# Oracle® Endeca Server

Data Loading Guide

Version 2.3.0 • July 2012 • Revision A

ORACLE

# Copyright and disclaimer

# Table of Contents

# Preface

Oracle® Endeca Server is the core search-analytical database. It organizes complex and varied data from disparate source systems into a faceted data model that is extremely flexible and reduces the need for up-front data modeling. This highly-scalable server enables users to explore data in an unconstrained and impromptu manner and to rapidly address new questions that inevitably follow every new insight.

# About this guide

This guide describes the two interfaces that allow you to load data into an Endeca data store.

The two interfaces are:

- Data Ingest Web Service, which enables loading, updating, and deleting records in the data store, as well as resetting the data store.
- Bulk Load Interface, which enables loading and replacing records in the data store.

Both interfaces are used by the Oracle Endeca Information Discovery Integrator.

The guide assumes that you are familiar with Oracle Endeca Server concepts and the front-end application development for the Oracle Endeca Server, as well as the specifics of your ETL tool.

# Who should use this guide

This guide is intended for developers who are responsible for using ETL utilities to load source data into the Oracle Endeca Server.

# 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 Customer Support

Oracle Endeca Customer 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 Endeca Customer Support through Oracle's Support portal, My Oracle Support at *https://support.oracle.com*.

Chapter 1

# Introduction

This section provides an introductory overview to the Data Ingest Web Service.



## Overview of the Data Ingest Web Service

The Data Ingest Web Service loads data into a running data store on the Oracle Endeca Server and can also update existing records.

The Data Ingest Web Service therefore allows you to use a data integration platform, such as Integrator, to load data into an application.

You can access the Data Ingest Web Service WSDL at the following URL:

```
http://localhost:<port>/ws/diws/<DataStore>?wsdl
```

where the `localhost` and `port` are the host and port of the running Oracle Endeca Server, and `DataStore` is the name of the Endeca data store.

The Data Ingest Web Service enables performing these tasks:

- Provision the primordial schema records for an initial data store configuration.
- Reset the data store by removing its records and schema.
- Add new records to a running data store. The service accepts batches of records to add. The number of records in each batch is set by the client program. The records can be added to an empty data store (this operation is called an initial load) or to one that already has records.
- Add managed attribute values to a running data store. If the managed values belong to a managed attribute that is not currently in the data store, the service will also create the managed attribute.
- Modify existing records in a running data store. You can add or remove standard attribute values and managed values from already added records.
- Delete records or record data from a running data store.

The Data Ingest Web Service is able to modify a record multiple times in a single transaction (any combination of create, add assignments, delete assignments, and delete record).

The service returns a response indicating the number of records, standard attributes, or managed attribute values that were added or removed as a result of the request. In addition, error messages are returned via a fault mechanism.

The data is sent by an ETL client (such as Integrator) via a program that is running on the client. Typically, ETL client programs written by users use stubs generated from the Data Ingest WSDL and calls from the ETL tool's SDK.

## Interaction with transactions

Any request to the Data Ingest Web Service can contain an optional element `OuterTransactionId` that specifies the ID of an outer transaction (if it has been started by the Transaction Web Service).

This element must be specified as the first element in the request only if a request made by the Data Ingest service is started after a request to start an outer transaction has been made by the Transaction Web Service.

If no outer transactions have been started, the `OuterTransactionId` should not be specified in the request, or the value of this element should be empty. (If the attribute's value is empty, the request ignores the element and interprets it as not specified.)

## About Integrator

Integrator is a high-performance data integration platform that lets you extract source records from a variety of source types (from flat files to databases) and send those records to either the Data Ingest Web Service or the Bulk Load Interface, both of which in turn load the records into the data store.

The records are loaded into the data store via one of the four custom connectors that communicate with the Data Ingest Web Service or a connector that uses the Bulk Load Interface.

For details on Integrator, see the *Oracle Endeca Information Discovery Integrator Guide*, *Oracle Endeca Information Discovery Integrator Server Guide*, and *Oracle Endeca Information Discovery Integrator Components Guide*.

## Data Ingest API

The Data Ingest API is a framework that provides ETL developers with a flexible mechanism to load records from an ETL data source to a running data store. Because it is defined by WSDL documents, the Data Ingest API is language-agnostic. That is, it can be used with any programming language that has Web services support. Thus, the API lets developers choose their favorite development environment (Java, Visual Studio .NET, etc.) on which to write their components.

The *Oracle Endeca Server API Reference* is the documentation generated from the WSDL and XSD files that describe a Web service. This reference provides API-level information about Web services that are packaged with the Oracle Endeca Server. The *Oracle Endeca API Reference* is located in the `doc` directory of the Oracle Endeca Server installation.

## About the Bulk Load Interface

Besides the Data Ingest API, the Bulk Load Interface is available to ingest records into an Endeca data store. The Bulk Load API exists in the form of a collection of Java classes in a single `endeca_bulk_load.jar` file, which is shipped in the Endeca Server's `apis` directory. For information on the Bulk Load API, see *Bulk Load API on page 36*.

# List of operations

This topic lists the operations available in the Data Ingest Web Service.

The operations are the following:

| Operation | Description |
|---|---|
| `ingestRecords` | Adds, modifies, and deletes records (including removing standard attributes and managed attribute value assignments). |
| `ingestDimensionValues` | Adds and updates managed values. |
| `clearMdex` | Deletes the data records and schema records from the data store, indicating the number of records deleted. This operation must be followed by the `provisionMdex` operation.<br>Unlike `deleteRecords`, `clearMdex` deletes all records and does not require specifying record IDs. |
| `provisionMdex` | Provisions the primordial schema records in the data store. This operation is typically run after the `clearMdex` operation. It adds the primordial records (such as PDRs and DDRs), and resets these records to their default values. |

# Important usage note

This topic discusses operations in the data store you can perform using the Data Ingest Web Service. It also lists several data store operations that are not recommended.

## Supported data store operations

You can use the Data Ingest Web Service to perform the following actions on the records in the data store:

- Add new individual records to the data store
- Add managed attribute values
- Add key-value assignments on the existing records in the data store
- Delete specified record assignments
- Delete individual records
- Remove all data records and the schema records from the data store
- Provision the data store (by creating the schema records, such as PDRs and DDRs, and resetting these records to their default values).

## Operations that are not recommended

While the actions listed below can be done with the Data Ingest Web Service, it is not recommended to use the Data Ingest Web Service for them.

The Data Ingest Web Service does not perform reference checks, so it is possible to render your data store invalid or unusable by performing certain operations using this Web service. Here is a list of operations to use with caution:

- Removal of system records — these are records that define your schema (such as PDR, DDR, or other system records). For example, removing a PDR may invalidate a group to which the attribute defined by this PDR belongs. As a workaround, you can remove all records and schema and then provision the data store using the **Reset Data Store** connector in Integrator. For more information on custom connectors available in Integrator, see the *Oracle Endeca Information Discovery Integrator Components Guide*.
- Creation of records that define groups or entities (used for views in Studio). It is recommended to use either the Configuration Web Service or Studio for these actions. For information on the Configuration Web Service, see the *Oracle Endeca Server Developer's Guide*. For information on Studio, see the *Oracle Endeca Information Discovery Studio User's Guide*.

# Data Ingest logging

The Data Ingest Web Service writes its output to the Dgraph logs.

By default, each SOAP request for the Data Ingest Web Service is written to the Dgraph request log.

The Ingest SOAP response provides fault and summary information. If Dgraph verbose logging is turned on (via the Dgraph `-v` flag), this information, as well as the entire SOAP request, is written to the Dgraph standard-out log. The Dgraph `stdout`/`stderr` log is created with the Dgraph `--out` flag.

# Generating client stubs

To create a client application that consumes the Data Ingest Web Service, you need the Web service's WSDL file to generate client stubs.

A WSDL file specifies value types, exceptions, and available methods in a Web service in a programmatic fashion. Typically, a client developer uses a tool that parses the WSDL file and generates client-side stubs (also called proxy classes) and value types. These generated files include all the code necessary to serialize and deserialize SOAP messages and make the SOAP layer transparent to the client developer. The Data Ingest WSDL files can be used with any language that has Web services support.

Tools that generate client stub code from the WSDLs that have been tested are the following:

- Apache CXF 2.2 or later.
- Apache Axis2 1.5.1 or later.
- Web Services Description Language Tool (wsdl.exe), available as part of the Microsoft .NET Framework SDK.

For details on using a WSDL code-generation utility, refer to the utility's documentation.

Keep in mind that the exact syntax of a class member depends on the output of the WSDL tool that you are using. Therefore, check the client stub classes that are generated by your WSDL tool for the exact syntax of the class members.

## Obtaining the WSDL from the deployed service

The Data Ingest Web Service has a unique URL associated with it. If you append **?wsdl** to the service endpoint URL, and specify the name of the data store, the service will automatically generate a service description for the deployed service and return it as XML in your browser, as in this example URL:

```
http://localhost:5555/ws/ingest/datastore?wsdl
```

You can also use this URL in your WSDL tool to generate the stubs, as in this Apache Axis2 example:

```
wsdl2java -uri http://localhost:5555/ws/ingest/datastore
?wsdl -d xmlbeans -s -p com.endeca.dataingest.axis2.addrecords
```

You can insert the `wsdl2java` command in a batch or shell script, or in a build file.

Chapter 2

# Prerequisite Information

This section provides overview information you need to know before using the Data Ingest Web Service.



## Version and namespaces for the Data Ingest Web Service

This topic describes the version of the Data Ingest Web Service and two namespaces used for data ingest operations.

XML namespaces provide a method for qualifying element and attribute names used in Extensible Markup Language documents by associating them with namespaces identified by URI references. The Data Ingest Web Service uses two namespaces.

### Data Ingest Web Service version and namespace

The namespace for the Data Ingest Web Service (DIWS) is:

```
http://www.endeca.com/MDEX/ingest/1/0
```

This namespace reflects the version of the Data Ingest Web Service. This namespace is included in the WSDL document for the web service.

In this example, the string `1/0` indicates the version as 1.0, where 1 is the major verison, and 0 is the minor version. Note that the version in the service that you have installed may not match this example.

Changes to minor versions are backward-compatible. If any backward-compatible versions exist, additional namespaces are included in the WSDL, listing them. You can use any backward-compatible minor version that is listed. For example, if both 1.0 and 1.1 versions are listed in the WSDL, you can use either of them.

Changes to major versions are not backward-compatible, thus previous major versions are not listed in the WSDL namespaces.

**Important:** After you upgrade the Oracle Endeca Server, verify the versions of the web services you have been using against the installed versions, to avoid version mismatch. It is recommended to use the web service versions that match the ones installed with the Oracle Endeca Server.

In particular:

- If the minor version of the web service on your client does not match the version installed with the Oracle Endeca Server, you can still use this version if it is listed in the WSDL namespaces, although it is recommended to upgrade.
- If the major version of the web service on your client does not match the version of the web service installed with the Oracle Endeca Server, you must upgrade to the most recent major version of the web service (this may include upgrading client code to use client stubs generated from the most recent versions).

For more information on web service versions, see the *Oracle Endeca Server Developer's Guide*.

The `xmlns` attribute specifies this namespace for a DIWS prefix for a document, as in this example, also showing the version:

```
<ingest:ingestRecords
   xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
   ...
   <ingest:addAssignments>
      ...
   </ingest:addAssignments>
</ingest:ingestRecords>
```

After this declaration, all DIWS elements will use the same prefix, which will be associated with the same namespace. In the example, the prefix **ingest** is defined for all DIWS elements, such as the `ingest:addAssignments` element.

You can use a prefix of your own choosing, but it must be bound to the DIWS namespace listed above. In this guide, the prefix **ingest** will be used in the examples.

## mdex element namespace

The namespace for `mdex` elements is:

```
http://www.endeca.com/MDEX/XQuery/2009/09
```

The important `mdex` elements used in data ingesting are `mdex:record` for records and the nine property types, such as the `mdex:string` property type.

You must also use the `xmlns` attribute to set the mdex namespace in your XML documents:

```
<ingest:ingestRecords
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <ingest:addAssignments>
      <mdex:record>
         <FactSalesID>683</FactSalesID>
         <Price type="mdex:double">59.99</Price>
         <NumInStock type="mdex:int">68</NumInStock>
         <Description type="mdex:string">Road gloves</Description>
      </mdex:record>
   </ingest:addAssignments>
</ingest:ingestRecords>
```

# Property types for Dgraph records

This topic describes the format of the property types supported by the Dgraph process of the Oracle Endeca Server and the Data Ingest Web Service.

The following table lists the property types that are used by the Dgraph process of the Oracle Endeca Server to create standard attributes:

| Dgraph property name | Property type |
|---|---|
| `mdex:string` | Represents XML-valid character strings. |
| `mdex:int` | Represents a 32-bit signed integer. |
| `mdex:long` | Represents a 64-bit signed integer. |
| `mdex:double` | Represents a floating point. |
| `mdex:boolean` | Represents a Boolean. |
| `mdex:time` | Represents the time of day to a resolution of milliseconds. |
| `mdex:dateTime` | Represents the date and time to a resolution of milliseconds. |
| `mdex:duration` | Represents a length of time with a resolution of milliseconds. |
| `mdex:geocode` | Represents latitude and longitude pairs. |

The type for properties is specified in the `type` attribute. The default type of created standard attributes is `mdex:string` if not otherwise specified. Assignments for an existing standard attribute that specify a type different from that of the associated standard attribute will succeed or fail as per the underlying `put-record` functionality.

## Errors from incorrect property values

You must ensure that you specify the appropriate value type for each property type. For example, attempting to assign a double value (such as 19.99) to an `mdex:int` property will return an `ingestFault` indication a parsing error:

```
<detail>
  <ingest:ingestFault xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0">
     <ingest:errorDetail>Error applying updates: Unable to parse
       property value "19.99" for property "NumInStock" with
       type "mdex:int" on record FactSalesD:569
     </ingest:errorDetail>
  </ingest:ingestFault>
</detail>
```

The `"Unable to parse property value"` error should be returned for any mismatched property value, including using an incorrect case for Boolean values (for example, specifying `"FALSE"` instead of `"false"`).

## string property

`mdex:string` properties represent character strings.

An `mdex:string` property represents variable-length character strings. The characters should conform to the specification for valid XML characters, as described in the W3C XML document at this URL: *http://www.w3.org/TR/REC-xml/#charsets*

Keep in mind that `mdex:string` is the default property data type. That is, if you do not explicitly specify the property type when creating a standard attribute, then `mdex:string` will be used as the property type of the Dgraph record for that attribute.

### Example of ingesting string properties

This example shows how to use string property types for record assignments:

```
<ingest:ingestRecords
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <ingest:addAssignments>
      <mdex:record>
         <FactSalesID>568</FactSalesID>
         <Description type="mdex:string">winter tights</Description>
      </mdex:record>
   </ingest:addAssignments>
</ingest:ingestRecords>
```

## numeric properties

The Dgraph records can have three numeric properties.

The three numeric properties are:

- `mdex:int`
- `mdex:long`
- `mdex:double`

### int properties

An `mdex:int` property represents a 32-bit signed integer. It has a minimum value of -2147483648 and a maximum value of 2147483647 (inclusive).

### long properties

An `mdex:long` property represents a 64-bit signed integer. It has a minimum value of -9223372036854775808 and a maximum value of 9223372036854775807 (inclusive).

### double properties

An `mdex:double` property represents a floating point value. Values can be specified in a decimal-point format (such as 20.0) or in a scientific notation format using "e" or "E" (such as 2.0E1).

### Example of ingesting numeric properties

This example shows how to use the numeric property types for record assignments:

```
<ingest:ingestRecords
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <ingest:addAssignments>
      <mdex:record>
         <FactSalesID>504</FactSalesID>
         <Price type="mdex:double">19.99</Price>
         <NumInStock type="mdex:int">45</NumInStock>
         <TotalSold type="mdex:long">92233720</TotalSold>
      </mdex:record>
   </ingest:addAssignments>
</ingest:ingestRecords>
```

## geocode property

`mdex:geocode` properties represent latitude and longitude pairs.

`mdex:geocode` properties use the format:

```
latvalue lonvalue
```

where each is a double-precision floating-point value:

- *latvalue* is the latitude of the location in whole and fractional degrees. Positive values indicate north latitude and negative values indicate south latitude.
- *lonvalue* is the longitude of the location in whole and fractional degrees. Positive values indicate east longitude, and negative values indicate west longitude.

The latitude and longitude numbers may be separated by arbitrary white space or tab characters. Values are always re-serialized with a single space character regardless of the form of the parsed string.

For example, the following request updates Record 778 with a Location geocode property:

```
<ingest:ingestRecords
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <ingest:addAssignments>
      <mdex:record>
         <FactSalesID>778</FactSalesID>
         <Location type="mdex:geocode">42.365615 -71.075647</Location>
      </mdex:record>
   </ingest:addAssignments>
</ingest:ingestRecords>
```

The value of the geocode property specifies a location at 42.365615 north latitude, 71.075647 west longitude.

## boolean property

`mdex:boolean` property values are useful for tracking true/false conditions.

The valid Boolean values for the `mdex:boolean` property type are:

- `true` or `1` (i.e., `1` is a synonym for `true`)
- `false` or `0` (i.e., `0` is a synonym for `false`)

Note that `true` and `false` are case sensitive and must be specified in lower case.

For example, the following request updates Record 492 with two Boolean properties:

```
<ingest:ingestRecords
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <ingest:addAssignments>
      <mdex:record>
         <FactSalesID>492</FactSalesID>
         <isInStock type="mdex:boolean">true</isInStock>
         <isActive type="mdex:boolean">1</isActive>
      </mdex:record>
   </ingest:addAssignments>
</ingest:ingestRecords>
```

In the example, both properties (isInStock and isActive) are set to true.

## dateTime property

`mdex:dateTime` properties represents a single point in time.

An `mdex:dateTime` property represents the year, month, day, hour, minute, and seconds of a time point, with the optional specification of fractional seconds. You can specify a datetime value as either a universal (UTC) date time or as a local time plus a UTC timezone offset. Note that specifying just a local time is not supported.

### format for universal datetime

The `mdex:dateTime` format for a UTC date time is:

```
yyyy '-' mm '-' dd 'T' hh ':' mm ':' ss {'.' s+} Z
```

where:

- *yyyy* represents a four-digit year. The year value may not be negative, which means that specifying a year prior to 1 BCE is not supported. Year 0000 is not a valid year.
- The first *mm* is a two-digit numeral that represents the month. Numerals representing the first nine months must have a leading zero, such as 07 for July.
- *dd* is a two-digit numeral that represents the day of the month, such as 03 for the third day of the month or 30 for the thirtieth day.
- T is a literal separator indicating that time-of-day follows.
- *hh* is a two-digit numeral that represents the hour. Note that specifying 24 is not permitted (to represent 24, use all zeros for the time portion).
- The second *mm* is a two-digit numeral that represents the minute.
- *ss* is a two-digit numeral that represents the whole seconds.
- *'.' s+* is optional and, if present, represents the fractional seconds. The internal representation is only precise to the millisecond, which means that a specification of four or more digits is truncated to three digits.
- Z (added to the time without a space) is a literal indicator that this date time is Coordinated Universal Time (UTC, sometimes called Greenwich Mean Time). Z is the zone designator for the zero UTC offset.

Note that a hyphen ('-') is the separator between parts of the date portion, a colon (':') is the separator between parts of the time-of-day portion, and a period ('.') is the separator for fractional seconds.

For example, to indicate noon on November 18, 2010 in New York City, you would specify:

```
2010-11-18T17:00:00Z
```

### format for local time plus UTC offset

Alternatively, you can specify the value for an `mdex:dateTime` property as a local time plus a UTC offset. The format for this representation is:

```
yyyy '-' mm '-' dd 'T' hh ':' mm ':' ss {'.' s+} zzzzzz
```

The meanings of the date and time portions are the same as the universal datetime format. *zzzzzz* represents the timezone. Timezones are durations of hours and minutes. Timezones may be specified as positive or negative durations.

The format for a timezone is:

```
('+' | '-') hh ':' mm
```

where:

- *hh* is a two-digit numeral (with leading zeros as required) that represents the hours. The value for *hh* cannot be greater than 14.
- *mm* is a two-digit numeral that represents the minutes. The value for *mm* cannot be greater than 59. However, if *hh* is 14, then *mm* must be 00.
- '+' indicates a non-negative duration.
- '-' indicates a non-positive duration.

For example, to indicate noon on November 18, 2010 in New York City, you would specify:

```
2010-11-18T12:00:00+05:00
```

Note that this time represented in this example is the same as the "2010-11-18T17:00:00Z" time in the universal datetime format.

### Example of ingesting dateTime properties

The following request updates Record 506 with two dateTime properties:

```
<ingest:ingestRecords
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <ingest:addAssignments>
      <mdex:record>
         <FactSalesID>506</FactSalesID>
         <dT1 type="mdex:dateTime">2010-11-18T17:00:00Z</dT1>
         <dT2 type="mdex:dateTime">2010-11-18T12:00:00+05:00</dT2>
      </mdex:record>
   </ingest:addAssignments>
</ingest:ingestRecords>
```

The dT1 property uses the universal datetime format while the dT2 property specifies the datetime as a local time plus a UTC offset.

## time property

`mdex:time` properties represent an instant of time that recurs every day.

An `mdex:time` property represents the hour and minutes of an instance of time, with the optional specification of fractional seconds. A timezone is not allowed as part of the time representation.

The `mdex:time` format is:

```
hh ':' mm ':' ss {'.' s+}
```

where:

- *hh* is a two-digit numeral that represents the hour. Use a leading zero for a single-digit hour, such as 04.
- The second *mm* is a two-digit numeral that represents the minute.
- *ss* is a two-digit numeral that represents the whole seconds.
- *'.' s+* is optional and, if present, represents the fractional seconds. The internal representation is only precise to the millisecond, which means that a specification of four or more digits is truncated to three digits.

A colon (':') is the separator between hours, minutes, and whole seconds, while a period ('.') is the separator for fractional seconds.

Be sure to use a leading zero for single-digit hours, minutes, and whole seconds.

### Example of ingesting time properties

The following request updates Record 624 with two time properties:

```
<ingest:ingestRecords
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <ingest:addAssignments>
      <mdex:record>
         <FactSalesID>624</FactSalesID>
         <time1 type="mdex:time">13:25:43.261</time1>
         <time2 type="mdex:time">09:14:52</time2>
      </mdex:record>
   </ingest:addAssignments>
</ingest:ingestRecords>
```

Note that the time2 property uses a leading zero (i.e., "09") to specify the hour. Omitting the leading zero will cause the operation to fail, with a fault similar to this example:

```
<faultstring>Error applying updates: Unable to parse property
value "9:14:52" for property "time2" with type "mdex:time" on
record FactSalesID:624</faultstring>
```

## duration property

`mdex:duration` properties represent a duration of time.

An `mdex:duration` property represents a duration of the days, hours, and minutes of an instance of time. A timezone is not allowed as part of the time representation.

The `mdex:duration` format is:

```
'P' {d 'D'} 'T' {h 'H'} {m 'M'} {s {'.' s+} 'S'}
```

where:

- `P` is a mandatory literal that indicates that this is a period of time.
- For the `d'D'` parameter, *d* specifies the number of days while the literal `D` indicates that this is the days field.
- `T` is a literal date/time separator that must be present if (and only if) any time fields are specified.
- For the `h'H'` parameter, *h* specifies the number of hours while the literal `H` indicates that this is the hours field.
- For the `m'M'` parameter, *m* specifies the number of minutes while the literal `M` indicates that this is the minutes field.
- For the `s'S'` parameter, *s* specifies the number of whole seconds while the literal `S` indicates that this is the seconds field. *'.' s+* is optional and, if present, represents the fractional seconds (the internal representation is only precise to the millisecond, which means that a specification of four or more digits is truncated to three digits).

Note that all time durations are optional, but at least one must be present. An optional preceding minus sign ('-') is allowed to indicate a negative duration.

## duration format examples

This example specifies a duration of 429 days, 1 hour, 2 minutes, and 3 seconds:

```
P429DT1H2M3S
```

This example specifies a duration of 429 days:

```
P429D
```

This example specifies a duration of 429 days, 2 minutes, and 3.25 seconds:

```
P429DT2M3.25S
```

This example specifies a 1 hour and 2 minutes:

```
PT1H2M
```

This example specifies a negative duration of 429 days and 3 seconds:

```
-P429DT3S
```

## Example of ingesting duration properties

The following request updates Record 344 with five duration properties:

```
<ingest:ingestRecords
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <ingest:addAssignments>
      <mdex:record>
         <FactSalesID>344</FactSalesID>
         <duration1 type="mdex:duration">P429DT1H2M3S</duration1>
         <duration2 type="mdex:duration">P429D</duration2>
         <duration3 type="mdex:duration">P429DT2M3.25S</duration3>
         <duration4 type="mdex:duration">PT1H2M</duration4>
         <duration5 type="mdex:duration">-P429DT3S</duration5>
      </mdex:record>
   </ingest:addAssignments>
</ingest:ingestRecords>
```

Note that the duration5 property has a negative duration value.

# Default values for new attributes in the data store

New standard attributes and managed attributes created during an ingest are given a set of default values.

During any data ingest operation, if a non-existent standard attribute is specified for a record, the specified standard attribute is automatically created by the Data Ingest Web Service. Likewise, non-existent managed attributes specified for a record are also automatically created. Note that you cannot disable this automatic creation of properties.

## Default values for standard attributes

The PDR for a standard attribute that is automatically created will use the system default settings, which (unless they have been changed by the data developer) are:

| PDR property | Default setting |
|---|---|
| `mdex-property_Key` | Set to the standard attribute name specified in the request. |
| `mdex-property_Type` | Set to the Dgraph property type specified in the request. If no property type was specified, defaults to an `mdex:string` type. |
| `mdex-property_IsPropertyValueSearchable` | `true` (the standard attribute will be enabled for value search) |
| `mdex-property_IsSingleAssign` | `false` (a record may have multiple value assignments for the standard attribute) |
| `mdex-property_IsTextSearchable` | `false` (the standard attribute will be disabled for record search) |
| `mdex-property_IsUnique` | `false` (more than one record may have the same value of this standard attribute) |
| `mdex-property_TextSearchAllowsWildcards` | `false` (wildcard search is disabled for this standard attribute) |
| `system-navigation_Select` | `single` (allows selecting only one refinement from this standard attribute) |
| `system-navigation_ShowRecordCounts` | `true` (record counts will be shown for a refinement) |
| `system-navigation_Sorting` | `record-count` (refinements are sorted in descending order, by the number of records available for each refinement) |

### Default values for managed attributes

A managed attribute that is automatically created will have both a PDR and a DDR created by the Data Ingest Web Service. The default values for the PDR are the same as listed in the table above, except that `mdex-property_IsPropertyValueSearchable` will be `false` (i.e., the managed attribute will be disabled for value search).

The DDR will use the system default settings, which (unless they have been changed by the data developer) are:

| DDR property | Default setting |
|---|---|
| `mdex-dimension_Key` | Set to the managed attribute name specified in the request. |
| `mdex-dimension_EnableRefinements` | `true` (refinements will be displayed) |
| `mdex-dimension_IsDimensionSearchHierarchical` | `false` (hierarchical search is disabled during value searches) |
| `mdex-dimension_IsRecordSearchHierarchical` | `false` (hierarchical search is disabled during record searches) |

# NCName format for attributes in the data store

The names of standard attributes and managed attributes in the data store must be in an NCName format.

The NCName format is defined in the W3C document Namespaces in XML 1.0 (Second Edition), located at this URL: *http://www.w3.org/TR/REC-xml-names/#NT-NCName*

As defined in the W3C document, an NCName must start with either a letter or an underscore (but keep in mind that the W3C definition of Letter includes many non-Latin characters). If the name has more than one character, it must be followed by any combination of letters, digits, periods, dashes, underscores, combining characters, and extenders. (See the W3C document for definitions of combining characters and extenders.) The NCName cannot have colons or white space.

After creating the Endeca attribute, you can use the `mdex-property_DisplayName` property on the PDR to specify a display name. The display name, which can use a non-NCName format, is intended to serve as an easy-to-understand name for the data store's attribute when it is displayed in the application's front end (such as in Studio's **Results Table** component).

# Interaction with the Transaction Web Service

All requests made with the Data Ingest Web Service can optionally specify the outer transaction ID.

If you submit any request to the Data Ingest Web Service after a Transaction Web Service request that starts an outer transaction, the request must specify the outer transaction ID. If no transactions have been started, the ID attribute must be omitted in the request.

The outer transaction ID is issued by the Transaction Web Service, once a request is sent to it to start an outer transaction. From that point on, all requests issued to the Oracle Endeca Server's data store must reference this ID, until the outer transaction is committed.

(If an outer transaction is in progress and you don't know the ID, you can obtain it by using the `listOuterTransaction` operation on the Transaction Web Service.)

The format of the request that has an outer transaction ID specified may be similar to the following (top-level namespaces are omitted in this example):

```
<ingest:clearMdex >
<OuterTransactionId>MyID</OuterTransactionID>
</ingest>
```

**Note:** The `<OuterTransactionId>` element must be the first element in the request.

# Troubleshooting connection timeouts

You can use the Dgraph `--net-timeout` flag to help prevent timeout issues during data ingest operations.

The Dgraph process of the Oracle Endeca Server has a default request timeout of 30 seconds. This setting determines the maximum number of seconds that the Dgraph process waits for the client to download data from queries across the network. The client can be an end user sending a query to the Oracle Endeca Server, or for data ingest operations, an ETL client program that is loading records into the data store.

If the client opens a connection with the server, the server will wait (for the length of the timeout period) for the receipt of client data on that socket. If the client does not send data within the timeout limit, then the server will drop the connection and log an HTTP 408 error in the Dgraph log.

For ingest operations, this timeout limit may pose problems if you have a DIWS client that takes longer to send data. If the timeout limit is exceeded, the ingest request fails (because the server closes the connection) and the record batch is not loaded into the data store residing on the server.

If you continually see HTTP 408 errors in the logs, first verify that your ETL client is working properly. For example, make sure that the program is not spending an unusual amount of time in an operation that would cause it to exceed the timeout limit.

If you believe that the ETL client is executing as expected but needs a longer request timeout period, then you can try increasing the request timeout setting. Use the Dgraph `--net-timeout` flag to set the request timeout to a number that works for the ETL client. You will probably have to experiment with several settings to find the one that is optimal for your needs.

Chapter 3

# Adding New Records

This section describes how to initially load records into the data store, as well as how to ingest additional new records and managed values.



## About primary-key attributes

A primary key is required in order to add, delete, or modify a record in the data store.

Each record in the data store is uniquely identified by a unique record identifier, which is a combination of a unique standard attribute and a value that appears only on that record (that is, no other record in the data set has the same key-value pair that is on this record). This unique standard attribute is called a primary-key attribute. The primary-key attribute and a value assigned to a record becomes the primary key of the record. Every record added to the data store must have a primary key.

The primary-key attribute type can be of any supported Dgraph property types. The name of the primary-key attribute must be in an NCName format.

Typically, you would use the `mdex:string` or `mdex:int` types for the primary-key attribute. When you use the `primaryKey` element to create the primary-key attribute, the resulting property type will be whatever type is specified by the record's primary key (which is in the `addAssignments` element). Note that the primary-key attribute is created only if it is assigned to a record.

The PDR (Property Description Record) for the primary-key attribute must have this property set:

- `mdex-property_IsUnique` must be set to `true`. This means that a value may be assigned to at most one record.

For example, assume that the name of a primary-key attribute is `partID`. The value **partID=P123** can be assigned to only one record in the data set and is the primary key for that record. No other record can have this key-value pair. As a result, this primary key uniquely identifies this record in the data set.

**Note:** Keep in mind that multiple primary key attributes can exist in the data store. Each record must have one (and only one) primary key. The Data Ingest Web Service will throw an error if you attempt to add a second primary-key attribute to a record that already has a primary key.

## Using the primaryKey element

When adding a record, the primary-key attribute (that will be assigned on the record) must already exist in the data store or (if it does not exist) must be specified in the add-records request with the `primaryKey` element. This example shows the `primaryKeys` section of a request:

```
<ingest:ingestRecords
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <ingest:primaryKeys>
      <ingest:primaryKey name="partID"/>
   </ingest:primaryKeys>
   <ingest:addAssignments>
      <mdex:record>
         <partID type="mdex:string">P123</partID>
         <modelNum type="mdex:int">2562</modelNum>
      </mdex:record>
   </ingest:addAssignments>
</ingest:ingestRecords>
```

The example creates one primary-key attribute (**partID**), which is used to create a record whose primary key is **partID=P123**.

## Default values for primary-key attributes

If you specify a non-existent attribute as the primary key, the standard attribute is automatically created by the Data Ingest Web Service. The PDR for the attribute will use the system default settings, which (unless they have been changed by the data developer) are:

| PDR property | Default setting |
|---|---|
| `mdex-property_Key` | Set to the name specified in the request. |
| `mdex-property_Type` | Set to the Dgraph property type specified in the request. If no property type was specified, defaults to an `mdex:string` type. |
| `mdex-property_IsPropertyValueSearchable` | `true` (the attribute will be enabled for value search) |
| `mdex-property_IsSingleAssign` | `true` (a record may have at most one value for the attribute) |
| `mdex-property_IsTextSearchable` | `false` (the attribute will be disabled for record search) |
| `mdex-property_IsUnique` | `true` (a value may be assigned to at most one record) |
| `mdex-property_TextSearchAllowsWildcards` | `false` (wildcard search is disabled for this attribute) |
| `system-navigation_Select` | `single` (allows selecting only one refinement from this attribute) |
| `system-navigation_ShowRecordCounts` | `true` (record counts will be shown for a refinement) |

| PDR property | Default setting |
|---|---|
| `system-navigation_Sorting` | `record-count` (refinements are sorted in descending order, by the number of records available for each refinement) |

# Adding new records

The `addAssignments` element of the `ingestRecords` operation allows you to add new records to the data store.

The records to be added are considered totally additive. That is, if a record with the same primary key already exists in the data store, the key-value pair list of the added record will be merged into the existing record. If attribute values with the same name already exist, then the added key-value pairs will be additional values for the same attribute (multi-assign).

**Note:** The `addAssignments` element is also used to extend (update) existing records. This usage is described in the following chapter.

## ingestRecords request

An add-records request uses the `ingestRecords` operation with the `addAssignments` element. The record to be added must have a primary-key assignment. It can have other key-value pair assignments as needed.

**Note:** If you submit the `ingestRecords` request after a Transaction Web Service request that starts an outer transaction, the request must specify the outer transaction ID. If no outer transactions have been started, the ID attribute must be omitted in the request.

The basic request format is:

```
<ingest:ingestRecords
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <ingest:addAssignments>
      <mdex:record>
         <primaryKeyProp>primaryKeyValue</primaryKeyProp>
         <!-- List of other property assignments -->
      </mdex:record>
   </ingest:addAssignments>
</ingest:ingestRecords>
```

For example, this request adds one record (with the primary key P123) to the data store.

```
<ingest:ingestRecords
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <ingest:addAssignments>
      <mdex:record>
         <partID>P123</partID>
         <color type="mdex:string">red</color>
         <price type="mdex:double">19.99</price>
      </mdex:record>
   </ingest:addAssignments>
</ingest:ingestRecords>
```

The primary key of the record is the **partID** primary-key attribute (which in this case must already exist in the data store of the Oracle Endeca Server). The request also creates the **color** and **price** attributes, which previously did not exist in the data store.

## Success response

An `ingestRecordsResponse` for a successful add-records request looks like this example:

```
<ingest:ingestRecordsResponse
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0">
   <ingest:numPropertiesCreated>2</ingest:numPropertiesCreated>
   <ingest:numRecordsAffected>1</ingest:numRecordsAffected>
   <ingest:numRecordsDeleted>0</ingest:numRecordsDeleted>
</ingest:ingestRecordsResponse>
```

The sample response shows that one record was created and that two attributes (the **color** and **price** attributes) were also created. The **partID** attribute was not created because it already existed in the data store.

## Failure response

On failure, a SOAP fault is returned. The `ingestFault` and `errorDetail` elements should contain the error that caused the failure.

For example, assume that one of the record assignments contained a mismatched attribute element that looked like this:

```
<partNum type="mdex:int">24869</price>
```

The `errorDetail` element would return an error similar to this:

```
'request' cannot be parsed as XML.
Reason: Unable to fetch resource:Expected end of tag 'partNum'
```

In this example, the reason for the error is that the `</partNum>` ending tag was not found (because `</price>` was mistakenly used instead).

## State of the data ingest process on failure

The Data Ingest Web Service uses an all-or-nothing insertion strategy for each batch of records. This means that if at least one record in a batch is considered invalid by the Dgraph process of the Oracle Endeca Server, then all of the records are rejected. For example, if a batch of 1000 records contains 999 valid records and 1 invalid record, then the 999 valid records (and the invalid record) are not loaded into the data store.

If the data ingest process is interrupted (for example, by the ETL client or the Dgraph process crashing), then the current batch (i.e., the batch that was being processed when the interruption occurred) is not loaded into the data store. However, all previous valid batches have been loaded into the data store. For example, if 5000 batches are to be loaded and an interruption occurs during batch 3500, then batch 3500 is not loaded into the data store, but the previous 3499 batches will be present in the data store.

## Standard attribute assignments and creations

When adding standard attributes, the operation works as follows for the new attribute (i.e., the attribute to be added):

- If the new attribute already exists in the data store but with a different type, an error is thrown and the new attribute is not added.

- If the new attribute already exists in the data store and is of the same type, no error is thrown and nothing is done.

Standard attribute names must use an NCName format. The standard attribute name is used as the element name for the assignment, in this format:

```
<propertyName type="propertyType">property value</propertyName>
```

For example, assigning a standard attribute named **ItemID** would look like this:

```
<ItemID type="mdex:int">247</ItemID>
```

Standard attributes are created as needed when non-existent attributes are specified for a record. The PDR for the attribute will use the system default settings, which is explained in the "Default values for new attributes in the data store" topic in this guide. Note that you cannot disable this automatic creation of attributes.

# Initial loading of records

The use case for the initial load of records assumes that you are loading records into an empty data store that has been created in the Oracle Endeca Server.

The initial data load is performed via one or more invocations of the Data Ingest Web Service `ingestRecords` operation specifying one or more `mdex:record` elements.

The following actions take place during the initial load of records:

- All of your source records are loaded into the data store.
- Appropriate primary-key attributes are created by using the `primaryKey` element in the request. (This is because the initially created data store contains no primary-key attributes and they must be created.)

The request for an initial load should use one or more `primaryKey` elements to create the primary-key attributes. An example of a full request is:

```
<ingest:ingestRecords
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <ingest:primaryKeys>
      <ingest:primaryKey name="partID"/>
      <ingest:primaryKey name="supplierID"/>
   </ingest:primaryKeys>
   <ingest:addAssignments>
      <mdex:record>
         <partID type="mdex:string">P123</partID>
         <modelNum type="mdex:int">2562</modelNum>
      </mdex:record>
      <mdex:record>
         <supplierID type="mdex:string">S456</supplierID>
         <location type="mdex:geocode">42.365615 -71.075647</location>
      </mdex:record>
   </ingest:addAssignments>
</ingest:ingestRecords>
```

The request first creates the `partID` and `supplierID` primary-key attributes, and then adds two new records to the data store. The primary key of the first record is `partID=P123` while `supplierID=S456` is the primary key of the second record. The request also creates two standard attributes (`modelNum` and `location`) because they do not exist in the data store.

**Note:** If you submit the `ingestRecords` request after a Transaction Web Service request that starts an outer transaction, the request must specify the outer transaction ID. If no outer transactions have been started, the ID attribute must be omitted in the request.

To load records into an empty data store:

1. Use the `create-ds` command of the Oracle Endeca Server to create an instance of the data store and start the Dgraph process.
2. Create an `ingest:ingestRecords` request, similar to the example above, and send the request to the Data Ingest Web Service.
   The request is typically created and managed by a ETL client.
3. After the request is made, check the `ingestRecordsResponse` to determine if the request transaction was successful.

A successful `ingestRecordsResponse` returned from the above sample request should look like this:

```
<ingest:ingestRecordsResponse xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0">
   <ingest:numPropertiesCreated>4</ingest:numPropertiesCreated>
   <ingest:numRecordsAffected>2</ingest:numRecordsAffected>
   <ingest:numRecordsDeleted>0</ingest:numRecordsDeleted>
</ingest:ingestRecordsResponse>
```

# Adding records after the initial load

You can add more records to the data store at any time after the initial loading of records is complete.

Adding more records after the data store is created and its Dgraph process is up and running with the initially-loaded record set is very similar to the initial-load scenario, which means:

- You use the `ingestRecords` operation with the `addAssignments` element and one or more `mdex:record` elements.
- If you are adding new records with new primary keys, you must use the `primaryKey` element in the request. Otherwise, do not use this element if the new records use an existing primary-key attribute.
- As with an initial load operation, standard attributes are created as needed when non-existent attributes are specified for a new record. The PDR for the standard attribute will use the system default settings.
- If a standard attribute is configured as multi-assign, a record can have multiple assignments of that attribute.
- You can add multiple records with the same request. You can also update other, existing records with the same request.

In addition, the request can contain `deleteRecords` elements to delete records.

## New record request

The format of the `ingestRecords` request to add new records is the same as documented in the "Adding new records" topic in this section.

**Note:** If you submit the `ingestRecords` request after a Transaction Web Service request that starts an outer transaction, the request must specify the outer transaction ID. If no outer transactions have been started, the ID attribute must be omitted in the request.

For example, this request adds two records to the data store:

```
<ingest:ingestRecords
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
```

```
   <ingest:addAssignments>
      <mdex:record>
         <partID>P247</partID>
         <color>blue</color>
         <weight>3</weight>
         <price>19.99</price>
      </mdex:record>
      <mdex:record>
         <supplierID>S394</supplierID>
         <company>Acme Inc.</company>
         <phone>1-555-123-4567</phone>
      </mdex:record>
   </ingest:addAssignments>
</ingest:ingestRecords>
```

Note that none of the key-value assignments specify a Dgraph property type. This is because all the attributes already exist in the data store and therefore do not need to be created. The type of the assignment property value must match the type of the attribute.

# Loading managed attribute values

The `ingestDimensionValues` operation allows you to load managed values into the data store.

Within the `ingestDimensionValues` structure, the `ingestDimensionValue` element specifies the managed attribute to which each managed value belongs. If the managed attribute does not exist in the data store, the service automatically creates the managed attribute. For the default values of the managed attribute's PDR and DDR, see the "Default values for new attributes in the data store" topic.

You can use the `ingestDimensionValues` operation to load an externally managed taxonomy (EMT) into the data store. When loaded, externally managed taxonomies are added as managed attributes and managed values.

## ingestDimensionValues request

An `ingestDimensionValues` operation request uses this format:

```
<ingest:ingestDimensionValues
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <!--Zero or more repetitions:-->
   <ingest:dimensionValue dimension="dimName" displayName="dimValName"
         parentSpec="pSpec" spec="dimValId">
      <!--Zero or more repetitions:-->
      <ingest:synonym>synName</ingest:synonym>
      <!--Optional:-->
      <ingest:properties>
         <!--Zero or more repetitions:-->
         <propName type="mdex:string">propValue</propName>
      </ingest:properties>
   </ingest:dimensionValue>
</ingest:ingestDimensionValues>
```

Each `DimensionValue` element defines one managed value. The meanings of the attributes and sub-elements are:

| Element/Attribute | Purpose |
|---|---|
| `dimension` | The name of the managed attribute to which the managed value belongs. The name must use the NCName format. |
| `displayName` | The name for the managed value. |
| `parentSpec` | Specifies the parent ID (managed attribute spec) for this managed value, If this is a root managed value, use a forward slash (/) as the ID. If this is a child managed value, specify the unique ID of the parent managed value. |
| `spec` | A unique string identifier for the managed value. It is the responsibility of the client to provide the identifier for the request. |
| `synonym` | Optionally defines the name of a synonym. You can add synonyms to a managed value so that users can search for other text strings and still get the same records as a search for the original managed value name. Synonyms can be added to both root and child managed values. |
| `properties` | Optionally defines a property for a managed value. Managed value properties provide descriptive information about a given managed value and are intended to be used for display purposes by the application. |

The following example creates the Component managed attribute and adds three managed values (Derailleur, Tire, and Michelin) to it:

```
<ingest:ingestDimensionValues
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
    <ingest:dimensionValue
          dimension="Component"
          displayName="Derailleur"
          parentSpec="/"
          spec="22">
        <ingest:synonym>Chain</ingest:synonym>
        <ingest:synonym>Gear</ingest:synonym>
    </ingest:dimensionValue>
    <ingest:dimensionValue
          dimension="Component"
          displayName="Tire"
          parentSpec="/"
          spec="47">
        <ingest:properties>
            <myStrProp type="mdex:string">source:CAS</myStrProp>
        </ingest:properties>
    </ingest:dimensionValue>
    <ingest:dimensionValue
          dimension="Component"
          displayName="Michelin"
          parentSpec="47"
          spec="35" />
</ingest:ingestDimensionValues>
```

In the example, the Derailleur and Tire managed values are at the root of the managed attribute, while the Michelin managed value is a child of the Tire managed value. Note also that two synonyms were created for

the Derailleur managed value and a string property (named myStrProp) was created for the Tire managed values.

## ingestDimensionValuesResponse

An `ingestDimensionValuesResponse` for a successful operation would look like this example:

```
<ingest:ingestDimensionValuesResponse
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0">
   <ingest:numDimensionsCreated>1</ingest:numDimensionsCreated>
   <ingest:numDimensionValuesCreated>3</ingest:numDimensionValuesCreated>
</ingest:ingestDimensionValuesResponse>
```

In the sample response, the `numDimensionsCreated` element shows that one managed attribute was created, while the `numDimensionValuesCreated` element shows that three managed values were created.

## Failure response

On failure, a SOAP fault is returned. The `ingest:ingestFault` and `ingest:errorDetail` elements should contain the error that caused the failure.

For example, assume that the following request was made to create the Saddle child managed value:

```
<ingest:ingestDimensionValues>
    <ingest:dimensionValue dimension="Component" displayName="Saddle"
        parentSpec="58" spec="39" />
</ingest:ingestDimensionValues>
```

The `ingest:errorDetail` element would return an error similar to this:

```
<soapenv:Fault>
   <faultcode>soapenv:Client</faultcode>
   <faultstring>Error applying updates: Dimension value put refers to parent spec "58",
       which does not exist in dimension "Component"
   </faultstring>
   <detail>
      <ingest:ingestFault xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0">
         <ingest:errorDetail>Error applying updates: Dimension value put refers to
            parent spec "58", which does not exist in dimension "Component"
         </ingest:errorDetail>
      </ingest:ingestFault>
   </detail>
</soapenv:Fault>
```

In this example, the reason for the error is that the request refers to a non-existent parent managed value (58 in the example).

Chapter 4

# Updating Records

This section describes how you can incrementally modify the data store by updating and deleting records.



## About updates

The `ingestRecords` operation lets you incrementally update the data store running in the Oracle Endeca Server, including adding additional records.

Using the Data Ingest Web Service, you can perform the following types of incremental updates:

- Add a brand-new record to the data set.
- Update an existing record by adding standard attribute and/or managed values.
- Update an existing record by removing standard attributes and/or managed values.

### How updates are applied

The records to be added are considered totally additive. That is, if a record with the same primary key already exists in the data store, the key-value pair list of the added record will be merged with the existing record.

If a standard attribute with the same name already exists (but has a different assigned value), then the added attribute will be an additional value for the same attribute (multi-assign). For example, if the existing record has one standard attribute named `color` with a value of `red` and the request adds a `color` standard attribute with a value of `blue`, then the resulting record will have two `color` attributes.

Keep in mind, however, that you cannot add a second value to a single-assign attribute. (That is, a standard attribute whose PDR has the `mdex-property_IsSingleAssign` set to `true`.) In the `color` example, if `color` were a single-assign attribute and the record already had one `color` assignment, then an attempt to add a second `color` assignment would fail.

When adding standard attributes, the operation works as follows for the new standard attribute (i.e., the standard attribute to be added):

- If the new standard attribute already exists in the data store but with a different type, an error is thrown and the new standard attribute is not added.
- If the new standard attribute already exists in the data store and is of the same type, no error is thrown and nothing is done.

- If the new standard attribute is a primary-key attribute and a managed attribute already exists with the same name, an error is thrown and the new standard attribute is not added.

Note that updating a record can cause it to change place in the default order. That is, if you have records ordered A, B, C, D, and you update record B, records A, C, and D remain ordered. However, record B may move as a result of the update, which means the resulting order might end up as B,A,C,D or A,C,B,D or another order.

### Order of update operations

An `ingestRecords` request can contain all four types of updates. In this case, the order of processing is:

1. `deleteRecords` requests are processed first.
2. `wildcardDeletes` requests are processed second.
3. `deleteAssignments` requests are processed third.
4. `addAssignments` requests are processed last.

If a record is included in the `deleteRecords` element and is also included in one or more of the other elements, then the record is deleted and added again in the same transaction.

If a record attribute or attribute assignment is specified in the `wildcardDeletes` or `deleteAssignments` list and is specified again in the `addAssignments` list, then the attribute assignment is deleted and added again in the same transaction.

If identical records, standard attributes or assignments are specified in any of the elements, the redundant entries are ignored.

### Affected records with update operations

The `numRecordsAffected` element in the `ingestRecordsResponse` lists how many records were affected (i.e., modified) by an `ingestRecords` operation.

However, it is possible that an affected record may not actually be changed by the operation. Any operation that results in the output record being the same as the input record will mark the record as affected but will leave it unchanged. These types of unaffected operations are adding an assignment that already exists, deleting an assignment that does not exist, performing a wildcard delete on a record property that has no assignments, and deleting an assignment and then adding the same assignment.

## Adding key-value assignments

Records in the data store can be updated with new assignments for standard attributes and managed values.

The `ingestRecords` operation, when used with the `addAssignments` element, lets you update existing records in the data store by adding standard attribute values and/or managed values. The element can also create a standard or managed attribute if the attribute to be added does not exist. In this case, it is added with the defaults listed in the "Default values for new attributes in the data store" topic in this guide.

Because the Dgraph process of the Oracle Endeca Server performs type-checking when adding standard attributes, keep the following in mind:

- When adding a value for a pre-existing standard attribute, make sure that the new value is of the proper type. An error will occur for type mismatches (for example, if you attempt to assign the string "red" to an integer standard attribute).

- When creating and adding a new standard attribute, you should specify the Dgraph process property type.
- Any standard attribute that is not specifically typed will be treated by default as a string type.

You can assign multiple values from a given standard attribute only if the attribute is configured as a multi-assign standard attribute. That means that the PDR for the standard attribute has the `mdex-property_IsSingleAssign` property set to `true`. If the `addAssignments` list attempts to assign multiple values to a standard attribute that does not accept multiple values, an error is signaled.

Managed values can be added to records even if the managed attribute to which they belong does not exist in the data store. In this case, the Data Ingest Web Service automatically creates the managed attribute.

## addAssignments request

You use the `addAssignments` element in a request to add key-value pairs to an existing record. The request must specify the primary key of the record to be updated, using this request format:

```
<ingest:ingestRecords
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <ingest:addAssignments>
      <mdex:record>
         <primaryKeyProp>keyValue</primaryKeyProp>
         <propToAdd>keyValue</propToAdd>
         <!-- Other property assignments -->
      </mdex:record>
   </ingest:ingest:addAssignments>
</ingest:ingestRecords>
```

For example, this request updates a record (with the primary key P123) with three standard attributes (color, price, and numInStock) and one managed value (the managed value with the managed value spec of 4 in the Style managed attribute):

```
<ingest:ingestRecords
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <ingest:addAssignments>
      <mdex:record>
         <partID>P123</partID>
         <color>yellow</color>
         <price>18.99</price>
         <numInStock type="mdex:int">10</numInStock>
         <Style>4</Style>
      </mdex:record>
   </ingest:addAssignments>
</ingest:ingestRecords>
```

Note that the Dgraph process property type of the `numInStock` standard attribute is specified, because the standard attribute does not exist and therefore will be created as part of the request. The other standard attributes and the `Style` managed attribute already exist in the data store.

**Note:** If you submit the `ingestRecords` request after a Transaction Web Service request that starts an outer transaction, the request must specify the outer transaction ID. If no outer transactions have been started, the ID attribute must be omitted in the request.

# Removing record assignments

You can update records in a running data store by removing standard attribute and managed value assignments.

The `ingestRecords` operation has two elements that delete assignments from records in the data store:

- `deleteAssignments`
- `wildcardDeletes`

Both elements can delete standard attribute assignments as well as managed value assignments. Note that the standard attributes or managed values are not removed from the data store; they are removed only from the specified records.

You can use both elements in the same `ingestRecords` operation. In this case, the `wildcardDeletes` request is processed before the `deleteAssignments` request.

Both elements are case sensitive, including the standard attribute and managed value names and their assignment values.

**Note:** If you submit the `ingestRecords` request after a Transaction Web Service request that starts an transaction, the request must specify the outer transaction ID. If no outer transactions have been started, the ID attribute must be omitted in the request.

## deleteAssignments request

The `deleteAssignments` element of the `ingestRecords` operation removes individual standard attribute and/or managed value assignments from Endeca records, but does not otherwise affect the record. You can remove one or more assignments in the same request.

The request must specify the primary key of the record to be updated. The `deleteAssignments` request format is:

```
<ingest:ingestRecords
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <ingest:deleteAssignments>
      <mdex:record>
         <primaryKeyProp>keyValue</primaryKeyProp>
         <propName>keyValueToRemove</propName>
         <!-- Other property or dimension value assignments to remove -->
      </mdex:record>
   </ingest:deleteAssignments>
</ingest:ingestRecords>
```

To remove an individual assignment, specify the key name (i.e., standard attribute name or managed value name) and its value, as in this example:

```
<ingest:ingestRecords
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <ingest:deleteAssignments>
      <mdex:record>
         <partID>P123</partID>
         <color>red</color>
         <color>blue</color>
         <Component>crankset</Component>
      </mdex:record>
   </ingest:deleteAssignments>
</ingest:ingestRecords>
```

The example removes two values ("red" and "blue") of the `color` standard attribute assignment and one managed value ("crankset") from the `Component` managed value assignment.

A successful `ingestRecordsResponse` returned from the above sample request should look like this:

```
<ingest:ingestRecordsResponse
     xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0">
   <ingest:numPropertiesCreated>0</ingest:numPropertiesCreated>
   <ingest:numRecordsAffected>1</ingest:numRecordsAffected>
   <ingest:numRecordsDeleted>0</ingest:numRecordsDeleted>
</ingest:ingestRecordsResponse>
```

The `numRecordsAffected` element in the response shows that one record was successfully modified.

## wildcardDeletes request

The `wildcardDeletes` element of the `ingestRecords` operation removes all assignments from the same standard attribute or managed value at once.

The request must specify the primary key of the record to be updated, using this request format:

```
<ingest:ingestRecords
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <ingest:wildcardDeletes>
      <mdex:record>
         <primaryKeyProp>keyValue</primaryKeyProp>
         <propToRemove />
         <!-- Other property or dimension value assignments to remove -->
      </mdex:record>
   </ingest:wildcardDeletes>
</ingest:ingestRecords>
```

Note that unlike the `deleteAssignments` usage, the `wildcardDeletes` element requires that the *propToRemove* specification cannot have an assignment value. Only the name of the standard attribute or managed value can be specified.

To remove all assignments on the record from a specific standard attribute or managed value, use a single tag with the attribute, as in this example that removes all assignments from the `color` standard attribute:

```
<ingest:ingestRecords
      xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0"
      xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <ingest:wildcardDeletes>
      <mdex:record>
         <partID>P123</partID>
         <color/>
         <sizes/>
      </mdex:record>
   </ingest:wildcardDeletes>
</ingest:ingestRecords>
```

In the example, if record P123 had six assignments from the `color` standard attribute, then all six assignments would be removed; if it had three assignments from the `sizes` standard attribute, all three would be removed.

If successful, the operation returns the same `ingestRecordsResponse` as a `deleteAssignments` request.

# Deleting records

The Data Ingest Web Service lets you delete records from a running data store.

You use the `deleteRecords` element in a request to delete a record. The request must specify the primary key of the record to be deleted, using this request format:

```
<ingest:ingestRecords>
   <ingest:deleteRecords>
      <mdex:record>
         <primaryKeyProp>keyValue</primaryKeyProp>
      </mdex:record>
   </ingest:deleteRecords>
</ingest:ingestRecords>
```

Multiple records can be deleted in the same request. Each record must be specified within an `mdex:record` element.

If you submit the `ingestRecords` request after a Transaction Web Service request that starts an outer transaction, the request must specify the outer transaction ID. If no outer transactions have been started, the ID attribute must be omitted in the request.

To delete a record from the data store:

1. Make certain that both the data store's Dgraph process and the Data Ingest service are running.
2. Create a `deleteRecords` request, similar to the example below that deletes two records, and send the request to the Data Ingest service.

   ```
   <ingest:ingestRecords>
      <ingest:deleteRecords>
         <mdex:record>
            <partID>PK-123</partID>
         </mdex:record>
         <mdex:record>
            <supplierID>SV-789</supplierID>
         </mdex:record>
      </ingest:deleteRecords>
   </ingest:ingestRecords>
   ```

3. After the request is made, check the `ingestRecordsResponse` to determine if the request transaction was successful.

A successful `ingestRecordsResponse` returned from the above sample request should look like this:

```
<ingest:ingestRecordsResponse
     xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0">
   <ingest:numPropertiesCreated>0</ingest:numPropertiesCreated>
   <ingest:numRecordsAffected>0</ingest:numRecordsAffected>
   <ingest:numRecordsDeleted>2</ingest:numRecordsDeleted>
</ingest:ingestRecordsResponse>
```

Note that when specifying an invalid or missing primary key record, the `deleteRecords` operation will not fail, but instead will ignore the record. In this case, the `numRecordsDeleted` element in the response will have a value of 0 (zero) and an entry (similar to the following example) is made in the Dgraph log:

```
Request: - fn:trace(, delete-records.xq:
A record with specifier (partID = SV-352) does not exist.)
```

Chapter 5

# Resetting the Data Store

The `clearMdex` and `provisionMdex` operations of the Data Ingest Web Service are intended to be used together for removing the data and configuration from the Data Store while keeping the Dgraph process of the data store running on the Oracle Endeca Server. Running these operations implies that you have exported the configuration and will import it, after clearing and provisioning the data store.

In a typical scenario, `clearMdex` is run as part of updating an existing data store without having to stop the Dgraph process or remove the configuration. This scenario implies that you export your configuration, remove all data and configuration records with `clearMdex`, provision the data store with `provisionMdex`, and import the configuration.

It is useful to note the difference between the `clearMdex` operation and the `deleteRecords` element in the `ingestRecords` operation of the Data Ingest Web Service (since both of these options let you delete records):

- `deleteRecords` deletes one or more specific data records.
- `clearMdex` deletes all data records and schema records from the data store.

While you can send these operations directly to the data store, both of these operations are utilized by the **Reset Data Store** connector in Integrator. The connector is used for removing the records and the configuration from the data store and provisioning the data store while keeping the Dgraph process for the data store running. For more information on the connector, see the *Oracle Endeca Information Discovery Integrator Components Guide*.

It is recommended to run `clearMdex` and `provisionMdex` in conjunction, using the **Reset Data Store** connector in Integrator. In addition, it is recommended to run these operations within an outer transaction. You can start an outer transaction through the **Transaction RunGraph** connector in Integrator.



## Removing all records from the data store

Use the `clearMdex` operation in the Data Ingest Web Service to remove all data records and also the schema records that define the data store configuration).

Before using this operation, ensure that you export the configuration. Also, it is assumed that you intend to remove all data records by using this operation (for example, with the intent to populate the data store with a new set of records).

To remove all data and schema records, use the `clearMdex` element in a Data Ingest Web Service request. The request should use this format:

```
<ingest:clearMdex/>
```

**Note:** If you submit this request after a Transaction Web Service request that starts an outer transaction, the `clearMdex` request must specify the outer transaction ID element as the first element in the request. If no transactions have been started, the ID element must be omitted or its value should be empty.

To remove all records from the Oracle Endeca Server:

1. Create a `clearMdex` request, similar to the example below, and send the request to the Data Ingest Web Service:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0">
 <soapenv:Header/>
     <soapenv:Body>
        <ingest:clearMdex>
           <ingest:OuterTransactionId>txId</ingest:OuterTransactionId>
        </ingest:clearMdex>
      </soapenv:Body>
   </soapenv:Envelope>
```

   This request removes the data records and the schema records in the data store.

A successful `clearMdexResponse` returned from the above sample request should return the number of records deleted, and look like this:

```
<ingest:clearMdexResponse>
 <ingest:OuterTransactionId>txId</ingest:OuterTransactionId>
  <ingest:numRecordsDeleted>175</ingest:numRecordsDeleted>
</ingest:clearMdexResponse>
```

Note that if you specify an outer transaction ID that does not match the ID of the currently running transaction, the `clearMdex` operation fails, notifying you of the transaction ID that is in progress. In addition, if no outer transactions have been started with the Transaction Web Service, but you still specify an element with the value for an ID, this request also fails.

After you have removed all the data and schema records from the data store, you want to provision it again with the default configuration settings. To provision the data store, run `provisionMdex`.

# Provisioning the data store

To provision the data store, use the `provisionMdex` operation in the Data Ingest Web Service. This operation creates the primordial records (such as PDRs and DDRs), and resets these records to their default values.

It is assumed that you run this operation after running `clearMdex`. It is also assumed that you have previously exported your configuration defined in the schema records and will import it after you run the `provisionMdex` operation.

To provision the data store and create PDRs and DDRs with their default values, use the `provisionMdex` element in a Data Ingest Web Service request. The request should use this format:

```
<ingest:provisionMdex/>
```

**Note:** If you submit the provision request after a Transaction Web Service request that starts a an outer transaction, the provision request must specify the outer transaction ID as the first element. If no outer transactions have been started, the ID element must be omitted in the request, or its value must be empty.

To provision the data store:

1. Create a `provisionMdex` request, similar to the example below and send the request to the Data Ingest Web Service:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ingest="http://www.endeca.com/MDEX/ingest/1/0">
 <soapenv:Header/>
     <soapenv:Body>
        <ingest:provisionMdex>
           <ingest:OuterTransactionId>txId</ingest:OuterTransactionId>
        </ingest:provisionMdex>
      </soapenv:Body>
   </soapenv:Envelope>
```

   This request adds the primordial schema records in the data store and sets these schema records to their defaults.

A successful `provisionMdexResponse` returned from the above sample request should look similar to this example:

```
<ingest:provisionMdexResponse>
            <ingest:OuterTransactionId>txId</ingest:OuterTransactionId>
            <ingest:numPropertiesCreated>2</ingest:numPropertiesCreated>
            <ingest:numRecordsAffected>2</ingest:numRecordsAffected>
            <ingest:numRecordsDeleted>0</ingest:numRecordsDeleted>
</ingest:provisionMdexResponse>
```

After you have run the `provisionMdex` operation, you can import your configuration and run `admin/<DataStore>?op=updateaspell` to update the spelling dictionary.

If you use **Reset Data Store** for running `clearMDEX` and `provisionMDEX` operations, then this connector also updates the spelling dictionary.

Chapter 6

# Bulk Load API

This chapter describes how to use the Bulk Load API.



## About the Bulk Load API

Data records can be added or replaced via the Dgraph's Bulk Load Interface.

Besides the Data Ingest API, the Bulk Load Interface is available to ingest records into an Endeca data store. The Bulk Load API exists in the form of a collection of Java classes in a single `endeca_bulk_load.jar` file, which is shipped in the Endeca Server's `apis` directory. The Javadoc for the Bulk Load API is located in the `apis/doc/bulk_load` directory.

### Bulk Load characteristics

The characteristics of the interface are:

- The API can load data source records only. It cannot load PDRs, DDRs, managed attribute values, the GCR, or the Dgraph configuration documents.
- Existing records in the Endeca data store are replaced, not updated. That is, the replace operation is not additive. Therefore, the key/value pair list of the incoming record will completely replace the key/value pair list of the existing record.
- A primary-key attribute (also called the record spec) is required for each record to be added or replaced.
- If an assignment is for a standard attribute (property) that does not exist in the Endeca data store, the new standard attribute is automatically created with system default values for the PDR. For these default values, see *Default values for new attributes in the data store on page 15*.

The interface rejects non-XML 1.0 characters upon ingest. That is, a valid character for ingest must be a character according to production 2 of the XML 1.0 specification. If an invalid character is detected, an exception is thrown with this error message:

```
Character <c> is not legal in XML 1.0
```

The record with the invalid character is rejected.

### Post-ingest behavior

There are two operations that must occur at some time after each bulk-load ingest:

- A merge of the ingested records to a single generation, which re-indexes the database to optimize query performance.
- A rebuild of the aspell spelling dictionary, so that the newly-added data will be available for spelling DYM and autocorrect.

The `BulkIngester` constructor's `doFinalMerge` parameter lets you set when the post-ingest merge occurs:

- If set to `true`, the merge is forced immediately after ingest. This behavior is intended to maximize query performance at the end of a single, large, homogenous data update that would occur during a regularly scheduled update window.
- If set to `false`, a merge is not forced at the end of an update, but instead relies on the regular background merge process to keep the generations in order over time. This behavior is more suitable for parallel heterogeneous data updates where low overall update latency is paramount.

The `BulkIngester` constructor's `doUpdateDictionary` parameter lets you specify when the aspell spelling dictionary is updated:

- A setting of `true` means a dictionary update is forced immediately after the ingest.
- A setting of `false` means the dictionary update is disabled. You can later update the dictionary via the `updateaspell` administrative operation, which is described in the *Oracle Endeca Server Administrator's Guide*.

If you are doing multiple, consecutive bulk-load operations, you can set both properties to `false` on all except the last one.

## Bulk Load sample program

The following sample program will be used to illustrate the Bulk Load classes and methods.

```
class BulkLoadDemo {

    Record makeProductRecord(String name, int size, double price) {

        Record.Builder recBuilder = Record.newBuilder();

        Assignment.Builder assgtBuilder = Assignment.newBuilder();
        assgtBuilder.setName("Name");
        assgtBuilder.setStringValue(name);
        assgtBuilder.setDataType(Assignment.DataType.STRING);
        Assignment nameAssgt = assgtBuilder.build();
        recBuilder.setSpec(nameAssgt);

        assgtBuilder.clear();
        assgtBuilder.setName("Size");
        assgtBuilder.setIntValue(size);
        assgtBuilder.setDataType(Assignment.DataType.INT);
        Assignment sizeAssgt = assgtBuilder.build();
        recBuilder.addAssignments(sizeAssgt);

        assgtBuilder.clear();
        assgtBuilder.setName("Price");
        assgtBuilder.setDoubleValue(price);
        assgtBuilder.setDataType(Assignment.DataType.DOUBLE);
        Assignment priceAssgt = assgtBuilder.build();
        recBuilder.addAssignments(priceAssgt);
```

```
        Record product = recBuilder.build();

        return product;
    }

    void main(int argc, String[] argv) {

        ErrorCallback errorCallback = new ErrorCallback() {
            void handleError(String reason, Record reject) {
                System.out.println("Record "
                        + reject.getSpec().getName()
                        + " rejected: " + reason);
            }
        };

        FinishedCallback finishedCallback = new FinishedCallback() {
            void handleFinished() {
                System.out.println("Finished!");
            }
        };

        AbortCallback abortCallback = new AbortCallback() {
            void handleAbort(String reason) {
                System.out.println("Aborted: " + reason);
            }
        };

        StatusCallback statusCallback = new StatusCallback() {
            void handleStatus(long recordsAdded,
                    long recordsQueued,
                    long recordsRejected,
                    String state) {
                System.out.println("Added: " + recordsAdded + "\n"
                        + "Queued: " + recordsQueued + "\n"
                        + "Rejected: " + recordsRejected + "\n"
                        + "State: " + state + "\n");
            }
        };

        BulkIngester ingester("endecaserver.example.com",
                1234,        // port
                false,       // useSSL
                true,        // doFinalMerge
                true,        // doUpdateDictionary
                90000        // timeout in ms
                errorCallback,
                finishedCallback,
                abortCallback,
                statusCallback);

        Record widget = makeProductRecord("Widget", 12, 99.95);
        Record thing = makeProductRecord("Thing", 110, 3.14);

        ingester.begin();
        ingester.sendRecord(widget);
        ingester.requestStatusUpdate();
        ingester.sendRecord(thing);
        ingester.endIngest();
    }
}
```

# Records, assignments, and data types

A `Data.Record` is the fundamental data type for loading data into an Endeca data store.

## Records and assignments

Client programs using the Bulk Load API to load data into an Endeca data store must convert the source data records from their native format into a `Record` object. Conceptually, a `Record` is a named set of key/value pairs. A key/value pair is a `Data.Assignment`. The key is a `String` and the value can be any of the supported data types listed below.

The name of the `Record` is also an `Assignment`, and can therefore also be any of these types; for this reason it is known as the Spec. The Spec serves as the primary key of the data record and each `Record` must have one.

The Spec is set on the `Record` with the `setSpec()` method of the `Data.Record.Builder` class, as shown in this snippet from the sample program:

```
Record.Builder recBuilder = Record.newBuilder();

   Assignment.Builder assgtBuilder = Assignment.newBuilder();
   assgtBuilder.setName("Name");
   assgtBuilder.setStringValue(name);
   assgtBuilder.setDataType(Assignment.DataType.STRING);
   Assignment nameAssgt = assgtBuilder.build();
   recBuilder.setSpec(nameAssgt);
```

Non-Spec key/value assignments are set with the `addAssignments()` method of the same `Data.Record.Builder` class:

```
assgtBuilder.setName("Price");
assgtBuilder.setDoubleValue(price);
assgtBuilder.setDataType(Assignment.DataType.DOUBLE);
Assignment priceAssgt = assgtBuilder.build();
recBuilder.addAssignments(priceAssgt);
```

## Builders

`Data` and `Record` objects are created via an associated `Builder` type:

- The `Data.Record.Builder` has `set()` and `clear()` methods for the spec and for the `Assignment` objects.
- The `Data.Assignment.Builder` has the same methods for its key and all of the supported data types.

All of the setter functions in the `Builder` classes return the object they were invoked on, so they can be strung together. This makes it possible, if desired, to build the entire thing on one line:

```
Data.Assignment.newBuilder().setName("SKU").setString("AB1234").setDataType(Data.Assignment.DataType.
STRING).build();
```

## Data types

The `Data.Assignment.DataType` enum contains entries for each of the data types supported by the Dgraph process. These correspond to the `Assignment.Builder` class's setter methods as follows:

| Enum DataType | Assignment.Builder setter | Dgraph data type |
|---|---|---|
| `BOOLEAN` | `setBoolValue()` | `mdex:boolean` |
| `DOUBLE` | `setDoubleValue()` | `mdex:double` |
| `DATETIME` | `setStringValue()` | `mdex:dateTime` |
| `DURATION` | `setStringValue()` | `mdex:duration` |
| `GEOCODE` | `setStringValue()` | `mdex:geocode` |
| `INT` | `setInt32Value()` | `mdex:int` |
| `INT64` | `setInt64Value()` | `mdex:long` |
| `STRING` | `setStringValue()` | `mdex:string` |
| `TIME` | `setStringValue()` | `mdex:time` |

The `setDataType()` method tells the Endeca server what the `Assignment` data type is. For example, if you call `setStringValue("foo")` and `setDataType(Data.Assignment.DataType.DOUBLE)` and include the resulting `Assignment` in a `Record` that you send to the server, the server will attempt to extract the `DOUBLE` value and get back nothing. If you do not specify a data type, the `Assignment` is invalid.

Examples of setting the data types are as follows:

```
// BOOLEAN example:
assgtBuilder.setBoolValue(false);
assgtBuilder.setDataType(Assignment.DataType.BOOLEAN);

// DOUBLE example:
assgtBuilder.setDoubleValue(99.95);
assgtBuilder.setDataType(Assignment.DataType.DOUBLE);

// DATETIME example:
assgtBuilder.setStringValue("2010-11-18T17:00:00Z");
assgtBuilder.setDataType(Assignment.DataType.DATETIME);

// DURATION example:
assgtBuilder.setStringValue("P429DT1H2M3S");
assgtBuilder.setDataType(Assignment.DataType.DURATION);

// GEOCODE example:
assgtBuilder.setStringValue("42.365615 -71.075647");
assgtBuilder.setDataType(Assignment.DataType.GEOCODE);

// INT example:
assgtBuilder.setInt32Value(128);
assgtBuilder.setDataType(Assignment.DataType.INT);

// INT64 example:
assgtBuilder.setInt64Value(92233720);
assgtBuilder.setDataType(Assignment.DataType.INT64);
```

```
// STRING example:
assgtBuilder.setStringValue("Road Bikes");
assgtBuilder.setDataType(Assignment.DataType.STRING);

// TIME example:
assgtBuilder.setStringValue("09:14:52");
assgtBuilder.setDataType(Assignment.DataType.TIME);
```

# Sending records to the data store

The `BulkIngester` class is the primary entry point for the client-side Bulk Load Interface for loading data into an Endeca data store.

`BulkIngester` makes a socket connection to the Endeca data store and spawns a thread to handle replies. Its `sendRecord()` method sends the provided record over the wire to the data store.

Clients to this interface must:

1. Define classes that implement the four callback interfaces (`ErrorCallback`, `FinishedCallback`, `AbortCallback`, and `StatusCallback`), and perform the appropriate action when their handler methods are called (which happens in the response thread).
2. Instantiate a `BulkIngester` object with the appropriate parameters required by the constructor.
3. Call the `begin()` method to start the response thread. If this is not called, an `IOException` will be thrown.
4. Call `sendRecord()` repeatedly to send `Record` objects to the Endeca data store.
5. When finished sending records, call `endIngest()` to terminate the response thread and close the socket.

## Defining callback interfaces

The `BulkIngester` constructor requires the four callback interfaces as parameters:

- `ErrorCallback` handles error conditions. The `handleError()` method
- `FinishedCallback` is called when the Dgraph reports that it has finished with the ingestion. No further records will be accepted without calling `begin()` again.
- `AbortCallback` handles abort conditions. An abort condition can happen either in `BulkIngester` or on the Dgraph.
- `StatusCallback` handles status updates, including the number of successfully ingested records and the number of rejected records.

`ErrorCallback` is especially useful as it reports the reason that a record was rejected. The sample program defines this callback as:

```
ErrorCallback errorCallback = new ErrorCallback() {
    void handleError(String reason, Record reject) {
        System.out.println("Record "
                + reject.getSpec().getName()
                + " rejected: " + reason);
    }
};
```

## Instantiating a BulkIngester object

The `BulkIngester` constructor requires ten parameters, in this order:

- `host` – the name (a `String`) of the machine on which the Endeca data store is running.
- `port` – the bulk load port (an `int`) of the Endeca data store.
- `useSSL` – a `boolean` to specify whether to use SSL for the connection.
- `doFinalMerge` – a `boolean` that specifies whether a merge is forced immediately after ingest.
- `doUpdateDictionary` – a `boolean` that specifies whether the aspell dictionary is updated immediately after ingest.
- `timeout` – the timeout in milliseconds (an `int`) for connecting to the Endeca data store.
- `errorCallback` – the `ErrorCallback` object.
- `finishedCallback` – the `FinishedCallback` object.
- `abortCallback` – the `AbortCallback` object.
- `statusCallback` – the `StatusCallback` object.

The sample program constructs the `BulkIngester` as follows:

```
BulkIngester ingester("endecaserver.example.com",
        1234,        // port
        false,       // useSSL
        true,        // doFinalMerge
        true,        // doUpdateDictionary
        90000        // timeout in ms
        errorCallback,
        finishedCallback,
        abortCallback,
        statusCallback);
```

## Beginning and ending the ingest

After the client program has made a connection to the Endeca data store, the ingest process requires the use of these `BulkIngester` methods in this order:

1. The `begin()` method starts the ingest process.
2. A series of `sendRecord()` calls actually sends the `Record` objects to the data store.
3. The `endIngest()` method terminates the ingest process.

The sample program, which ingests only two records, is coded as follows:

```
Record widget = makeProductRecord("Widget", 12, 99.95);
Record thing = makeProductRecord("Thing", 110, 3.14);

ingester.begin();
ingester.sendRecord(widget);
ingester.requestStatusUpdate();
ingester.sendRecord(thing);
ingester.endIngest();
```

Note that the `requestStatusUpdate()` method is used to retrieve the status of the ingest operation.

# Index