.
4. Click **OK** to finish creating the spider.
5. Select **File** > **Save**.

Specifying timeouts

The spider may be configured with three timeout values specified in the **Timeout** tab. These optional values control connection timeouts and URL retrieval timeouts for each URL that the spider fetches.

If you do provide values, the spider sends them with each URL to the record adapter. The record adapter generates `Endeca.Fetch` properties for each record. The property values become parameters to the `RETRIEVE_URL` expression during the fetch.

To specify timeouts:

1. In the **Pipeline Diagram editor**, double click the **Spider** component.
   The **Spider editor** displays.
2. Click the **Timeout** tab.
3. To limit the time that the spider spends retrieving a URL before aborting the fetch, type a value in the **“Maximum time spent fetching a URL”** text box.
4. To limit the time that the spider spends making a connection to a host before aborting the retrieve operation, type a value in the **“Maximum time to wait for a connection to be made”** text box.
5. If you want to abort a fetch based on transfer rate, type a value in the **Bytes/Sec for at Least** text box and the **Second** text box.

Related Links

Generated record properties reference on page 164
The following table describes all the properties generated by various components in a crawler pipeline that can be included in your Endeca records.

About specifying proxy servers

You can specify one proxy server for both HTTP and HTTPS URLs, or specify two different servers to handle the two URL types.

There are several ways to configure the spider component for use with proxy servers:

- You can specify a single proxy server, through which the spider accesses both HTTP and HTTPS URLs.
- You can specify separate proxy servers for HTTP URLs and HTTPS URLs.
- You can bypass proxy server settings for a specified URL.

You specify these settings on the **Proxy** tab of the **Spider editor**.

Specifying a single proxy server

You can configure a spider component to use a single proxy server when accessing HTTP or HTTPS URLs.

To specify a single proxy server for HTTP and HTTPS:

1. Click the **Proxy** tab.
2. Select **Use a Proxy Server to Fetch URLs** from the list.
3. In the **Host** text box of the **Proxy server** frame, type the name of the proxy server.
4. In the **Port** text box, type the port number that the proxy server listens to for URL requests from the spider.
5. If you want to bypass the specified proxy server for a URL, click the **Bypass URLs** button.
   The **Bypass URLs editor** displays.
6. Type the name of the host you want to access without the use of a proxy server and click **Add**.

   You can use wildcards to indicate a number of Web servers within a domain. Repeat this step as necessary for additional URLs.
7. Click **OK**.

Related Links

Specifying separate proxy servers on page 184
You can configure a spider component to use a separate proxy servers depending on whether it is accessing HTTP or HTTPS URLs.

Specifying separate proxy servers

You can configure a spider component to use a separate proxy servers depending on whether it is accessing HTTP or HTTPS URLs.

To specify separate proxy servers for HTTP and HTTPS:

1. On the **Proxy** tab of the **Spider editor**, select **Use Separate HTTP/HTTPS Proxy Servers** from the list.
2. In the **Host** text box of the **HTTP Proxy server** frame, type the name of the proxy server.
3. In the **Port** text box, type the port number that the proxy server listens to for HTTP URL requests from the spider.
4. In the **Host** text box of the **HTTPS Proxy server** frame, type the name of the proxy server.
5. In the **Port** text box, type the port number that the proxy server listens to for HTTPS URL requests from the spider.

6. If you want to bypass the specified proxy server for a URL, click the **Bypass URLs** button.
   The **Bypass URLs editor** displays.
7. Type the name of the host you want to access without the use of a proxy server and click **Add**.
   Repeat this step as necessary for additional URLs.
8. Click **OK** on the **Bypass URLs editor**.
9. Click **OK** on the **Spider editor**.
10. Select **File** > **Save**.

Related Links

Specifying a single proxy server on page 184

You can configure a spider component to use a single proxy server when accessing HTTP or HTTPS URLs.

About removing unnecessary records after a crawl

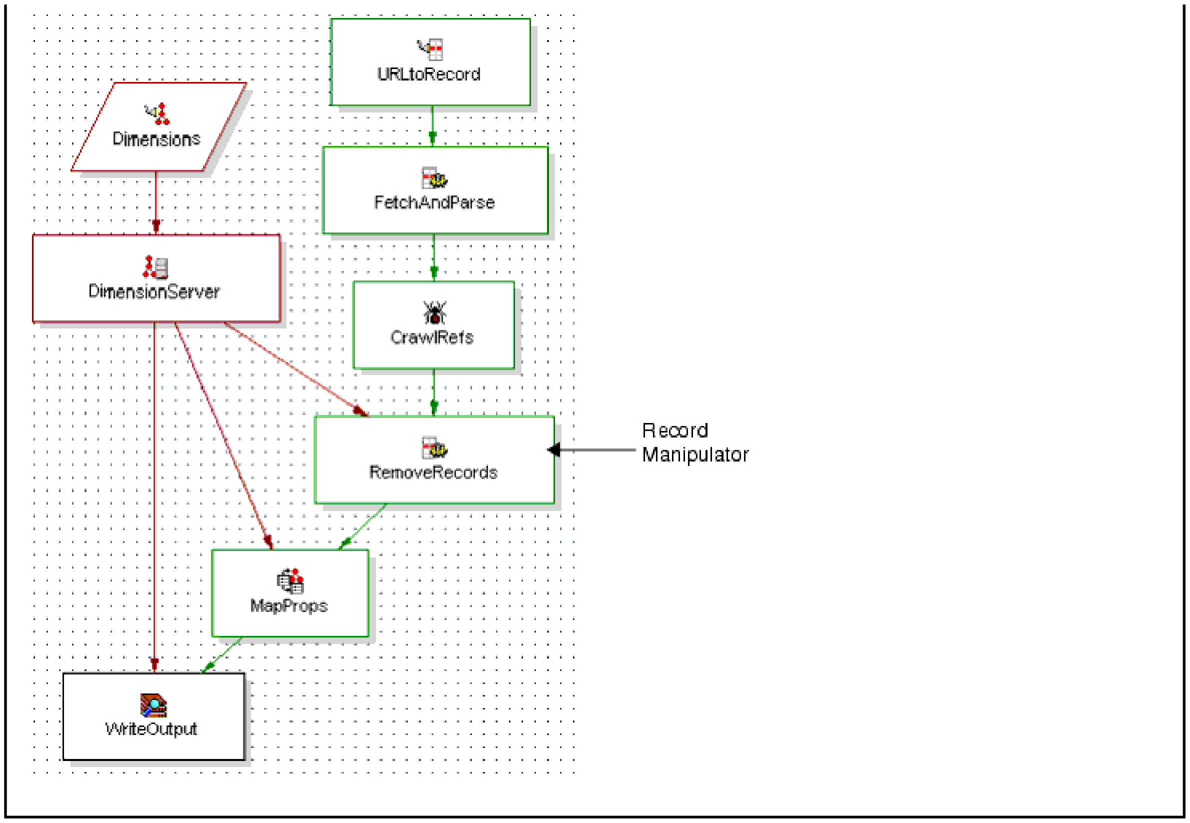
After the pipeline has processed all the source documents, you may want to remove any records that merely reflect source data structure before Forge writes out these records with an indexer adapter.

This record removal is typically necessary when records are created based on directory pages, index pages, or other forms of source documents that reflect the structure of the source data but do not correspond to a source document that you need in an application.

If you do not remove these records before indexing, the records become available to users of your Endeca application. For example, suppose a spider crawls a directory list page at `..\data\incoming\red\index.html` and creates a corresponding record. You are unlikely to want users to search the record for the `index.html` page because it primarily contains a list of links; however, the spider must crawl the index page to queue and retrieve the other pages that `index.html` links to, such as `..\data\incoming\red\product1.html`, `..\data\incoming\red\product2.html`, `..\data\incoming\red\product3.html`, and so on.

You can remove records from a pipeline using a `REMOVE_RECORD` expression. In the pipeline, the `REMOVE_RECORD` expression must appear in a record manipulator that is placed after the record processing loop. Specifically, the expression must appear after the spider component because the spider needs to crawl all URLs that may appear on a directory page.

Note the position of the RemoveRecords component in the following pipeline example:



Adding the `REMOVE_RECORD` expression to your pipeline

You can remove records after a crawl by adding a `REMOVE_RECORD` expression to your pipeline.

For additional information on expression configuration, see the *Endeca Data Foundry Expression Reference*.

To add REMOVE_RECORD to a pipeline:

1. In the **Project** tab of Developer Studio, double-click **Pipeline Diagram**.
2. In the **Pipeline Diagram editor**, click **New**.
3. Select **Record** > **Manipulator**.
   The **New Record Manipulator editor** displays.
4. In the **Name** text box, type in the name of the record manipulator.
5. From the **Record source** drop-down list, choose the name of the spider that you created.
6. From the **Dimension source** drop-down list, choose the dimension source for the pipeline.
7. To add the new record manipulator to the project, click **OK**.
8. Open the **PropertyMapper** component and change its record source to the new record manipulator you just created.
9. Select **File** > **Save**.
10. In the Pipeline Diagram, double-click the record manipulator.
    The **Expression Editor** displays.
11. Starting at the first line in the **Expression editor**, insert a `REMOVE_RECORD` expression.

REMOVE_RECORD is typically used within an IF expression to remove records that meet or do not meet certain criteria. There are no nested expressions within REMOVE_RECORD to configure how it functions.

Note: It is not necessary to provide attribute values for the LABEL or URL attributes.

12. Click **Check Syntax** to ensure the expressions are well-formed.
13. Click **Commit Changes** and close the **Expression editor**.

About configuring authentication

Forge can be configured to provide basic or HTTPS authentication, as well as client authentication or authentication for a Microsoft Exchange server.

About configuring basic authentication

When the Endeca Crawler crawls a Web site that requires basic authentication, it needs to provide the site with a valid username and password before the Web server will transmit a response. You can use a key ring file to supply Forge with an appropriate username/password pair to access a particular site that requires basic authentication.

The following is a sample key ring file that could be used to configure Forge for basic authentication:

```
<KEY_RING>
<SITE HOST="www.endeca.com" PORT="6000">
 <HTTP>
  <REALM NAME="Sales Documents">
   <KEY>BOcxV3wFSGuoBqbhPHkFGmA=</KEY>
  </REALM>
 </HTTP>
</SITE>
</KEY_RING>
```

To use this key ring file, you specify its location via the third argument of the RETRIEVE_URL expression in the Forge crawler pipeline, which is used to fetch URLs from the targeted Web server, as shown below (the relevant line is in boldface):

```
<EXPRESSION TYPE="VOID" NAME="RETRIEVE_URL">
 <!-- this expression generates a filename for
   the retrieved file: -->
 <EXPRESSION TYPE="STRING" NAME="CONCAT">
  <EXPRESSION TYPE="STRING" NAME="CONST">
   <EXPRNODE NAME="VALUE" VALUE="&cwd;"/>
  </EXPRESSION>
  <EXPRESSION TYPE="STRING" NAME="DIGEST">
   <EXPRESSION TYPE="PROPERTY" NAME="IDENTITY">
    <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Identifier"/>
   </EXPRESSION>
  </EXPRESSION>
 </EXPRESSION>
 <!-- this expression node specifies the path to the
   key ring file: -->
 <EXPRNODE NAME="KEY_RING" VALUE="key_ring.xml"/>
</EXPRESSION>
```

The path to the key ring file is expressed relative to the pipeline file or as an absolute path. In the above example, the key ring file is in the same directory as the pipeline file.

Note that the specified key ring applies only to the `RETRIEVE_URL` expression from which it is referenced.

The `KEY_RING` element

The `KEY_RING` element is the root element of the key ring file. All other components of the key ring file are contained within the `KEY_RING` element.

The `SITE` element

The `SITE` element is used to refer to a target Web site or server.

All of the directives within a `SITE` element are targeted at the site or server specified by the parent `SITE` element. For example, the `HTTP` element in the sample key ring file refers to an `HTTP` connection to the Web site on host `www.endeca.com at port 6000`.

The `SITE` element may contain one sub-element for each URL scheme by which it can be accessed. The authentication parameters for each of these schemes are specified in the body of each scheme sub-element. The two schemes that currently support authentication are HTTP and HTTPS, represented by `HTTP` and `HTTPS` elements, respectively.

The `SITE` element has one required attribute, `HOST`, and one optional attribute, `PORT`.

The `HOST` attribute

The value of the `HOST` attribute should be the fully-qualified domain name of the server that hosts the target site.

The `PORT` attribute

If the target site is not accessed via the default port for all relevant URL schemes, the `PORT` attribute can be used to specify the port explicitly. If the `PORT` attribute is unspecified, the default port for each access scheme specified will be used.

For example, the following sample key ring file would be used to specify the authentication configuration settings for accessing host `www.endeca.com` via port 80 for HTTP and port 443 for HTTPS:

```
<KEY_RING>
 <SITE HOST="www.endeca.com">
  <HTTP>
   <!-- HTTP Settings... -->
  </HTTP>
  <HTTPS>
   <!-- HTTPS Settings... -->
  </HTTPS>
 </SITE>
</KEY_RING>
```

The `HTTP` element

The `HTTP` element is used to encapsulate the basic authentication settings for accessing the parent host via HTTP.

Some parts of a site may be password-protected, while others are not. The parts of an HTTP site that require authentication are called realms.

A realm is an arbitrary name for a directory on an HTTP server and all of its contents, including subdirectories. For instance, the realm “Sales Documents” (referenced in the sample key ring file) might refer to the directory:

```
http://www.endeca.com:6000/sales/
```

which in turn contains the “contracts” and “bookings” subdirectories, each of which may contain some Word documents or Excel spreadsheets. If a Forge crawler attempted to access any of this content, including the “sales”, “contracts”, or “bookings” directories themselves, it would be prompted for a username and password to gain access to the “Sales Documents” realm.

To provide Forge with a username/password pair for accessing this realm, a `REALM` element is used. An HTTP site may have many realms, so an `HTTP` element may contain any number of `REALM` sub-elements.

The `KEY` element

The body of a `KEY` element can contain a username/password pair or a pass phrase. For protection, Forge expects the contents of a `KEY` element to be encrypted.

Related Links

About using Forge to encrypt keys and pass phrases on page 192
Forge requires the username/password pairs or pass phrases kept in `KEY` elements within the key ring file to be stored in an encrypted form which only Forge can decode.

The `REALM` element

Each `REALM` element is used to setup basic authentication for a particular named realm on the target site.

The `REALM` element has one required attribute, `NAME`, which specifies the name of the realm. The body of a `REALM` element must contain one (and only one) `KEY` element, which encapsulates the username and password combination that should be used by Forge to access the specified realm on the target site.

About configuring HTTPS authentication

HTTPS configuration is similar to HTTP authentication configuration. Forge supports HTTPS authentication of the server, client authentication with certificates, and secure communication over HTTPS.

About boot-strapping server authentication

To make an HTTPS connection, all that is often required is for Forge (as a client) to be able to authenticate the server. When Forge connects to a server via HTTPS it will attempt to validate the server’s certificate by checking its signature.

Therefore, Forge must be supplied with the public keys of the certificate authority (CA) that signed the server’s certificate. This information can be provided via a key ring file that contains a `CA_DB` element, as in this example:

```
<KEY_RING>
<CA_DB>eneCA.pem</CA_DB>
```

```
<SITE HOST="www.endeca.com" PORT="6000">
 <HTTP>
  <REALM NAME="Sales Documents">
   <KEY>BOcxV3wFSGuoBqbhPHkFGmA=</KEY>
  </REALM>
 </HTTP>
</SITE>
</KEY_RING>
```

The CA_DB element

The body of a `CA_DB` element specifies the path to a PEM format certificate which contains one or more public keys that Forge should use to validate the CA signatures it encounters on server certificates when it retrieves URLs via HTTPS.

The path to this certificate may be relative to the parent pipeline XML file or an absolute path.

If Forge is unable to find the public key of the CA that signed a server certificate that it receives when attempting to initiate an HTTPS transfer, it will fail to retrieve the requested document and report an error. If a certificate chain is necessary to validate the server certificate, the public key of each CA along the chain must be present in the `CA_DB` in order for host authentication to succeed.

About disabling server authentication for a host

By default, Forge always attempts to validate CA signatures for every HTTPS host. However, host authentication can be disabled for an individual host by setting the `AUTHENTICATE_HOST` attribute of the appropriate HTTPS element in the key ring to `FALSE`.

Related Links

The AUTHENTICATE_HOST attribute on page 190
The `HTTPS` element has one optional attribute, `AUTHENTICATE_HOST`. This attribute specifies whether or not to verify the CA signature of server certificates received from the target host.

The `HTTPS` element

The `HTTPS` element is the analog of the `HTTP` element. It encapsulates the HTTPS configuration information that applies to a particular site, which is defined by the `HTTPS` element's parent `SITE` element.

The `AUTHENTICATE_HOST` attribute

The `HTTPS` element has one optional attribute, `AUTHENTICATE_HOST`. This attribute specifies whether or not to verify the CA signature of server certificates received from the target host.

By default, the value of this attribute is `TRUE`. To disable host authentication for HTTPS connections to the target host, set this attribute to `FALSE`, as shown below:

```
<HTTPS AUTHENTICATE_HOST="FALSE"/>
```

About configuring client authentication

In order for Forge to be able to connect to a server that requires client authentication, it must be supplied with an appropriate client certificate as well as an associated private key.

Some HTTPS servers may require clients to authenticate themselves. A client does this by presenting a certificate that has been signed by a CA that the server trusts. Forge can be supplied with a certificate and private key, as in the following example:

```
<KEY_RING>
<CA_DB>cacert.pem</CA_DB>
<SITE HOST="www.endeca.com" PORT="6000">
 <HTTPS>
  <CERT PATH="clientcert.pem" PRIV_KEY_PATH="clientkey.key">
   <KEY>AqS6+A3u+ivX</KEY>
  </CERT>
 </HTTPS>
</SITE>
</KEY_RING>
```

The `CERT` element

The `CERT` element has two required attributes, `PATH` and `PRIV_KEY_PATH`, which specify the locations of the certificate and private key.

One `CERT` element can be inserted in the body of an `HTTPS` element to bootstrap the HTTPS connection with a certificate and corresponding private key for a site that requires client authentication.

If these files are protected by a pass phrase, the pass phrase can be provided in the body of a `KEY` child element of the `CERT` element, as in the example below:

```
<KEY_RING>
<CA_DB>cacert.pem</CA_DB>
<SITE HOST="www.endeca.com" PORT="6000">
 <HTTPS>
  <CERT PATH="clientcert.pem" PRIV_KEY_PATH="clientkey.key">
   <KEY>AqS6+A3u+ivX</KEY>
  </CERT>
 </HTTPS>
</SITE>
</KEY_RING>
```

As with HTTP username/password keys, Forge expects a key for `CERT` to be stored in an encrypted form.

Related Links

About using Forge to encrypt keys and pass phrases on page 192
: Forge requires the username/password pairs or pass phrases kept in `KEY` elements within the key ring file to be stored in an encrypted form which only Forge can decode.

The `PATH` attribute

The `PATH` attribute of a `CERT` element specifies the location of the certificate file. The path may be expressed relative to the pipeline file or as an absolute path, and the certificate must be stored in the PEM format.

The `PRIV_KEY_PATH` attribute

The `PRIV_KEY_PATH` attribute specifies the path to a PEM format file containing the private key associated with the certificate referenced in the `PATH` element. This path may be expressed relative to the pipeline file or as an absolute path.

About authenticating with a Microsoft Exchange server

A key ring file may also be used to specify authentication configuration for a Microsoft Exchange Server when using a record adapter with an `EXCHANGE` format.

The Exchange server will expect a valid username and password combination, which may be specified via a `KEY` element embedded in an `EXCHANGE_SERVER` element within a key ring, as in the following example:

```
<KEY_RING>
<EXCHANGE_SERVER HOST="exchange.mycompany.com">
 <KEY>B9qtQOON6skNTFTHm9rnn04=</KEY>
</EXCHANGE_SERVER>
</KEY_RING>
```

The `EXCHANGE_SERVER` element

This element opens a block of configuration for authenticating to an Exchange server. It has one required attribute, the `HOST` attribute, which specifies the name of the Exchange server the supplied configuration information applies to.

About authenticating with a proxy server

A key ring file may be used to specify authentication configuration for proxy servers.

Note: Basic authentication is the only method supported by Forge for authenticating with proxy servers.

The proxy server will expect a valid username and password combination, which may be specified via a `KEY` element embedded in a `PROXY` element within a key ring, as in the following example:

```
<KEY_RING>
<PROXY HOST="proxy.mycompany.com" PORT="8080">
 <KEY>J9dtQOOR6skPTFTHm5rnn08=</KEY>
</PROXY>
</KEY_RING>
```

The `PROXY` element

The `PROXY` element contains configuration for proxy authenticating. It has two required attributes.

The `HOST` attribute specifies the host name of the proxy server for which the supplied configuration information applies.

The `PORT` attribute specifies the port number on the proxy host (specified in the `HOST` attribute) for which the supplied configuration information applies.

About using Forge to encrypt keys and pass phrases

Forge requires the username/password pairs or pass phrases kept in `KEY` elements within the key ring file to be stored in an encrypted form which only Forge can decode.

Forge provides a command-line argument, `--encryptKey`, which should be used to put the contents of `KEY` elements in this form. The encrypt key flag has the following syntax:

```
forge --encryptKey [username:]passphase
```

Encrypting a username/password pair

The following example shows how to run Forge to encrypt a username/password pair (username=`sales`, password=`endeca`) for use in an `HTTP` block of a key ring file:

```
forge --encryptKey sales:Endeca
```

As the example illustrates, the username and password must be entered together, separated by a colon, as the argument to the `--encryptKey` flag. Forge then outputs the encrypted key, which you then insert in the body of the applicable `KEY` element.

Encrypting a pass phrase

To encrypt the pass phrase "burning down the house" Forge should be executed with the following command:

```
forge --encryptKey "burning down the house"
```

Appendix D

Differential Crawling

This section provides an overview of differential crawling.

Overview of a differential crawling pipeline

Conceptually, a differential crawl is similar to a full crawl, with the exception that a differential crawl will only download those documents that have been modified since the previous crawl.

The differential crawling cannot be run from Developer Studio or Endeca Workbench. It can be run only via a control script. For a script example, see the *Endeca Control System Guide*.

Important: The Endeca Crawler is deprecated, and will be removed in a future version of the Endeca Information Access Platform. Therefore, if you are beginning a new project, it is recommended that you use the Endeca Web Crawler, which is a component of the Endeca Content Acquisition System.

This section assumes that you have read “Implementing the Endeca Crawler," and that you are familiar with creating a full crawl pipeline.

Related Links

The Endeca Crawler on page 157
This section describes how to configure and run the Endeca Crawler.

The full crawling pipeline on page 170
These sections describe how to create and configure a full crawling pipeline using Developer Studio.

About enabling differential crawling for the spider

Both full crawl and differential pipelines must have a spider component. The main configuration difference is that the differential spider has a URL specified in the **“Differential crawl URL”** field of the **Spider editor**.

When a URL is specified for this field, instead of performing a full crawl every time the pipeline is run, the spider will only download those documents that have been modified since the last run.

The spider determines which documents have changed by maintaining a state file (at the “Differential crawl URL” location). This state file contains the results of the previous pipeline run, and is compared to the results of the current run. The spider will fetch the document's headers, such as size and date modified; if these are different than what is in the state file, the entire document is downloaded. If these are not different, the document is not downloaded.

Note: The crawler supports the HTTP/1.1 specification, such as the `Cache-Control` and `Pragma` directives and the `If-Modified-Since` header field.

This means that the output of the spider contains a record for every document, but only the bodies of those documents that have been modified since the last spider run. If this is used as the only record source for an MDEX Engine, data will be missing, since unchanged documents will have no bodies. The solution is to join previously-crawled data with the new data.

Related Links

About joining previously-crawled data on page 196
: To include the bodies of unchanged documents, a previous full crawl's output must be joined back into the pipeline (Forge's output can be fed directly into a record adapter using the "binary" format).

Differential spider on page 201
: A differential crawl spider is configured in the same way as a full crawl spider, with the exception of the **Differential Crawl URL** field.

About joining previously-crawled data

To include the bodies of unchanged documents, a previous full crawl's output must be joined back into the pipeline (Forge's output can be fed directly into a record adapter using the "binary" format).

The entire pipeline then has two sources of input:

- the differential crawl
- output from the previous run

These two sources can be joined with a First Record join in a record assembler, with the differential crawl taking priority over the previous crawl, such that newly modified documents take precedence over old versions of the same document.

Keep in mind, however, that the differential crawl contains a record for every document, including those that have not changed. If this is sent directly into the First Record join, it will override the previous crawler's information. Thus, a record manipulator must be placed ahead of the differential crawl, but before the join. This record manipulator must remove any records where the `Endeca.Document.IsUn¬changed` property exists and is `true`, or `Endeca.Document.Status` exists and is “Fetch Skipped”. `Endeca.Document.IsUnchanged` will be `true` when metadata about the document has changed but the content has not; `Endeca.Document.Status` will be “Fetch Skipped” when neither the metadata nor the content has changed.

Related Links

First record join on page 72
: 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.

About removing invalid content

The differential crawl will contain a record for documents that previously existed, but have now disappeared (or are no longer valid if the parameters of the spider have changed). These records will have an `Endeca.Document.Status` property equal to “Fetch Failed” or “Fetch Aborted” and must be removed from the output.

It is recommended to do this after the join, so that all references to a document that no longer exists are eliminated (and thus the final output can be used as input for the next run). Another record manipulator must be placed after the join to remove these records.

It is also recommended to remove those records where the `Endeca.Document.IsRedirection` property exists and is `true`; these typically do not have value within search indexes. This is true of all crawlers, and is not necessary to enable differential crawling. Note that these records should be removed after the join.

How the Endeca.Document.IsUnchanged property is set

Differential crawls generate the `Endeca.Document.IsUnchanged` property.

The setting of the `Endeca.Document.IsUnchanged` property indicates whether a document is considered to be changed or not.

At the highest level, the crawler tries to conform to the HTTP/1.1 specification when determining whether it considers a document to be changed. In general, the rules are:

- Whether a document is re-fetched is determined by a combination of header fields, including such directives as `Expires`, `Last-Modified`, `Max-Age`, and `Cache-Control`.
- Metadata for a document is fetched with the document, stored by the spider, and used by the `RETRIEVE_URL` expression.
- If a document sets the `Cache-Control` metatag to “must-revalidate”, this metadata will be re-fetched during each crawl. Otherwise, the stored version is used.
- If a document sets the `Cache-Control` metatag to “no-cache” or “no-store”, or if it sets the `Pragma` field to “no-cache”, the `IsUnchanged` property is set to `false`.
- If the date the document was fetched, plus the `Max-Age`, is less than the current date, the `IsUnchanged` property is set to `true`.
- The Endeca software locally computes the approximate current time on the server from which the document was fetched. If the computed time is less than the `Expires` date, the `IsUnchanged` property is set to `true`. Otherwise, the spider will consider the `IsUnchanged` property to `false`.
- The `RETRIEVE_URL` expression, then checks the `Last-Modified` date on the document with the existing revision; if the remote modification date is not after the `Last-Modified` date on the local document, the `IsUnchanged` property is set to `true`.

An important caveat to keep in mind is that because of the way that the above rules are implemented, it is possible that a re-fetch of a document can be skipped without ever checking with the server to see whether the document has changed. For example, you can manually edit a document, but it is possible that the `Endeca.Document.IsUnchanged` property may remain set to `true`.

Caveats for differential crawling

Because differential crawls depend heavily on page HTTP headers (such as `content size` and `date`), it is critical that the server being crawled produce accurate, differential crawl-friendly metadata.

There are server configurations that are not differential crawl-friendly. One example is a content management system that republishes its pages on a nightly basis. Although the relevant content within the majority of pages does not change, the metadata does change. Also, non-critical text within the page's content (such as the current date) may be updated. This kind of nightly publishing changes enough information in each document that the differential crawler believes it is a new document and downloads it every night. Thus, the benefits of the differential crawl are lost.

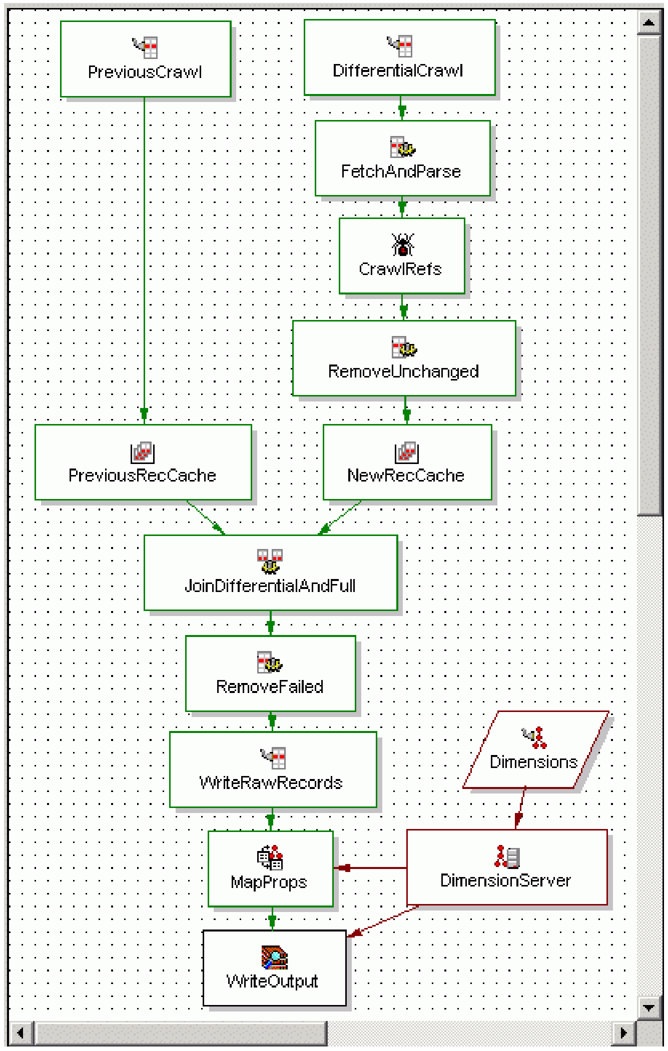
Another example is a dynamic site containing changing data, but not changing metadata. The dynamic site may be pulling constantly updated information from a database, but the server issues unchanged metadata. While the differential crawl should recognize content changes alone, it is possible that some changed documents will not be downloaded.

The best way to determine if you have one of these server configurations is by diligent testing of the crawl results. Such testing can include temporarily adding a record manipulator in each of the full and differential source streams. The manipulator would add a property (named `DifferentialStatus`) on each record with a value of “Cached” (if the record is from a previous crawl) or “Fresh” (if the record is from the current crawl). The property could then be mapped to a dimension, with refinement statistics, that details the number of documents that have been downloaded fresh versus cached from the previous crawl.

Sample differential crawl pipeline

This section includes a diagram and overview of a sample differential crawl pipeline.

The following Pipeline Diagram shows the contents of the pipeline:



The table below provides brief descriptions of the components:

Component Name	Description
PreviousCrawl	Input record adapter for records in the previous crawl.
DifferentialCrawl	Input record adapter for records in the current crawl.
FetchAndParse	Record manipulator that downloads and parses URLs discovered during crawling.
CrawlRefs	Spider component that enqueues and follows URLs discovered during crawling.
RemoveUnchanged	Record manipulator that removes any unchanged records from the differential crawl.
PreviousRecCache	Record cache for records in the previous crawl that will feed the join.

Component Name	Description
NewRecCache	Record cache for records in the current crawl that will feed the join.
JoinDifferentialAndFull	Record assembler that performs a First Record join between the differential crawl and the previous full crawl.
RemoveFailed	Record manipulator that removes any invalid records.
WriteRawRecords	Output record adapter that saves the raw records, before property mapping, of the join between the differential crawl and the previous crawl.
MapProps	Property mapper that maps source properties into Endeca properties and dimensions.
WriteOutput	Indexer adapter that prepares output for Dgidx.
Dimensions	Dimension adapter providing the dimension source.
DimensionServer	Dimension server.

Note: Although you can have two pipelines in your project (one that performs only full crawls and the other dedicated to differential crawls), it is simpler to have one pipeline that can perform both types of crawls. This is the type of pipeline used in this sample implementation.

Record adapters

The sample pipeline has two input record adapters and one output record adapter.

Setting the DifferentialCrawl input record adapter

The DifferentialCrawl input record adapter reads the documents associated with the URLs provided by the CrawlRefs spider component, and creates a record for each document.

To enter the correct settings for the DifferentialCrawl input record adapter:

1. In the **Record Adapter editor**, select the **General** tab.
2. Ensure **Direction** is set to `Input`.
3. Ensure **Format** is set to `Document`.
4. Leave the **URL** field blank.
5. Leave the **Encoding** option blank.
   This record adapter does not read from a file, but rather from the spider.
6. Select the **Pass Throughs** tab.

7. In the **Name** text box, enter `URL_SOURCE`
8. In the **Value** text box, enter the name of the spider component (such as CrawlRefs).

Setting the PreviousCrawl input record adapter

The PreviousCrawl input record adapter reads the records found in the previous crawl. The information found here is used to populate those URLs that the differential crawl did not download.

To enter the correct settings for the PreviousCrawl input record adapter:

1. In the **Record Adapter editor**, select the **General** tab.
2. Ensure **Direction** is set to `Input`.
3. Ensure **Format** is set to `Binary`.
4. In the **URL** field, enter the path from the **URL** field of the WriteRawRecords record adapter (i.e., `../partition0/state/previouscrawl.records.binary`).

   That is, the input of this component is the output of the WriteRawRecords component.
5. Leave the **Encoding** option blank

   This setting is ignored for binary record adapters. The other tabs may be left in their default state.

Setting the WriteRawRecords output record adapter

The WriteRawRecords output record adapter saves the raw records, before property mapping, of the join between the differential crawl and the previous crawl.

The output of this adapter is used to feed the input of the PreviousCrawl component upon the next run of the pipeline.

To enter the correct settings for the WriteRawRecords output record adapter:

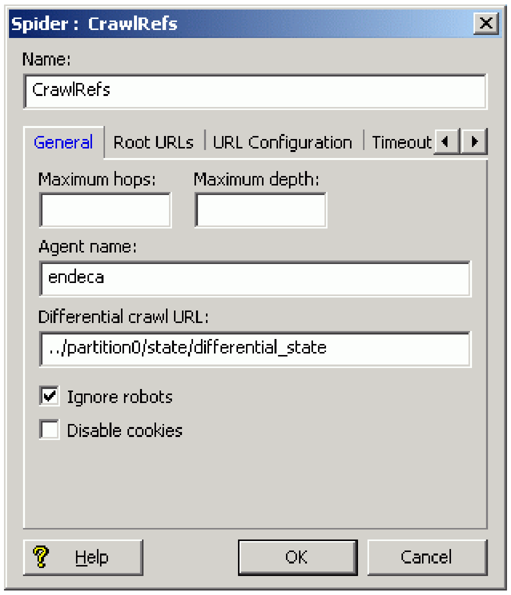
1. In the **Record Adapter editor**, select the **General** tab.
2. Ensure **Direction** is set to `Output`.
3. Ensure **Format** is set to `Binary`.
4. In the **URL** field, enter the pathname of the adapter's binary output file, such as `../partition0/state/previouscrawl.records.binary`.
5. Leave the **Encoding** option blank.

   This setting is ignored for binary record adapters.
6. Select the **Sources** tab.
7. For **Record Source**, select the RemoveFailed record manipulator.
8. For **Dimension Source**, use the `None` default setting.

Differential spider

A differential crawl spider is configured in the same way as a full crawl spider, with the exception of the **Differential Crawl URL** field.

This field must be filled in, as in this example:



Name: CrawlRefs

Maximum hops:

Maximum depth:

Agent name: endeca

Differential crawl URL: ../partition0/state/differential_state

[✓] Ignore robots

[ ] Disable cookies

The URL specifies the location of the state file that contains the results of the previous pipeline run. With that one exception, the CrawlRefs spider is configured identically to the full crawl spider.

Related Links

Creating a spider on page 178
Follow the steps below to set up a spider in your Endeca Crawler pipeline.

About enabling differential crawling for the spider on page 195
Both full crawl and differential pipelines must have a spider component. The main configuration difference is that the differential spider has a URL specified in the **"Differential crawl URL"** field of the **Spider editor**.

Setting Record Caches

The PreviousRecCache and NewRecCache record caches feed the record assembler for the First Record join.

To enter the correct settings for the each record cache:

1. In the **Record Cache editor**, select the **General** tab.
2. Ensure **Maximum Records** is set to `-1` so that the cache will load all records.
3. Leave **Combine Records** unchecked.
4. Select the **Sources** tab.
5. Set the **Record Source** value:
   - For PreviousRecCache, use the PreviousCrawl record adapter.
   - For NewRecCache, use the RemoveUnchanged record manipulator.
6. Set the **Dimension Source** value to `None`.
7. Select the **Record Index** tab.
8. For **Record Index Key**, add the `Endeca.Identifier.MD5` property.
9. Check **Discard records with duplicate keys**.

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.

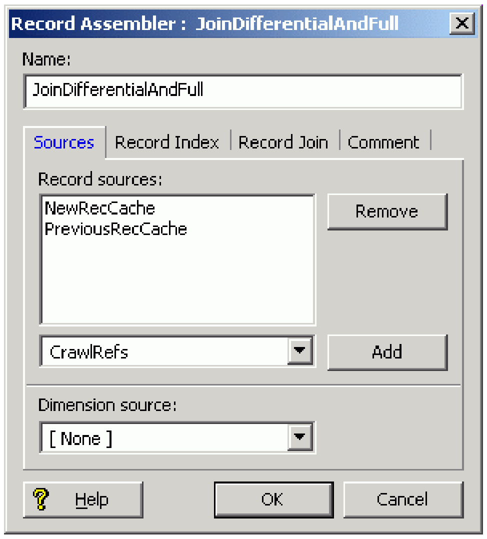
Record assembler

The JoinDifferentialAndFull component is a record assembler that performs a First Record join between the current and previous crawls.

In this first record join, if a record from the current crawl compares equally to a record from the previous crawl, the record from the current crawl is processed and the record from the previous crawl is discarded.

Sources tab of the record assembler

The **Sources** tab of the **Record Assembler editor** should look like this:


Record Assembler : JoinDifferentialAndFull

Name: JoinDifferentialAndFull

Record sources:
NewRecCache
PreviousRecCache

CrawlRefs

Dimension source: [None]

The two record sources are the PreviousRecCache and NewRecCache record caches. The order in which you add these record sources is not important, because their priority is set in the **Record Join** tab (below).

Note also that no dimension source is specified for the assembler.

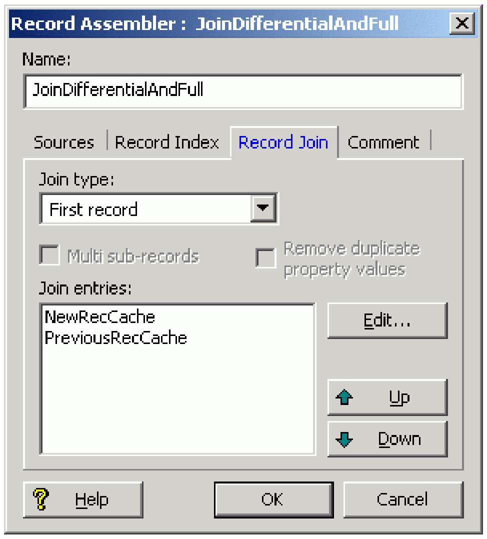
Record Index tab of the record assembler

The **Record Index** tab settings should be identical to those of the record caches:

- The `Endeca.Identifier.MD5` property is the record index key.
- The **Discard records with duplicate keys** checkbox should be marked.

Record Join tab of the record assembler

The **Record Join** tab should look like this:


Record Assembler : JoinDifferentialAndFull

Name: JoinDifferentialAndFull

Join type: First record

Join entries:
NewRecCache
PreviousRecCache

The selected join type is `First record`. Note that the NewRecCache component (which is the source for records from the current crawl) has a higher priority than the PreviousRecCache component (the source for records from the previous crawl).

Related Links

First record join on page 72

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.

Record manipulators

The pipeline has two record manipulators, named RemoveUnchanged and RemoveFailed. The FetchandParse record manipulator is not described in this appendix because it is identical to the record manipulator created for a full crawl.

Related Links

Creating a record manipulator on page 173

Expressions in a record manipulator perform document retrieval, text extraction, language identification, record or property clean up, and other tasks related to crawling. These expressions are evaluated against each record as it flows through the pipeline, and the record is changed as necessary.

RemoveUnchanged record manipulator

The RemoveUnchanged manipulator is placed after the CrawlRefs spider and before the NewRecCache record cache. It removes unaltered records so that they are not processed.

The RemoveUnchanged manipulator removes a record from a differential crawl if either of these conditions is true:

- The `Endeca.Document.Status` property has a value of "Fetch Skipped". In this case, neither the metadata nor content of the record has changed.

- The `Endeca.Document.IsUnchanged` property has a value of "`true`". In this case, the metadata of the record changed, but the content did not.

Two `IF` expressions implement the above logic. The first `IF` expression tests the value of the `Ende¬ca.Document.Status` property and removes a skipped record, as follows:

```
<EXPRESSION LABEL="" NAME="IF" TYPE="VOID" URL="">
  <COMMENT>Remove the record if it has been flagged as skipped;
  neither metadata nor content has changed.</COMMENT>
  <EXPRESSION LABEL="" NAME="PROP_EXISTS" TYPE="INTEGER" URL="">
    <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Document.Status"/>
  </EXPRESSION>
  <EXPRESSION LABEL="" NAME="IF" TYPE="VOID" URL="">
    <EXPRESSION LABEL="" NAME="MATH" TYPE="INTEGER" URL="">
      <EXPRNODE NAME="TYPE" VALUE="STRING"/>
      <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
      <EXPRESSION LABEL="" NAME="CONST" TYPE="STRING" URL="">
        <EXPRNODE NAME="VALUE" VALUE="Fetch Skipped"/>
      </EXPRESSION>
      <EXPRESSION LABEL="" NAME="IDENTITY" TYPE="PROPERTY" URL="">
        <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Document.Status"/>
      </EXPRESSION>
    </EXPRESSION>
    <EXPRESSION LABEL="" NAME="REMOVE_RECORD" TYPE="VOID" URL=""/>
  </EXPRESSION>
</EXPRESSION>
```

The second `IF` expression evaluates the `Endeca.Document.IsUnchanged` property to remove unchanged records, as follows:

```
<EXPRESSION LABEL="" NAME="IF" TYPE="VOID" URL="">
  <COMMENT>Remove the record if it has been flagged as unchanged;
  the metadata changed, but the content did not.</COMMENT>
  <EXPRESSION LABEL="" NAME="PROP_EXISTS" TYPE="INTEGER" URL="">
    <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Document.IsUnchanged"/>
  </EXPRESSION>
  <EXPRESSION LABEL="" NAME="IF" TYPE="VOID" URL="">
    <EXPRESSION LABEL="" NAME="MATH" TYPE="INTEGER" URL="">
      <EXPRNODE NAME="TYPE" VALUE="STRING"/>
      <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
      <EXPRESSION LABEL="" NAME="CONST" TYPE="STRING" URL="">
        <EXPRNODE NAME="VALUE" VALUE="true"/>
      </EXPRESSION>
      <EXPRESSION LABEL="" NAME="IDENTITY" TYPE="PROPERTY" URL="">
        <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Document.IsUnchanged"/>
      </EXPRESSION>
    </EXPRESSION>
    <EXPRESSION LABEL="" NAME="REMOVE_RECORD" TYPE="VOID" URL=""/>
  </EXPRESSION>
</EXPRESSION>
```

After you create the RemoveUnchanged record manipulator, you can enter the above expressions in the **Expression editor**.

Note: The record manipulator has no dimension source and no record index key settings.

RemoveFailed record manipulator

The RemoveFailed manipulator removes URLs that failed in the most recent run. It is placed after the JoinDifferentialAndFull record assembler and before the WriteRawRecords output record adapter.

The manipulator uses the following logic:

- If the `Endeca.Document.Status` property has a value of “Fetch Failed”, remove the record. In this case, the record existed in a previous iteration of the crawl but no longer exists.
- If the `Endeca.Document.Status` property has a value of “Fetch Aborted”, remove the record. In this case, the fetch attempt was unsuccessful.
- The `Endeca.Document.IsRedirection` property has a value of “`true`”, remove the record. In this case, the document is a redirection to another document, and is therefore not useful.

Three `IF` expressions implement the above logic. The first `IF` expression tests the `Endeca.Docu¬ment.Status` property for a “Fetch Failed” value and removes the record if `true`, as follows:

```
<EXPRESSION LABEL="" NAME="IF" TYPE="VOID" URL="">
      <COMMENT>If Endeca.Document.Status == "Fetch Failed", then remove the

    record. This will occur if the record existed in a previous iteration
    of the crawl but no longer exists.
   </COMMENT>
      <EXPRESSION LABEL="" NAME="PROP_EXISTS" TYPE="INTEGER" URL="">
        <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Document.Status"/>
      </EXPRESSION>
      <EXPRESSION LABEL="" NAME="IF" TYPE="VOID" URL="">
        <EXPRESSION LABEL="" NAME="MATH" TYPE="INTEGER" URL="">
          <EXPRNODE NAME="TYPE" VALUE="STRING"/>
          <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
          <EXPRESSION LABEL="" NAME="CONST" TYPE="STRING" URL="">
            <EXPRNODE NAME="VALUE" VALUE="Fetch Failed"/>
          </EXPRESSION>
          <EXPRESSION LABEL="" NAME="IDENTITY" TYPE="PROPERTY" URL="">
            <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Document.Status"/>
          </EXPRESSION>
        </EXPRESSION>
        <EXPRESSION LABEL="" NAME="REMOVE_RECORD" TYPE="VOID" URL=""/>
      </EXPRESSION>
</EXPRESSION>
```

The second `IF` expression tests the value of the `Endeca.Document.Status` property, this time for a “Fetch Aborted” value, as follows:

```
<EXPRESSION LABEL="" NAME="IF" TYPE="VOID" URL="">
  <COMMENT>If Endeca.Document.Status == "Fetch Aborted", then
  remove the record.</COMMENT>
  <EXPRESSION LABEL="" NAME="PROP_EXISTS" TYPE="INTEGER" URL="">
    <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Document.Status"/>
  </EXPRESSION>
  <EXPRESSION LABEL="" NAME="IF" TYPE="VOID" URL="">
    <EXPRESSION LABEL="" NAME="MATH" TYPE="INTEGER" URL="">
      <EXPRNODE NAME="TYPE" VALUE="STRING"/>
      <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
      <EXPRESSION LABEL="" NAME="CONST" TYPE="STRING" URL="">
        <EXPRNODE NAME="VALUE" VALUE="Fetch Aborted"/>
      </EXPRESSION>
      <EXPRESSION LABEL="" NAME="IDENTITY" TYPE="PROPERTY" URL="">
        <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Document.Status"/>
      </EXPRESSION>
    </EXPRESSION>
    <EXPRESSION LABEL="" NAME="REMOVE_RECORD" TYPE="VOID" URL=""/>
  </EXPRESSION>
</EXPRESSION>
```

The third `IF` expression tests the `Endeca.Document.IsRedirection` property and removes the record if `true`, as follows:

```
<EXPRESSION LABEL="" NAME="IF" TYPE="VOID" URL="">
  <COMMENT>If this document is a redirection, remove it; it has
  no value to us.
</COMMENT>
  <EXPRESSION LABEL="" NAME="PROP_EXISTS" TYPE="INTEGER" URL="">
    <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Document.IsRedirection"/>
  </EXPRESSION>
  <EXPRESSION LABEL="" NAME="IF" TYPE="VOID" URL="">
    <EXPRESSION LABEL="" NAME="MATH" TYPE="INTEGER" URL="">
      <EXPRNODE NAME="TYPE" VALUE="STRING"/>
      <EXPRNODE NAME="OPERATOR" VALUE="EQUAL"/>
      <EXPRESSION LABEL="" NAME="CONST" TYPE="STRING" URL="">
        <EXPRNODE NAME="VALUE" VALUE="true"/>
      </EXPRESSION>
      <EXPRESSION LABEL="" NAME="IDENTITY" TYPE="PROPERTY" URL="">
        <EXPRNODE NAME="PROP_NAME" VALUE="Endeca.Document.IsRedirection"/>

      </EXPRESSION>
    </EXPRESSION>
    <EXPRESSION LABEL="" NAME="REMOVE_RECORD" TYPE="VOID" URL=""/>
  </EXPRESSION>
</EXPRESSION>
```

Note: The record manipulator has no dimension source and no record index key settings.

Index

A

adding components to a pipeline 44
authentication
    boot-strapping 189
    client 191
    configuring basic 187
    disabling for a host 190
    HTTPS 189
    Microsoft Exchange server 192
    proxy server 192
Auto Generate mode
    described 36
    saving state information for 50

B

basic pipeline
    dimension adapter 49
    dimension server 50
    indexer adapter 51
    property mapper 51
    record adapter 48
    testing 53
boot-strapping server authentication 189

C

client authentication 191
combine joins 73
Combine Records setting in record caches 88
component names as used in a pipeline 44
converting documents to text
    Endeca Crawler 175, 176
CONVERTTOTEXT expression 176
crawl types in the Endeca Crawler 161
crawler errors 160
creating a spider 178

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 32
Default Maximum Length 41
    override 41
Developer Studio 18
Dgidx
    introduced 14
    running 17
Dgraph, running the 17
differential crawling
    caveats 198
    input record adapter 200, 201
    joining previously-crawled data 196
    output record adapter 201
    record adapters 200
    record assembler 203
    record cache 202
    record manipulators 204, 206
    removing invalid content 197
    sample pipeline 198
    spider configuration 195, 201
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
    implicit mapping 40
    Must Match mode 36
    Normal match mode 35
    priority order for advanced techniques 28
    source properties to like-named dimensions 40
    synonyms 57
    viewing existing 30
dimension search configured in Developer Studio 18
dimension server
    for persisting auto-generated dimensions 50
    overview 50
dimension values
    auto generating 36
    mapping to source property values 24
    specifying the order of 59
dimensions
    assigning multiple mappings to 32
    creating 56
    mapping to source properties 24
    specifying the order of 59
directory structure for the Endeca Application Controller 43
disjunct joins 70
dynamic business rules 60
    configuring in Developer Studio 18
    configuring in Endeca Workbench 18

E

emgr_update utility 104
encrypting keys with Forge 192
Endeca Application Controller
  architecture 20
  communicating with 20
  communicating with Endeca Workbench 21
  directory structure 43
  introduced 19
Endeca CAS 13
Endeca Crawler
  about removing unnecessary records after a crawl 185
  configuring for differential crawling 195
  converting documents to text 175, 176
  crawl types 161
  crawler errors 160
  creating a full crawling pipeline 170
  creating a record adapter to read documents 172
  creating a record manipulator 173
  creating a spider 178
  Document Conversion module
    other supported formats 155
    supported compressed formats 152
    supported database formats 153
    supported e-mail formats 154
    supported graphics formats 148
    supported presentation formats 152
    supported spreadsheet formats 147
    supported word processing formats 143
  implementation overview 171
  installing 157
  introduced 157
  Perl manipulator 178
  properties 164
  property name syntax 163
  redundant URLs 162
  removing document body properties 177
  RETRIEVE_URL expression 174
  root URL extraction settings 181
  security information 161
  specifying root URLs 180
  spider document processing 179
  supporting components 171
  transforming source documentation to Endeca records 158
  URL and record processing 162
  URL filters 182
  viewing all generated properties 170
Endeca Developer Studio
  creating a basic pipeline project 47
  creating and mapping dimensions 56
  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
Endeca ITL *(continued)*
  indexing
    about 24
  indexing with 13, 14, 24
  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 32
  creating 56
Endeca Tools setup information 19
Endeca tools suite 18
Endeca Workbench 18
Endeca.Document.IsUnchanged property 197
example syntax of URL filters 182
Exchange server authentication 192
explicit mapping
  creating 31
  described 26
externally created dimensions
  compared to externally managed taxonomies 93
  Developer Studio configuration 94
  importing 97
  introduced 93
  XML requirements 95
externally managed taxonomies
  definition 108
  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 26
first record joins 72
Forge
  encrypting keys with 192
  flags 135
  introduced 14
  running 17
Forge logging system 127
Forge metrics
  enabling 132
  using 133
Forge Metrics Web service 131
  API 133
  enabling SSL 132

H

higher cardinality joins 84

I

implicit mapping
  described 27
  enabling 40
  overriding with null mappings 32
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
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 86
  multiple values for join key 83
  outer 69
  overview 65
  performing in a database 66
  record index keys 67
  sort switch 71
  switch 70

K

key ring file 187
  AUTHENTICATE_HOST attribute 190
  CA_DB element 190
  CERT element 191
  EXCHANGE_SERVER element 192
  HOST attribute 188
  HTTP element 189
  HTTPS element 190
  KEY element 189
  KEY_RING element 188
  PATH attribute 191
  PORT attribute 188
  PRIV_KEY_PATH attribute 192
  PROXY element 192

key ring file *(continued)*
  REALM element 189
  SITE element 188

L

left joins
  described 68
  multiple keys for records 86
loading source data 22
logging
  aliasing v-levels 129
  command line interface 128
  EDF_LOG_LEVEL settings 130
  levels 127
  logLevel 128
  output file 129
  topics 128

M

mapping
  explicit 27
  source properties to dimensions 35
  source properties to like-named dimensions 40
match modes
  Auto Generate 36
  Must Match 36
  Normal 35
Multi Sub-records option for record assembler 86
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 27
  overriding implicit and default mappings 32

O

outer joins 69

P

PARSE_DOC expression 176
passing phrases with Forge 192
Perl assembler 59
Perl manipulator in Endeca Crawler 178
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

pipeline *(continued)*
  fundamentals 43
  placement of property mapper 28
  running 53
  sequential record processing 22
  URLs in 45
  using only one property mapper in 25
precedence rules
  introduced 18
  specifying in Developer Studio 18, 60
priority order of source property mapping 28
properties in Endeca Crawler 164
  name syntax 163
  viewing generated 170
property mapper
  creating 30
  described 51
  minimum configuration 26
  placement in pipeline 28
  using only one per pipeline 25, 51
  using the Mappings editor 30
proxy server authentication 192

R

record adapter
  creating for differential crawling 200
  creating for Endeca Crawler 172
  overview 48
  record index 49
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 86
  use in differential crawling 203
record cache
  adding for joins 79
  Combine Records setting 88
  configuring for differential crawling 202
  creating record indexes 75
  described 59
record index keys for joins 49, 67
record manipulator
  creating for differential crawling 204, 206
  creating for Endeca Crawler 173
record search configured in Developer Studio 18
record specifier property, creating 58
redundant URLs in Endeca Crawler 162
reference implementation, UI 53
REMOVE_RECORD expression 186
removing document body properties in Endeca Crawler 177
RETRIEVE_URL expression for Endeca Crawler 174
root URL extraction settings for Endeca Crawler 181

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 32
  mapping 55
  removing unknown 26
  specifying null mappings for 58
  standardizing 23
source property mapping
  described 25
  priority order 28
  types 27
  viewing existing 30
source property values
  defining maximum length for importing 41
  mapping to dimension values 35
source records 22
specifying root URLs for a spider 180
spider document processing in Endeca Crawler 179
spiders 59
  enabling for differential crawling 195, 201
  specifying proxy servers 184
  specifying record sources 183
  specifying timeouts 183
standardizing source properties 23
stemming 60
stop words 60
Stratify document classification
  adding a property mapper and indexer adapter 118
  building a taxonomy 112
  classification process 117
  configuring a classification server 115
  creating a pipeline 114
  dimension value synonyms 121
  Endeca integration with 109
  exporting a taxonomy 113
  id synonyms 122
  integrating the taxonomy 119
  introduced 107
  loading the dimensions 120
  mapping dimensions 123
  overview 110
  required tools 111
  terms and concepts 108
  updating taxonomies 124
structured data, about 108
switch joins 70
system operations 19
system provisioning 18

T

tagging Endeca records 17
taxonomy
  definition 108
  developing a Stratify 112
thesaurus entries
  configuring in Endeca Workbench 18
  introduced 60

U

UI reference implementation, using 53
unknown source properties, removing 26
unstructured data, about 108
URL and record processing in the Endeca Crawler 162

W

Web service, Forge Metrics 131

X

XML syntax for dimension hierarchy 96