Oracle® Endeca Server

Data Loading Guide

Version 7.6.1 • December 2013

ORACLE

Copyright and disclaimer

Table of Contents

Preface

Oracle® Endeca Server is a hybrid search-analytical engine that organizes complex and varied data from disparate sources. At the core of Endeca Information Discovery, the unique NoSQL-like data model and in-memory architecture of the Endeca Server create an extremely agile framework for handling complex data combinations, eliminating the need for complex up-front modeling and offering extreme performance at scale. Endeca Server also supports 35 distinct languages.

About this guide

This guide describes the Data Ingest Web Service of the Oracle Endeca Server, which enables loading and deleting records in an Endeca data domain, as well as resetting the data domain. The Data Ingest Web Service is used by Integrator ETL. It can also be used by any other ETL tool for loading and managing records in the data domain.

This guide also describes the Bulk Load Interface, which can load data source records only.

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

The following conventions are used in this document.

Typographic conventions

This table describes the typographic conventions used when formatting text in this document.

Typeface	Meaning
User Interface Elements	This formatting is used for graphical user interface elements such as pages, dialog boxes, buttons, and fields.
`Code Sample`	This formatting is used for sample code phrases within a paragraph.
Variable	This formatting is used for variable values. For variables within a code sample, the formatting is *`Variable`*.
`File Path`	This formatting is used for file names and paths.

Symbol conventions

This table describes the symbol conventions used in this document.

Symbol	Description	Example	Meaning
>	The right angle bracket, or greater-than sign, indicates menu item selections in a graphic user interface.	File > New > Project	From the File menu, choose New, then from the New submenu, choose Project.

Path variable conventions

This table describes the path variable conventions used in this document.

Path variable	Meaning
`$MW_HOME`	Indicates the absolute path to your Oracle Middleware home directory, which is the root directory for your WebLogic installation.
`$DOMAIN_HOME`	Indicates the absolute path to your WebLogic domain home directory. For example, if `endeca_server_domain` is the name of your WebLogic domain, then the `$DOMAIN_HOME` value would be the `$MW_HOME/user_projects/domains/endeca_server_domain` directory.
`$ENDECA_HOME`	Indicates the absolute path to your Oracle Endeca Server home directory, which is the root directory for your Endeca Server installation.

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 overview to the Data Ingest Web Service of the Oracle Endeca Server and lists its operations.

Overview of the Data Ingest Web Service

The Data Ingest Web Service loads data into a running data domain 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 ETL, to load data into an application.

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

```
http://localhost:<port>/endeca-server/ws/ingest/<dataDomain>?wsdl
```

where the `localhost` and `port` are the host and port of the running Oracle Endeca Server, `endeca-server` is the default context root for the Endeca Server Java application running in the WebLogic Server, and *dataDomain* is the name of the Endeca data domain.

The Data Ingest Web Service enables performing these tasks:

- Provision the schema records for an initial data domain configuration.
- Reset the data domain by removing its records and schema.
- Update the spelling dictionaries for the data domain.
- Add new records to a running data domain. 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 domain (this operation is called an initial load) or to one that already has records.
- Add managed attribute values to a running data domain. If the managed values belong to a managed attribute that is not currently in the data domain, the service will also create the managed attribute.
- Modify existing records in a running data domain. You can add, remove or replace standard attribute values and managed values from already added records.
- Replace or delete records or record data from a running data domain.
- Change the names of the standard attributes in the running data domain.

The Data Ingest Web Service can modify a record multiple times in a single transaction (any combination of create, add assignments, delete assignments, and delete record). Operations within this transaction are processed in order.

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 ETL) 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).

Specify this element with the value of the outer transaction ID as the first element in the request only if a request made by the Data Ingest Web Service is started after a request to start an outer transaction has been made by the Transaction Web Service.

Do not specify this element, or leave the value of it empty, if no outer transactions have been started. If the value is empty, the request ignores the element and interprets it as not specified.

About Integrator ETL

Integrator ETL 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 domain.

The records are loaded into the data domain 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 ETL, see the *Oracle Endeca Information Discovery Integrator ETL User's 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 domain. Because it is defined by WSDL and XSD 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 References* are generated from the WSDL and XSD files that describe each Web service. They provide API-level information about Web services that are packaged with the Oracle Endeca Server. The *Oracle Endeca API References* are 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 domain. 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 61*.

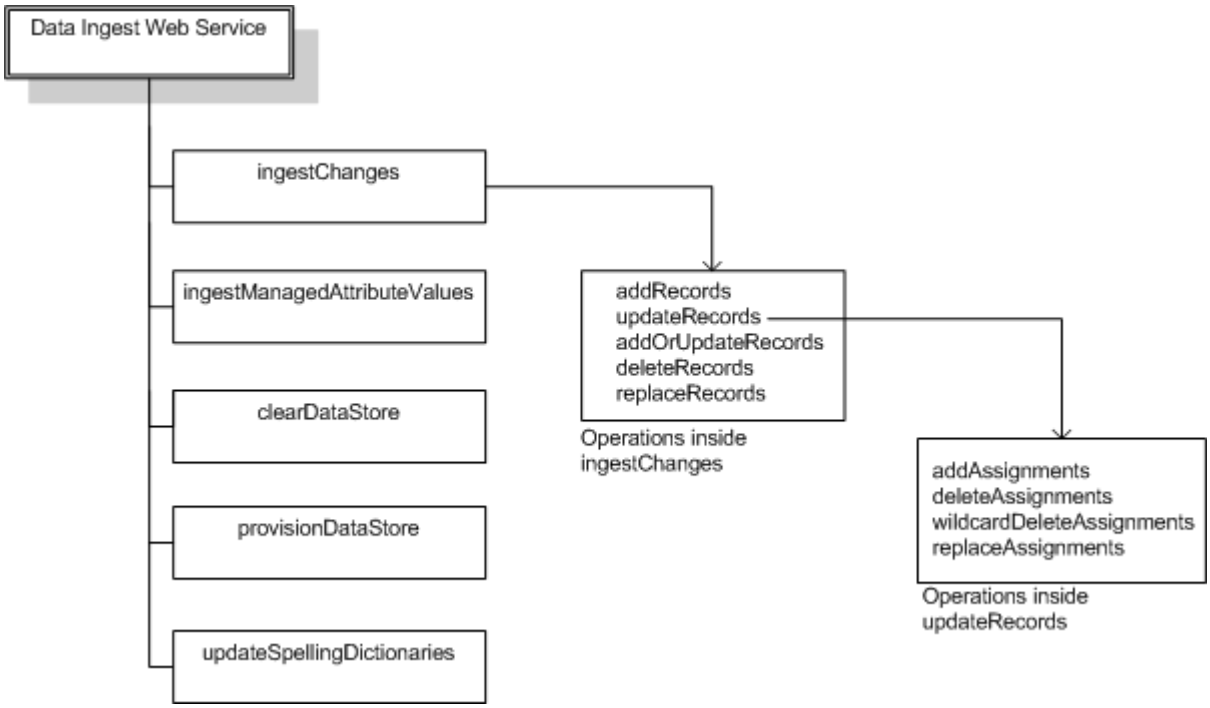
Diagram of the operations in the WSDL

This topic provides a diagram and lists the operations available in the Data Ingest Web Service.

Diagram of the Data Ingest Web Service WSDL

The following diagram provides a visual representation of the tree of operations in the Data Ingest Web Service WSDL.

Note: The structure of this guide does not follow the structure of the WSDL for this web service. In the WSDL, one of the operations — `ingestChanges` — contains many sub-operations, used for various different tasks. By comparison, this guide is broken into sections for adding records, updating records, deleting and replacing records, and adding managed attributes. When reading this guide, use the diagram in this section to locate a particular operation in the WSDL of the Data Ingest Web Service.



This diagram illustrates the Data Ingest Web Service structure:

- The Data Ingest Web Service contains five major operations: `ingestChanges`, `ingestManagedAttributeValues`, `clearDataStore`, `provisionDataStore`, and `updateSpellingDictionaries`.
- Inside `ingestChanges`, the following elements represent sub-operations: `addRecords`, `updateRecords`, `addOrUpdateRecords`, `deleteRecords`, and `replaceRecords`. Note that unlike `ingestChanges`, which contains many sub-elements, other top-level operations do not contain sub-elements representing operations.
- Further, inside the `updateRecods` element, the following elements are included, each of which also represents an operation of this web service: `addAssignments`, `deleteAssignments`, `wildcardDeleteAssignments`, and `replaceAssignments`.

To summarize, the operations in the Data Ingest Web Service are the following:

Ingest operation	Description
`ingestChanges`	Performs these functions via its five sub-operations: • `addRecords` only creates new records. • `addOrUpdateRecords` either creates new records (if the specified record specifier does not exist) or adds assignments to an existing records (if the specified record specifier does exist in the index for the data domain). • `updateRecords` updates existing records by adding, deleting, wildcard-deleting, and replacing value assignments on attributes. Can rename a standard attribute. • `replaceRecords` replaces existing records. • `deleteRecords` deletes existing records.
`ingestManagedAttributeValues`	Adds and updates assignment values for managed attributes.
`clearDataStore`	Deletes the data records and schema records from the data domain, indicating the number of records deleted. This operation must be followed by the `provisionDataStore` operation.
`provisionDataStore`	Provisions the primordial schema records in the data domain. This operation (which is typically run after the `clearDataStore` operation) adds the system records (such as PDRs and DDRs), and resets these records to their default values.
`updateSpellingDictionaries`	Updates the spelling dictionaries in the index for the data domain with the specified `name`. This operation enables spelling correction and lets you rebuild the dictionaries for spelling correction from the current index.

Important usage note

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

Recommended data domain operations

You can use the Data Ingest Web Service to perform the following actions in the data domain (the following list includes those operations that are recommended to perform and is not a full list of all supported operations):

- Add new individual records to the data domain
- Add key-value assignments on the existing records in the data domain

- Delete or replace specified record assignments
- Delete individual records, or replace them
- Change the name of a standard attribute in the running data domain
- Remove all data records from the data domain
- Provision the data domain (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 domain invalid or unusable by performing certain operations using this Web service. Here is a list of operations to use with caution:

- Removal of system (or schema) records — these are records that define your schema (such as PDR, DDR, or other system records). For example, removing a PDR may invalidate an attribute group or a search interface to which the attribute defined by this PDR belongs. As a workaround, you can remove all records and schema and then provision the data domain using the **Reset Data Domain** connector in Integrator ETL. For more information on custom connectors available in Integrator ETL, see the *Oracle Endeca Information Discovery Integrator ETL User's 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*.
- Renaming of individual standard attributes that belong to attribute groups, search interfaces, or entities (used for views in Studio). If you rename the attribute without removing it from the groups, search interfaces, or entities, you will invalidate them. Remove the standard attribute from groups, search interfaces, or entities before renaming it.

Data Ingest logging

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

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

The SOAP response provides fault and summary information. If verbose logging is turned on for the Dgraph processes of the data domain, this information, as well as the entire SOAP request, is written to the data domain's standard-out log. (You can set verbose logging in the data domain profile with the `endeca-cmd --put-dd-profile --args -v` command.)

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.

A variety of tools exist that allow you to generate client stub code from the WSDLs. 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 domain, 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 for the "books" data domain:

```
http://localhost:7001/endeca-server/ws/ingest/books?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:7001/endeca-server/ws/ingest/books
?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.

Namespaces and version for the Data Ingest Web Service

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

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: "ingest" namespace and "mdex" namespace.

The `xmlns` attribute specifies these namespaces, as in this example, which also shows the Web service's version, where `ns` is the "ingest" namespace, and `ns1` is the "mdex" namespace:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
  xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
```

The "ingest" namespace

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

```
http://www.endeca.com/MDEX/ingest/3/0
```

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

After this declaration, all DIWS elements will use the same prefix, which will be associated with the same namespace, as in this abbreviated example that uses the variable `ns` for this namespace:

```
<ns:ingestChanges>
<ns:deleteRecords>
  <ns:recordSpecifier>?</ns:recordSpecifier>
  </ns:deleteRecords>
</ns:ingestChanges>
```

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 `ns` will be used for this namespace in many examples.

The "mdex" namespace

The namespace for `mdex` elements is:

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

It is also shown here, as `xmlns:ns1`:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
  xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
```

The important elements from the `mdex` namespace used in data loading 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.

In this guide, the prefix `ns1` will be used for this namespace in the examples.

The version of the Data Ingest Web Service

The "index" namespace declaration in the Data Ingest Web Service request displays the version of this Web service:

```
xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
```

In this example, the string `/3/0` indicates the version as 3.0, where 3 is the major version, 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 3.0 and 3.1 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*.

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` XML attribute of a record element. 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.

The type is only checked by the Endeca Server when you initially create the attribute. If the attribute already exists, the Endeca Server ignores the type if it is specified, and does not perform any additional type checking to ensure that the type matches the type that exists in the Endeca Server for this attribute.

Errors from incorrect property values and types

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, as in this example:

```
Error applying updates: Unable to parse property value "19.99" for property "NumInStock"
with type "mdex:int" on record FactSalesID:569
```

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

Likewise, a type mismatch is caught whenever an assignment with the wrong type is added. When a type mismatch is encountered during a bulk load, the Dgraph will produce the following error and send it back over the response channel to the bulk load client:

```
Expected an assignment of type <type> for property <propertyName>, but got type <type> instead.
```

```
Change your schemas so that the type for this property is the same for your ingest pipeline and the
dgraph.
```

This error causes the bulk load to fail (which should be apparent from the Dgraph log), and the above message should be received by the client.

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:

```
<ns:ingestChanges>
<ns:updateRecords>
 <ns:recordSpecifier>FactSalesID='568'</ns:recordSpecifier>
  <ns:addAssignments>
   <ns1:attribute name="Description" type="mdex:string">winter tights</ns1:attribute>
  </ns:addAssignments>
 </ns:updateRecords>
</ns:ingestChanges>
```

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:

```
<ns:ingestChanges>
<ns:updateRecords>
 <ns:recordSpecifier>FactSalesID='568'</ns:recordSpecifier>
   <ns:addAssignments>
     <ns1:attribute name="Price" type="mdex:double">19.99</ns1:attribute>
     <ns1:attribute name="NumInStock" type="mdex:int">45</ns1:attribute>
     <ns1:attribute name="TotalSold" type="mdex:long">92233720</ns1:attribute>
    </ns:addAssignments>
   </ns:updateRecords>
 </ns:ingestChanges>
```

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:

```
<ns:ingestChanges>
<ns:updateRecords>
 <ns:recordSpecifier>FactSalesID='778'</ns:recordSpecifier>
   <ns:addAssignments>
     <ns1:attribute name="Location" type="mdex:geocode">42.365615 -71.075647</ns1:attribute>
    </ns:addAssignments>
   </ns:updateRecords>
 </ns:ingestChanges>
```

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:

```
<ns:ingestChanges>
<ns:updateRecords>
  <ns:recordSpecifier>FactSalesID='492'</ns:recordSpecifier>
   <ns:addAssignments>
     <ns1:attribute name="IsInStock" type="mdex:boolean">true</ns1:attribute>
     <ns1:attribute name="IsActive" type="mdex:boolean">1</ns1:attribute>
    </ns:addAssignments>
   </ns:updateRecords>
 </ns:ingestChanges>
```

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

dateTime property

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

The `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 time zone 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, 2011 in New York City, you would specify:

```
2011-11-18T17:00:00Z
```

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 time zone. Time zones are durations of hours and minutes. Time zones may be specified as positive or negative durations.

The format for a time zone 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, 2011 in New York City, you would specify:

```
2011-11-18T12:00:00+05:00
```

Note that this time represented in this example is the same as the "2011-11-18T17:00:00Z" time in the universal dateTime format.

Example of ingesting dateTime properties

The following request updates Record 506 with two dateTime properties:

```
<ns:ingestChanges>
<ns:updateRecords>
 <ns:recordSpecifier>FactSalesID='506'</ns:recordSpecifier>
 <ns:addAssignments>
  <ns1:attribute name="dT1" type="mdex:dateTime">2011-10-18T17:00:00Z</ns1:attribute>
  <ns1:attribute name="dT2" type="mdex:dateTime">2011-11-18T12:00:00+05:00</ns1:attribute>
    </ns:addAssignments>
   </ns:updateRecords>
```

```
 </ns:ingestChanges>
```

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.

The `mdex:time` property represents the hour and minutes of an instance of time, with the optional specification of fractional seconds. You can specify a time value as either a universal (UTC) time or as a local time plus a UTC time zone offset. Note that specifying just a local time is not supported.

format for universal time

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

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

where:

- *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).
- *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 time is Coordinated Universal Time (UTC, sometimes called Greenwich Mean Time). Z is the zone designator for the zero UTC offset.

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.

For example, to indicate 1:30 p.m. in New York City, you would specify:

```
18:30:00Z
```

format for local time plus UTC offset

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

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

The meanings of the hours, minutes, and seconds specifiers are the same as the universal time format. *zzzzzz* represents the time zone. Time zones are durations of hours and minutes. Time zones may be specified as positive or negative durations.

The format for a time zone 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 1:30 p.m. in New York City, you would specify:

```
13:30:00+05:00
```

Example of ingesting time properties

The following request updates Record 624 with two time properties:

```
<ns:ingestChanges>
<ns:updateRecords>
 <ns:recordSpecifier>FactSalesID='624'</ns:recordSpecifier>
   <ns:addAssignments>
     <ns1:attribute name="time1" type="mdex:time">13:25:43.261Z</ns1:attribute>
     <ns1:attribute name="time2" type="mdex:time">09:00:00+05:00</ns1:attribute>
    </ns:addAssignments>
   </ns:updateRecords>
 </ns:ingestChanges>
```

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.

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 time zone 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.

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:

```
<ns:ingestChanges>
<ns:updateRecords>
 <ns:recordSpecifier>FactSalesID='344'</ns:recordSpecifier>
   <ns:addAssignments>
     <ns1:attribute name="duration1" type="mdex:duration">P429DT1H2M3S</ns1:attribute>
     <ns1:attribute name="duration2" type="mdex:duration">P429D</ns1:attribute>
     <ns1:attribute name="duration3" type="mdex:duration">P429DT2M3.25S</ns1:attribute>
     <ns1:attribute name="duration4" type="mdex:duration">PT1H2M</ns1:attribute>
     <ns1:attribute name="duration5" type="mdex:duration">-P429DT3S</ns1:attribute>
   </ns:addAssignments>
  </ns:updateRecords>
</ns:ingestChanges>
```

Note that the `duration5` property has a negative duration value.

Default values for new attributes

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 attributes.

Default values for standard attributes

The PDR for a standard attribute that is automatically created will use the system default settings, which 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_Language`	Sets the language of the attribute's data. See the "Global PDR language ID" section below for information on the default setting for the language ID.
`mdex-property_IsPropertyValueSearchable`	`true` (the standard attribute will be enabled for value search)
`mdex-property_IsSingleAssign`	`true` (a record may have only one value assignment 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 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)

Global PDR language ID

A PDR's `mdex-property_Language` property specifies the language of that standard attribute, via a language ID (such as `en` for English or `de` for German). If a standard attribute is automatically created (by the Data Ingest Web Service or by the Bulk Load Interface), or if you do not specify a language ID when creating your attribute schema, then the value of the `mdex-property_Language` property for a PDR will be set to the system language ID value.

The value of the global PDR language ID is `unknown` by default. However, you can use the Configuration Web Service's `setPropertyDefaultLanguage` operation to change it to a value of your choice. The setting of this value is stored on disk in the index files for the specific Endeca data domain, so that it is available across restarts of that data domain. See the *Oracle Endeca Server Developer's Guide* for a list of the supported language IDs.

Note that the global PDR language ID (as set by the `setPropertyDefaultLanguage` operation) serves a different purpose than the `Language` element of the `ingestChanges` complex type. This `Language` element lets you set the language in which EQL error messages are returned and is described in *Identifying records with EQL on page 44*.

NCName format for attributes

The names of standard attributes and managed attributes in the data domain 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 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 domain 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 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:clearDataStore>
   <OuterTransactionId>MyID</OuterTransactionID>
</ingest>
```

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

Troubleshooting connection timeouts

To help prevent timeout issues during data ingest operations, use the `--net-timeout` flag for the data domain profile's configuration.

At least one Dgraph process of the Oracle Endeca Server starts for each data domain. It 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 domain.

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 the data loading operations, this timeout limit may pose problems if you have a Data Ingest Web Service 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 domain.

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 `--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. The following command for the data domain profile sets this timeout: `endeca-cmd --put-dd-profile --net-timeout <value>`.

Chapter 3

Adding New Records

This section describes how to initially load non-data records (such as PDRs, DDRs and managed attribute values), and data records into the data domain. It also describes how to ingest additional new records, and how to add managed attribute value assignments to records.

You can add new records by using either of these `ingestChanges` operations:

- The `addRecords` operation can only add new records. It is typically used during the initial loading of records into the data domain.
- The `addOrUpdateRecords` operation can either add a new record, or (if the specified primary key already exists) update an existing record. Although it can be used for initial loading of records, it is typically used to load new records after the initial load.

About adding PDRs and DDRs

PDRs and DDRs represent the non-data records. They are records that define the schema for your data records. Adding PDRs and DDRs allows you to define the characteristics of attributes on your data records.

A PDR defines a standard attribute. For example, for a standard attribute "Type", you can define whether it should allow multiple assignments, or be a single-assign attribute (allowing one value from each record). You can also define a spec for the standard attribute "Type", which is used when making assignments from this attribute on data records.

Similarly, a PDR and a DDR together define characteristics of a managed attribute. For example, each managed attribute, such as "location", has a name ("Location"), and the Spec attribute. The Spec attribute of the managed attribute is used when managed attribute values are added to the managed attribute. Other characteristics of a managed attribute specify how searching should behave on values from this attribute.

Since PDR and DDR are records (although they represent non-data records and define the schema for data records), you can add PDRs and DDRs as records, using the same `addRecords` sub-operation of the `ingestChanges` in the Data Ingest Web Service as is used for adding data records. It is important to add PDRs and DDRs before adding any data records.

For information on how to add any records (data or non-data), see *Adding new records on page 35*.

There is one important difference between adding data records and adding PDRs and DDrs. When you add data records, you can define for them any attributes you want. When you add a PDR and a DDR, it is important to know which attributes you should add. This is because the attributes on the PDR and the DDR records are defined by primordial records that exist in the index before any source data records are loaded. An easy way to determine the format for PDR and DDR records is to use `putProperties` and `PutDimensions` requests of the Configuration Web Service WSDL.

For example, the structure of `putProperties` request has this configuration:

```
<ns:putProperties>
            <!--Zero or more repetitions:-->
            <ns1:record>
               <!--Optional:-->
               <mdex-property_DisplayName>?</mdex-property_DisplayName>
               <mdex-property_IsSingleAssign>?</mdex-property_IsSingleAssign>
               <mdex-property_IsTextSearchable>?</mdex-property_IsTextSearchable>
               <mdex-property_IsUnique>?</mdex-property_IsUnique>
               <mdex-property_IsPropertyValueSearchable>?</mdex-property_IsPropertyValueSearchable>
               <mdex-property_Key>?</mdex-property_Key>
               <mdex-property_TextSearchAllowsWildcards>?</mdex-property_TextSearchAllowsWildcards>
               <mdex-property_Type>?</mdex-property_Type>
               <mdex-property_Language>?</mdex-property_Language>
               <system-navigation_Select>?</system-navigation_Select>
               <system-navigation_ShowRecordCounts>?</system-navigation_ShowRecordCounts>
               <system-navigation_Sorting>?</system-navigation_Sorting>
               <!--You may enter ANY elements at this point-->
            </ns1:record>
         </ns:putProperties>
```

This structure indicates which attributes you should add when adding a new PDR, including which of these attributes are required and which are optional. For example, for adding a PDR "location", you can add a record with these characteristics:

```
<ns:ingestChanges>
 <ns:addRecords>
   <ns:record>
     <ns1:attribute name="mdex-property_DisplayName" type="mdex:string">Location</ns1:attribute>
     <ns1:attribute name="mdex-property_IsPropertyValueSearchable" type="mdex:boolean">true<
/ns1:attribute>
     <ns1:attribute name="mdex-property_IsSingleAssign" type="mdex:boolean">false</ns1:attribute>
     <ns1:attribute name="mdex-property_IsTextSearchable" type="mdex:boolean">true</ns1:attribute>
     <ns1:attribute name="mdex-property_IsUnique" type="mdex:boolean">false</ns1:attribute>
     <ns1:attribute name="mdex-property_Key" type="mdex:string">location</ns1:attribute>
     <ns1:attribute name="mdex-property_Language" type="mdex:string">en</ns1:attribute>
     <ns1:attribute name="mdex-property_TextSearchAllowsWildcards" type="mdex:boolean">true<
/ns1:attribute>
     <ns1:attribute name="mdex-property_Type" type="mdex:string">mdex:string</ns1:attribute>
     <ns1:attribute name="system-navigation_Select" type="mdex:string">single</ns1:attribute>
     <ns1:attribute name="system-navigation_ShowRecordCounts" type="mdex:boolean">true<
/ns1:attribute>
     <ns1:attribute name="system-navigation_Sorting" type="mdex:string">record-count</ns1:attribute>
     <ns1:attribute name="system-property_GroupMembership" type="mdex:string">system_properties<
/ns1:attribute>
   </ns:record>
  </ns:addRecords>
</ns:ingestChanges>
```

Similarly, to check which attributes should be present when you need to add a DDR (since both a PDR and a DDR define each managed attribute), use the `putDimensions` structure, from the Configuration Web Service:

```
<ns:putDimensions>
            <!--Zero or more repetitions:-->
            <ns1:record>
               <mdex-dimension_Key>?</mdex-dimension_Key>
```

```
               <mdex-dimension_EnableRefinements>?</mdex-dimension_EnableRefinements>
               <!--Optional:-->
               <mdex-dimension-value_Spec>?</mdex-dimension-value_Spec>
               <!--Optional:-->
               <mdex-dimension-value_Parent>?</mdex-dimension-value_Parent>
               <mdex-dimension_IsDimensionSearchHierarchical>?<
/mdex-dimension_IsDimensionSearchHierarchical>
               <mdex-dimension_IsRecordSearchHierarchical>?<
/mdex-dimension_IsRecordSearchHierarchical>
               <!--You may enter ANY elements at this point-->
            </ns1:record>
         </ns:putDimensions>
```

This structure indicates which attributes you should add when adding a new DDR, after a corresponding PDR has already been added. For example, for adding a DDR "location", you can add:

```
<ns:ingestChanges>
 <ns:addRecords>
  <ns:record>
   <ns1:attribute name="mdex-dimension_Key" type="mdex:string">location</ns1:attribute>
   <ns1:attribute name="mdex-dimension_EnableRefinements" type="mdex:boolean">true</ns1:attribute>
   <ns1:attribute name="mdex-dimension-value_Spec" type="mdex:string">loc_spec</ns1:attribute>
   <ns1:attribute name="mdex-dimension-value_Parent" type="mdex:string">loc_parent</ns1:attribute>
  </ns:record>
 </ns:addRecords>
</ns:ingestChanges>
```

Note that `mdex-dimension-value_Spec` and `mdex-dimension-value_Parent` attributes are optional. If you specify them, as in this example, you can provide values for them. (You then use these values of Spec and Parent when adding managed attribute values.) However, if you omit the attributes from the request that adds a DDR, then they are added automatically, and assigned values according to this structure: `mdex-dimension_`*`value`*`_Spec` and `mdex-dimension_`*`value`*`_Parent`, where *`value`* is the name of the managed attribute. Such as, for the managed attribute value "location", the Spec would be named `mdex-dimension_location_Spec`, and the Parent would be named `mdex-dimension_location_Parent`.

For detailed information on properties in the PDRs and DDRs and their meanings, see the *Oracle Endeca Server Developer's Guide*.

Adding managed attribute values

Managed attribute values represent assignments on managed attributes. Once you add them to the index of the Endeca data domain, managed attribute values are represented as records. You can add managed attribute values to an empty data domain before adding any data records.

You add managed attribute values similar to how you add data records. Specifically, to load managed attribute values, you can issue one or more `ingestChanges` requests with `addRecords` sub-operations of the Data Ingest Web Service, specifying one or more `record` elements.

```
<ns:record>
   <!--Zero or more repetitions:-->
   <ns1:attribute name="?" type="?">?</ns1:attribute>
 </ns:record>
```

Each `record` element should specify the following attributes:

- The key of the managed attribute for which you are adding a managed attribute value. The managed attribute must already exist, or its PDR and DDR can be added in the same request.
- The spec of the new managed attribute value record you are adding.

- The spec of the parent managed attribute value record. This parent managed attribute value must already exist, or it can be added in the same request.

For example, the following request assumes that the `location` managed attribute already exists (and thus, its PDR and DDR are already loaded), and that the parent managed attribute value, whose spec is US, has already been added. (If you have not added PDRs and DDRs, see *About adding PDRs and DDRs on page 26*.) This request adds a new managed attribute value record, with the spec VT. The request specifies:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0" xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009
/09">
   <soapenv:Header/>
   <soapenv:Body>
    <ns:ingestChanges>
     <ns:addRecords>
      <ns:record>
       <ns1:attribute name="mdex-dimension_location_Parent" type="mdex:string">US</ns1:attribute>
       <ns1:attribute name="mdex-dimension_location_Spec" type="mdex:string">VT</ns1:attribute>
       <ns1:attribute name="mdex-dimension-value_Dimension" type="mdex:string">location<
/ns1:attribute>
      </ns:record>
     </ns:addRecords>
    </ns:ingestChanges>
   </soapenv:Body>
</soapenv:Envelope>
```

In this request, these attributes are specified:

- For the attribute `mdex-dimension-value_Dimension`, (which is the primordial property that defines a managed attribute key), — the key of the managed attribute, `location`.
- For the attribute `mdex-dimension_location_Parent`, (which is the name of the parent managed attribute value, as specified on the DDR), — the key of the parent managed attribute value, US.
- For the attribute `mdex-dimension_location_Spec`, (which is the name of the spec for the new managed attribute value record you are adding), — the spec of the managed attribute value record, VT.

Note: To obtain the attribute names of the managed attribute value spec (`mdex-dimension_location_Spec` in this example) and managed attribute value parent (`mdex-dimension_location_Parent`), issue a `getDimensions` Configuration Web Service request, specifying the managed attribute's key, for which you need this information. For example, issue a request for the managed attribute `location`.

The managed attribute values that you are adding are considered totally new records in the Endeca data domain. This means that if a managed attribute value record with the same value as the spec already exists in the data domain, the `addRecords` request will fail.

To load a managed attribute value record:

1. Make sure that you have created an Endeca data domain with the `endeca-cmd` commands of the Oracle Endeca Server, and that the data domain has been started. It is also assumed that you have not added any data records at this point.
2. Obtain the names of the managed attribute value spec and parent (if the managed attribute exists), by issuing the Configuration Web Service request for the managed attribute. For example, this request obtains information about the managed attribute `location` (the omitted namespace is `ns="http://www.endeca.com/MDEX/config/services/types/3/0"`:

   ```
   <ns:configTransaction>
      <ns:getDimensions>
         <mdex-dimension_Key>location</mdex-dimension_Key>
   ```

```
   </ns:getDimensions>
</ns:configTransaction>
```

This request returns the following information:

```
<config-types:results xmlns:config-types="http://www.endeca.com/MDEX/config/services/types/3
/0">
   <dimensions xmlns="http://www.endeca.com/MDEX/config/XQuery/2009/09">
      <record>
         <mdex-dimension-value_Parent type="mdex:string" xmlns
="">mdex-dimension_location_Parent</mdex-dimension-value_Parent>
         <mdex-dimension-value_Spec type="mdex:string" xmlns="">mdex-dimension_location_Spec<
/mdex-dimension-value_Spec>
         <mdex-dimension_EnableRefinements type="mdex:boolean" xmlns="">true<
/mdex-dimension_EnableRefinements>
         <mdex-dimension_IsDimensionSearchHierarchical type="mdex:boolean" xmlns="">true<
/mdex-dimension_IsDimensionSearchHierarchical>
         <mdex-dimension_IsRecordSearchHierarchical type="mdex:boolean" xmlns="">true<
/mdex-dimension_IsRecordSearchHierarchical>
         <mdex-dimension_Key type="mdex:string" xmlns="">location</mdex-dimension_Key>
      </record>
   </dimensions>
</config-types:results>
```

where:

- `mdex-dimension-value_Spec` has the value of the spec attribute name for managed attribute values. In this example, the value of this spec is `mdex-dimension_location_Spec`. Note that this value does not have to follow this format. When you add the managed attribute itself, you can choose any name. For example, if an attribute already exists that you want to use for this purpose, it must be of type string, unique, single-assign and have no assignments. You can also omit specifying the spec for the managed attribute values when creating the managed attribute (as in this example), because this attribute on the DDR is optional. If you don't specify it, an attribute is created and the spec's value assumes the format as in our example. For more information on managed attribute values, see the *Oracle Endeca Server Developer's Guide*.
- `mdex-dimension-location_Parent` defines the value for the parent managed attribute value. The naming convention is the same as for the spec: if the name was not specified when the DDR for the managed attribute was created, the format is the same as in our example, `mdex-dimension_location_Parent`. Alternatively, you can choose any name. For example, if an attribute already exists that you want to use for this purpose, it must be of type string, single-assign and have no assignments.

You use these attributes and their values in the subsequent `addRecords` request of the Data Ingest Web Service, which adds the managed attribute value.

3. Create an `ingest:ingestChanges` request, as in the following example, and send the request to the Data Ingest Web Service. In this request, specify the Spec for the new managed attribute value record you are adding (in our case, it is `VT`), the spec of the Parent managed attribute value (it is `US`), and the name of the associated managed attribute (`location`):

```
<ns:ingestChanges>
 <ns:addRecords>
   <ns:record>
    <ns1:attribute name="mdex-dimension_location_Parent" type="mdex:string">US<
/ns1:attribute>
    <ns1:attribute name="mdex-dimension_location_Spec" type="mdex:string">VT</ns1:attribute>
    <ns1:attribute name="mdex-dimension-value_Dimension" type="mdex:string">location<
/ns1:attribute>
    <ns1:attribute name="mdex-dimension-value_name" type="mdex:string">Vermont<
/ns1:attribute>
    <ns1:attribute name="mdex-dimension-value_synonyms" type
="mdex:string">The Green Mountain State</ns1:attribute>
```

```
    <ns1:attribute name="mdex-dimension-value_synonyms" type="mdex:string">The Ski State<
/ns1:attribute>
    <ns1:attribute name="mdex-dimension-value_rank" type="mdex:string">3</ns1:attribute>
   </ns:record>
  </ns:addRecords>
 </ns:ingestChanges>
```

The request is typically created and managed by an ETL client, or you can use soapUI.

Note that in this example, in addition to adding the managed attribute value spec (VT) and its other required characteristics (Parent and associated managed attribute), additional, optional parameters are also specified, such as the name of the managed attribute value (`Vermont`), which can be different from its Spec, the synonym of this value (you can specify one or more), and its static rank within this managed attribute.

4. After the request is made, check the `ingestChangesResponse` to determine whether the request was successful.

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

```
<ingest:ingestChangesResponse xmlns:ingest="http://www.endeca.com/MDEX/ingest/3/0">
   <ingest:numPropertiesCreated>0</ingest:numPropertiesCreated>
   <ingest:numRecordsAffected>1</ingest:numRecordsAffected>
   <ingest:numRecordsDeleted>0</ingest:numRecordsDeleted>
</ingest:ingestChangesResponse>
```

Note: When you use the `ingestChanges` operation of the Data Ingest Web Service for adding managed attribute values, it does not resolve ranking conflicts and lets you add any ranking values, even if they conflict with previously added values. If you would like the Endeca Server to automatically re-rank existing values and resolve ranking ties, use the `putManagedAttributeValues` operation of the Configuration Web Service for adding managed attribute values.

In addition to adding managed attribute values, you can also use `updateRecords` sub-operation of the Data Ingest Web Service to update assignments on the managed attribute values. However, you can update only values on the `mdex-dimension-value_name` and `mdex-dimension-value_rank` attributes, after adding the managed attribute value.

About the primary keys for data records

In the Oracle Endeca Server data model, the primary-key attribute (also known as the record specifier) is used to uniquely identify data records in the Endeca data domain.

Each data record in the Endeca data domain is uniquely identified by a unique value pair, 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 data record becomes the primary key (record specifier) of the record.

Each data record added to the Endeca data domain must have a unique assignment from exactly one primary key, so that the Oracle Endeca Server can uniquely identify it. This requirement is enforced when you are adding records to an empty data domain that does not yet have a schema (system records) defined in it.

You can use any attribute as a primary key as long as this attribute is guaranteed to be unique. An attribute is unique when no two records in a single Endeca data domain have the same value for it. (Note that by default,

a standard attribute is not unique. To make a standard attribute unique, you must update the standard attribute configuration in its PDR, before loading any records.)

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.

The PDR (Property Description Record) for the primary-key attribute must have the `mdex-property_IsUnique` attribute 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 data record in the Endeca data domain.

Each data domain must have at least one primary key standard attribute, although this primary key attribute could be different for different types of your data records. This allows the Oracle Endeca Server to handle different record types, each of which can have a meaningful identifying standard attribute. For example, a store that carries multiple types of items might identify book records by a `bookID` primary key attribute, and apparel records by an `apparelID` attribute.

Note that a data record can have only one assignment from a unique attribute. The Data Ingest Web Service will throw an error if you attempt to add two or more unique attributes to the same record. For example:

```
<ingest:ingestFault xmlns:ingest="http://www.endeca.com/MDEX/ingest/3/0">
   <ingest:errorDetail>Error applying updates: Assignment bookID: "5544" is second
      unique assignment on record Record:
      apparelID: "4455"
      bookID: "5544"</ingest:errorDetail>
</ingest:ingestFault>
```

As the example shows, the second unique assignment (`bookID`) will cause the error.

Collection spec

If you are using collections, each collection is created with its own collection spec (also called a unique property key). This collection spec follows the rules for primary keys, as it is the primary key for that collection. During ingest, this collection spec also serves as the primary key for the records in that collection. For information on creating collections, see the *Oracle Endeca Server Developer's Guide*.

Note that at least one collection (which is called a data set in Studio) must be present in an Endeca data domain before a Studio application can be configured to connect to that data domain. For this reason, it is recommended that you always ingest your source data into one or more collections.

Managed attribute value spec

If you are using managed attributes, each managed attribute value spec has its own primary key. These specs are used to uniquely identify managed attribute values in the index of the Endeca data domain. For information on how to work with managed attribute values, see the *Oracle Endeca Server Developer's Guide* (although you can also use the Data Ingest Web Service to add managed attribute values).

Adding primary-key attributes

Before you add a new record, a record must have a unique assignment from a primary-key attribute.

Adding a PDR for the primary-key attribute

When you add a new record with the Data Ingest Web Service, the service checks that the primary-key attribute already exists in the Endeca data domain. Therefore, if you have an empty data domain, before adding any records, you need to identify which attribute in your records is going to serve as the primary-key attribute, and add its PDR to the data domain.

The following example shows how to add a PDR for the primary-key attribute named `partID`. Notice that the `mdex-property_IsUnique` attribute for this PDR is set to `true`, which identifies the attribute `partID` as the primary-key attribute. The `mdex-property_IsSingle` attribute is set to `false` to ensure that only one assignment from that attribute is set on a record:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
   xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
<soapenv:Header/>
<soapenv:Body>
 <ns:ingestChanges>
  <ns:addRecords>
   <ns:record>
    <ns1:attribute name="mdex-property_Key" type="mdex:string">partID</ns1:attribute>
    <ns1:attribute name="mdex-property_DisplayName" type="mdex:string">Part ID</ns1:attribute>
    <ns1:attribute name="mdex-property_Type" type="mdex:string">mdex:string</ns1:attribute>
    <ns1:attribute name="mdex-property_Language" type="mdex:string">en</ns1:attribute>
    <ns1:attribute name="mdex-property_IsSingleAssign" type="mdex:boolean">true</ns1:attribute>
    <ns1:attribute name="mdex-property_IsUnique" type="mdex:boolean">true</ns1:attribute>
    <ns1:attribute name="mdex-property_IsTextSearchable" type="mdex:boolean">false</ns1:attribute>
    <ns1:attribute name="mdex-property_TextSearchAllowsWildcards" type="mdex:boolean">
      false</ns1:attribute>
    <ns1:attribute name="mdex-property_IsPropertyValueSearchable" type="mdex:boolean">
      true</ns1:attribute>
    <ns1:attribute name="system-navigation_Select" type="mdex:string">single</ns1:attribute>
    <ns1:attribute name="system-navigation_Sorting" type="mdex:string">record-count</ns1:attribute>
    <ns1:attribute name="system-navigation_ShowRecordCounts" type="mdex:boolean">true</ns1:attribute>
    <ns1:attribute name="system-property_GroupMembership" type="mdex:string">
      system_properties</ns1:attribute>
   </ns:record>
  </ns:addRecords>
 </ns:ingestChanges>
</soapenv:Body>
</soapenv:Envelope>
```

Now that the primary-key attribute `partID` is created in the data domain, the following example shows how to create a new record whose assignment on the primary-key attribute will be `partID=P123`. This example uses the `ingestChanges` operation of the Data Ingest Web Service with the `addRecords` element:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
     xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
  <soapenv:Header/>
   <soapenv:Body>
      <ns:ingestChanges>
        <ns:addRecords>
            <ns:record>
             <ns1:attribute name="partID" type="mdex:string">P123</ns1:attribute>
           </ns:record>
         </ns:addRecords>
       </ns:ingestChanges>
   </soapenv:Body>
```

```
</soapenv:Envelope>
```

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 following table shows the settings you should use for the primary-key attribute. Except for `mdex-property_IsUnique` which must be set to `true`, all other attributes of the primary-key attribute can use the system default settings for PDRs. If you have not yet added records to the data domain, you can change all settings on the PDR for the primary-key attribute, except for the `mdex-property_IsUnique` which must be always set to `true`:

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_Language`	Set to a supported language ID. Defaults to either `unknown` or to the language ID set by the Configuration Web Service's `setPropertyDefaultLanguage` operation.
`mdex-property_IsPropertyValueSearchable`	`true` (the attribute is enabled for value search)
`mdex-property_IsSingleAssign`	`false` (a record may have more than one value for the attribute)
`mdex-property_IsTextSearchable`	`false` (the attribute is 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 are shown for a refinement)
`system-navigation_Sorting`	`record-count` (refinements are sorted in descending order, by the number of records available for each refinement)

Adding new records

The `addRecords` sub-operation allows you to add new records to an Endeca data domain.

The `addRecords` sub-operation (of the `ingestChanges` operation) can only add new records; that is, it cannot update existing records.

The records to be added are considered totally new records in the Endeca data domain. This means that if a record with the same value as the specified primary key already exists in the data domain, the `addRecords` request will fail with an error message similar to this example:

```
<ingest:errorDetail>Error applying updates: Attempt to add a second identical assignment
to a unique property: partID="1234"</ingest:errorDetail>
```

In the example, a record with a primary key of "partID=1234" is to be added. However, a record with that primary key-value pair already exists, and (as the error says) you cannot add another identical value to a unique attribute.

Note: To extend (update) existing records, use either the `addOrUpdateRecords` sub-operation (described in *Adding records after the initial load on page 40*) or the `updateRecords` sub-operation (described in *Updating Records on page 41*).

addRecords request

An `addRecords` request uses the `ingestChanges` operation with the `addRecords` element. The record to be added can have key-value pair assignments as needed. One (and only one) of those assignments must be from a unique attribute.

Records that are being added are not available for other operations in the same request.

Note: If you submit the `addRecords` 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 `addRecords` request format is:

```
<ns:ingestChanges>
   <!--Optional:-->
   <ns:OuterTransactionId>?</ns:OuterTransactionId>
   <!--Optional:-->
   <ns:Language>en</ns:Language>
   <!--Zero or more repetitions:-->
   <ns:addRecords>
      <ns:record>
         <!--Zero or more repetitions:-->
         <ns1:attribute name="?" type="?">?</ns1:attribute>
      </ns:record>
   </ns:addRecords>
</ns:ingestChanges>
```

For example, this request adds one record, whose partID is P567, color is blue, and price is 19.99, to the data domain:

```
<ns:ingestChanges>
  <ns:addRecords>
    <ns:record>
      <ns1:attribute name="partID" type="mdex:string">P567</ns1:attribute>
      <ns1:attribute name="color" type="mdex:string">blue</ns1:attribute>
      <ns1:attribute name="price" type="mdex:double">19.99</ns1:attribute>
    </ns:record>
  </ns:addRecords>
```

```
</ns:ingestChanges>
```

The assignment for the `partID` attribute is added. This attribute has a unique assignment on a record. The request also creates the `color` and `price` attributes.

Note: When you create new standard attributes, specifying their type in the request is optional. If you do not specify the type, the standard attributes are created with the type `mdex:string`, which is the PDR default. However, if you want to create a standard attribute of a particular type, such as `mdex:double` in this example, you must explicitly specify this type when creating a new attribute.

Success response

An `ingestChangesResponse` for a successful `addRecords` request looks like this example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <ingest:ingestChangesResponse xmlns:ingest="http://www.endeca.com/MDEX/ingest/3/0">
         <ingest:numPropertiesCreated>2</ingest:numPropertiesCreated>
         <ingest:numRecordsAffected>1</ingest:numRecordsAffected>
         <ingest:numRecordsDeleted>0</ingest:numRecordsDeleted>
      </ingest:ingestChangesResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

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 domain.

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 a record with a primary key `P567` had already been added to the data domain, and you specify it again inside an `addRecords` request :

```
<ns1:attribute name="partID">P567</ns1:attribute>
```

The response would return an error similar to this:

```
<detail>
 <ingest:ingestFault xmlns:ingest="http://www.endeca.com/MDEX/ingest/3/0">
  <ingest:errorDetail>Error applying updates: Attempt to add a second
   identical assignment to a unique property: partID="P567"</ingest:errorDetail>
 </ingest:ingestFault>
</detail>
```

In this example, the reason for the error is that the record with the primary key `P567` already exists in the data domain.

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 domain.

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 domain. However, all previous valid batches have been loaded into the data domain. 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 domain, but the previous 3499 batches will be present in the data domain.

It is recommended to use outer transactions for data loading operations.

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 domain 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 domain 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:

```
<ns1:attribute name="?" type="?">?</ns1:attribute>
```

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

```
<ns1:attribute name="ItemID" type="mdex:int">247</ns1:attribute>
```

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. 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 domain 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 `ingestChanges` operation with `addRecords`, specifying one or more `record` elements.

Note: The initially-created data domain contains no primary key attribute, and no assignments on primary-key attributes. Therefore, the primary key and assignments on it must be created.

To summarize, the following actions take place during the initial load of records:

- A PDR for the primary key attribute must be added to the data domain.
- Some or all of your source records are loaded into the data domain.
- Assignments on the primary-key attribute are created by using the `attribute` element inside the `record` element of the `addRecords` operation:

  ```
  <attribute name="pKey" type="mdex:string">pKeyValue</attribute>
  ```

  where *`pKey`* is the name of the primary-key attribute (such as `partID`) and *`pKeyValue`* is the value of the primary key for the record that you are adding (such as `P775`).

An example of a full request is:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
  xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
 <soapenv:Header/>
 <soapenv:Body>
 <ns:ingestChanges>
  <ns:addRecords>
   <ns:record>
    <ns1:attribute name="mdex-property_Key" type="mdex:string">partID</ns1:attribute>
    <ns1:attribute name="mdex-property_DisplayName" type="mdex:string">Part ID</ns1:attribute>
    <ns1:attribute name="mdex-property_Type" type="mdex:string">mdex:string</ns1:attribute>
    <ns1:attribute name="mdex-property_Language" type="mdex:string">en</ns1:attribute>
    <ns1:attribute name="mdex-property_IsSingleAssign" type="mdex:boolean">false</ns1:attribute>
    <ns1:attribute name="mdex-property_IsUnique" type="mdex:boolean">true</ns1:attribute>
    <ns1:attribute name="mdex-property_IsTextSearchable" type="mdex:boolean">false</ns1:attribute>
    <ns1:attribute name="mdex-property_TextSearchAllowsWildcards" type="mdex:boolean">
     false</ns1:attribute>
    <ns1:attribute name="mdex-property_IsPropertyValueSearchable" type="mdex:boolean">
     true</ns1:attribute>
    <ns1:attribute name="system-navigation_Select" type="mdex:string">single</ns1:attribute>
    <ns1:attribute name="system-navigation_Sorting" type="mdex:string">lexical</ns1:attribute>
    <ns1:attribute name="system-navigation_ShowRecordCounts" type="mdex:boolean">true</ns1:attribute>
    <ns1:attribute name="system-property_GroupMembership" type="mdex:string">
    system_properties</ns1:attribute>
   </ns:record>
 <ns:record>
   <ns1:attribute name="partID" type="mdex:string">P789</ns1:attribute>
   <ns1:attribute name="modelNum" type="mdex:int">2562</ns1:attribute>
   <ns1:attribute name="location" type="mdex:geocode">42.365615 -71.075647</ns1:attribute>
  </ns:record>
 </ns:addRecords>
</ns:ingestChanges></soapenv:Body></soapenv:Envelope>
```

The request first creates the `partID` primary-key attribute, and then adds one new record to the data domain with the primary key `partID=P789`. The request also creates two standard attributes (`modelNum` and `location`), because they do not exist in the data domain.

Note: If you submit the `ingestChanges` 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 example above does not specify the ID and assumes that no outer transactions have been started.

To load records into an empty data domain:

1. Make sure that you have created an Endeca data domain with the Oracle Endeca Server `endeca-cmd` commands, and that the data domain has been started.
2. Create an `ingest:ingestChanges` 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, or you can use soapUI.
3. After the request is made, check the `ingestChangesResponse` to determine whether the request was successful.

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

```
<ingest:ingestChangesResponse xmlns:ingest="http://www.endeca.com/MDEX/ingest/3/0">
 <ingest:numPropertiesCreated>2</ingest:numPropertiesCreated>
 <ingest:numRecordsAffected>1</ingest:numRecordsAffected>
 <ingest:numRecordsDeleted>0</ingest:numRecordsDeleted>
</ingest:ingestChangesResponse>
```

Adding managed attribute value assignments to records

Managed attribute values are assigned to records similar to standard attributes.

The `addRecords` sub-operation can add managed attribute values to records. Likewise, the `updateRecords` and `addOrUpdateRecords` sub-operations (when used with a list of `addAssignments` elements) can add also add managed attribute values to records.

The syntax for adding the managed attribute value is:

```
<attribute name="maName">maValueSpec</attribute>
```

where:

- *maName* is the name of the managed attribute (such as "Component" in the example below).
- *maValueSpec* is the Spec, or the unique string identifier for the managed attribute value, as set by the value of the `mdex-dimension_value_Spec` attribute for the specific managed attribute value.

Note that like standard attribute assignments, assignments from managed attribute values on records can be updated, replaced, and deleted by `ingestChanges` operations.

Example

This example assumes that the PDR for the `Component` managed attribute has a multi-assign setting, and the `Derailleur`, `Tire`, and `Michelin` managed attribute values have been created. The `Color` standard attribute is also assumed to exist:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
   xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:ingestChanges>
         <ns:addRecords>
            <ns:record>
              <ns1:attribute name="partID">8344</ns1:attribute>
              <ns1:attribute name="Component">Derailleur</ns1:attribute>
              <ns1:attribute name="Component">Tire</ns1:attribute>
              <ns1:attribute name="Component">Michelin</ns1:attribute>
              <ns1:attribute name="Color">blue</ns1:attribute>
           </ns:record>
         </ns:addRecords>
      </ns:ingestChanges>
   </soapenv:Body>
</soapenv:Envelope>
```

A query for this record should return the following results:

```
<cs:RecordListEntry>
   <cs:Record>
      <cs:attribute name="Color" type="mdex:string">blue</cs:attribute>
      <cs:attribute name="Component" type="mdex:string" displayName="derailleur gears">Derailleur<
/cs:attribute>
      <cs:attribute name="Component" type="mdex:string" displayName="Michelin brand">Michelin<
/cs:attribute>
      <cs:attribute name="Component" type="mdex:string" displayName="bicycle tires">Tire<
/cs:attribute>
      <cs:attribute name="partID" type="mdex:int">8344</cs:attribute>
   </cs:Record>
   <cs:ComputedProperties/>
</cs:RecordListEntry>
<cs:DimensionHierarchy>
   <cs:DimensionValueWithPath>
```

```
      <cs:DimensionValue DimensionName="Component" Spec="Derailleur">derailleur gears<
/cs:DimensionValue>
      <cs:DimensionValue DimensionName="Component" Spec="/">Parts</cs:DimensionValue>
   </cs:DimensionValueWithPath>
   <cs:DimensionValueWithPath>
      <cs:DimensionValue DimensionName="Component" Spec="Michelin">Michelin brand</cs:DimensionValue>
      <cs:DimensionValue DimensionName="Component" Spec="Tire">bicycle tires</cs:DimensionValue>
      <cs:DimensionValue DimensionName="Component" Spec="/">Parts</cs:DimensionValue>
   </cs:DimensionValueWithPath>
   <cs:DimensionValueWithPath>
      <cs:DimensionValue DimensionName="Component" Spec="Tire">bicycle tires</cs:DimensionValue>
      <cs:DimensionValue DimensionName="Component" Spec="/">Parts</cs:DimensionValue>
   </cs:DimensionValueWithPath>
</cs:DimensionHierarchy>
```

Adding records after the initial load

This topic describes the `addOrUpdateRecords` sub-operation.

The `addOrUpdateRecords` sub-operation allows you to add more records to the data domain at any time after the initial loading of records is complete. `addOrUpdateRecords` works as follows:

- If the specified primary key (record specifier) does not exist in the current data set, the record is added with the primary key and the specified assignments. This mode is like the `addRecords` sub-operation.
- If the specified primary key already exists in the current data set, the record is updated with the new assignments. This mode is like the `updateRecords` sub-operation.

Adding more records after the data domain 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 `ingestChanges` operation with either the `addRecords` sub-operation (described earlier in this chapter) or the `addOrUpdateRecords` sub-operation (described in this topic).
- You identify new records by adding assignments on their 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 update other, existing records with the same `ingestChanges` request, by using the `updateRecords` sub-operation.
- Records that are being added are not available for other operations in the same request.
- You can delete other records in the same request, by using the `deleteRecords` sub-operation.

addOrUpdateRecords request

The `addOrUpdateRecords` basic syntax is:

```
<ingestChanges>
   <addOrUpdateRecords>
      <recordSpecifier>
         <attribute name="?" type="?">?</attribute>
      </recordSpecifier>
      <addAssignments>
```

```
         <attribute name="?" type="?">?</attribute>
      </ns:addAssignments>
   </addOrUpdateRecords>
</ingestChanges>
```

where:

- The `recordSpecifier` element contains the name and value of the primary-key attribute. Note that the primary-key attribute must already exist.
- The `addAssignments>` element contains the names and values of the attributes to be assigned to the record. If a specified attribute does not exist, it is created with default system values.

For example, this request adds one record with two assignments to the data domain:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
  xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:ingestChanges>
          <ns:addOrUpdateRecords>
            <ns:recordSpecifier>
               <ns1:attribute name="partID">8345</ns1:attribute>
            </ns:recordSpecifier>
            <ns:addAssignments>
               <ns1:attribute name="color">blue</ns1:attribute>
               <ns1:attribute name="price">29.99</ns1:attribute>
            </ns:addAssignments>
         </ns:addOrUpdateRecords>
      </ns:ingestChanges>
   </soapenv:Body>
</soapenv:Envelope>
```

Note that none of the key-value assignments specify an attribute type (such as the type for the `price` attribute, which is `mdex:double`). This is because all the attributes already exist in the data domain and therefore already have a type that was assigned when they were initially created (unless specified, the default string type was used). The type of the assignment attribute value must match the type of the attribute.

Note: If you submit the `ingestChanges` 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.

Chapter 4

Updating Records

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

About updates

This topic will describe how you can incrementally update the records in an Endeca data domain.

Using the `updateRecords` sub-operation of `ingestChanges`, you can perform the following types of incremental updates:

- Update an existing record by adding standard attribute and/or managed attribute value assignments.
- Update an existing record by replacing or removing standard attribute and/or managed attribute value assignments.

Note that the `addOrUpdateRecords` sub-operation, which can also update an existing record with new assignments, is described in *Adding records after the initial load on page 40*.

How updates are applied

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 domain 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 domain 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.

If an attribute is single-assign, that is, a standard attribute whose PDR has the `mdex-property_IsSingleAssign` set to `true`, you cannot add a second value to it. 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.

Order of update operations

An `ingestChanges` request can contain all types of updates (delete records, delete assignments, wildcard delete assignments, and add assignments). Before any such request is performed, some operations are broken down into their component operations:

- Record updates are broken down into assignment additions, deletions, wildcard deletions, and replacements.
- Record replacements are broken down into a record deletion and record addition.
- Assignment replacements are broken down into a wildcard deletion and assignment addition.

Once these operations are broken down, all of the remaining operations are sorted into groups by type, and each group is processed in the following order:

1. `deleteRecords` requests
2. `deleteAssignments` requests
3. `wildcardDeleteAssignments` requests
4. `addAssignments` requests
5. `addRecords` requests.

If a record attribute or attribute assignment is specified in the `wildcardDeleteAssignments` 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.

If you are renaming a standard attribute, you can include only this operation inside `updateRecords` in `ingestChanges`. (Additional `updateRecords` are allowed in the same request, and can contain other updates, but not for the attribute that is being renamed.)

Affected records with update operations

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

Identifying records with EQL

To identify one or more records you want to update or delete, you use EQL expressions.

About the record set specifier

Several `ingestChanges` operations that update or delete existing records require a ***record set specifier*** to determine the set of records to operate on. The record set specifier is evaluated from an EQL expression inside the `recordSpecifier` element of the operation.

The EQL expressions use one or more record attributes that identify a set of records that will be updated or deleted by the operation. The record attributes specified in the EQL expression can be any attribute on a record, including the record's primary-key attribute.

Note, however, that the primary-key attribute does have to be used in the EQL expression. For example, assume that your records have an attribute named "color" with values such as "red", "blue", and so on. You can then use an EQL expression that specifies that all records that have a "color=red" assignment will be deleted.

Keep in mind that an EQL expression can also be evaluated to identify zero records, in which case the operation will be performed but no updates or deletes will be actually be done. In this case, the `numRecordsAffected` element in the `ingestChangesResponse` will show that 0 (zero) records were affected.

For full information on EQL syntax, see the *Oracle Endeca Server EQL Guide*.

Identifying a single record with EQL

This example contains a `recordSpecifier` element with the EQL syntax that identifies a single record. In this data set, the primary-key attribute is the attribute named `partID` of type string. The following expression identifies a record with the primary key `P123`:

```
<recordSpecifier>"partID" = 'P123'</recordSpecifier>
```

where

```
"partID" ='P123'
```

is an EQL expression on a string value.

Similarly, if the primary-key attribute is the attribute named `modelNum` of type integer, then the EQL expression for identifying a record with the primary key 45 is:

```
<recordSpecifier>"modelNum" = 45</recordSpecifier>
```

Note: While these examples illustrate how to utilize a primary-key attribute in EQL, you can use an EQL expression on any attribute in your data set when updating or deleting records. When adding new records, you identify them by the EQL expression that specifies the values for the primary-key attribute and not any other attribute.

Identifying multiple records with EQL

This example identifies all records for which the value of the integer attribute `Num` is greater than 45:

```
<recordSpecifier>"Num" > 45</recordSpecifier>
```

The following example identifies those records for which the value of the attribute `price` (of type double) is greater than 19.99 and less than 49.99, non-inclusive:

```
<recordSpecifier>("price" &gt; 19.99) AND ("price" &lt; 49.99)</recordSpecifier>
```

Note: The greater than and less than signs in this example are escaped to ensure that the XML syntax of this request can be successfully parsed.

In this example, the EQL expression identifies those records for which assignment values exist on the attribute `tag`:

```
<recordSpecifier>"tag" IS NOT NULL</recordSpecifier>
```

The examples above assume that the attribute to be compared is a single-assign attribute. For multi-assign attributes, make sure you use the functions and operators that work with sets, as in this example:

```
<recordSpecifier>SOME i IN NumRevs SATISFIES (i > 45)</recordSpecifier>
```

For more information on using multi-assign attributes with EQL, see the *Oracle Endeca Server EQL Guide*.

Setting a language ID for EQL error messages

If the syntax of an EQL expression is incorrect, the Data Ingest Web Service returns an EQL syntax error message. The `ingestChanges` complex type has an optional `Language` element that lets you set the language in which EQL error messages are returned. The element looks like this:

```
<ns:ingestChanges>
   <!--Optional:-->
   <ns:OuterTransactionId>?</ns:OuterTransactionId>
   <!--Optional:-->
   <ns:Language>en</ns:Language>
   <!--Zero or more repetitions:-->
   <ns:updateRecords>
   ...
   </ns:updateRecords>
   ...
</ns:ingestChanges>
```

The supported languages and their corresponding language IDs are:

- Chinese (simplified): `zh_CN`
- Chinese (traditional): `zh_TW`
- English: `en`
- French: `fr`
- German: `de`
- Italian: `it`
- Japanese: `ja`
- Korean : `ko`
- Portuguese: `pt`
- Spanish: `es`

The `en` (English) language ID is the default.

Adding record assignments

Records in the data domain can be updated with one or more new assignments for standard attributes and managed values.

The `updateRecords` sub-operation, when used with a list of `addAssignments` elements, allows you to update existing records in the data domain 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 default values for PDRs.

Because the Dgraph process 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 attribute 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 domain. In this case, the Data Ingest Web Service automatically creates the managed attribute.

Update sub-operation types

For update scenarios, the `ingestChanges` operation has two update sub-operations that support the `addAssignments` element. Which one you use depends on your needs:

- The `updateRecords` sub-operation can only update existing records, but cannot add new records. That is, if the specified primary key is not found, then the operation does nothing.
- The `addOrUpdateRecords` sub-operation will first attempt to update existing records, just like the `updateRecords` sub-operation. But if the specified primary key is not found, then the operation's fallback behavior is to act like the `addRecords` sub-operation and create new records based on the values of the specified primary keys and added assignment values.

This chapter will document the `updateRecords` sub-operation. For a description of `addOrUpdateRecords`, see *Adding records after the initial load on page 40*.

updateRecords request

You use the `addAssignments` element in an `updateRecords` 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:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
   xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:ingestChanges>
         <ns:updateRecords>
            <ns:recordSpecifier>?</ns:recordSpecifier>
```

```
              <ns:addAssignments>
              <ns1:attribute name="?">?</ns1:attribute>
            </ns:addAssignments>
          </ns:updateRecords>
       </ns:ingestChanges>
   </soapenv:Body>
</soapenv:Envelope>
```

For example, this `updateRecords` request updates a record (with the primary key P123) with one standard attribute `modelNum` of type integer:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
  xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
  <soapenv:Header/>
   <soapenv:Body>
      <ns:ingestChanges>
        <ns:updateRecords>
          <ns:recordSpecifier>"partID" = 'P123'</ns:recordSpecifier>
              <ns:addAssignments>
              <ns1:attribute name="modelNum" type="mdex:int">2562</ns1:attribute>
            </ns:addAssignments>
         </ns:updateRecords>
       </ns:ingestChanges>
   </soapenv:Body>
</soapenv:Envelope>
```

Note that the attribute type of the `modelNum` standard attribute is specified, because the standard attribute does not exist and therefore will be created as part of the request.

Note: If you submit the `ingestChanges` 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 domain by removing standard attribute and managed value assignments.

The `updateRecords` sub-operation of `ingestChanges` includes two elements that delete assignments from records in the data domain:

- `deleteAssignments`
- `wildcardDeleteAssignments`

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 domain; they are removed only from the specified records.

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

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

Assignment deletion cannot be combined in the same request with any operation that would modify the assignments to be deleted.

Note: If you submit the `ingestChanges` 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 `updateRecords` sub-operation removes individual standard attribute and/or managed value assignments from the records in your data domain, but does not otherwise affect the record. You can remove one or more assignments in the same request.

To remove individual assignments:

- Indicate which records should be affected by including the EQL filter inside the `recordSpecifier` element in your request.
- Specify the attributes from which assignment values should be deleted for these records.

The `deleteAssignments` request format is:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
  xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:ingestChanges>
         <ns:updateRecords>
            <ns:recordSpecifier>?</ns:recordSpecifier>
              <ns:deleteAssignments>
                 <ns1:attribute name="?">?</ns1:attribute>
            </ns:deleteAssignments>
          </ns:updateRecords>
      </ns:ingestChanges>
   </soapenv:Body>
</soapenv:Envelope>
```

where the string that you add to a `recordSpecifier` element is an EQL expression identifying records.

The following example illustrates this request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
  xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:ingestChanges>
         <ns:updateRecords>
            <ns:recordSpecifier>"price" > 45</ns:recordSpecifier>
              <ns:deleteAssignments>
                 <ns1:attribute name="color">purple</ns1:attribute>
            </ns:deleteAssignments>
          </ns:updateRecords>
      </ns:ingestChanges>
   </soapenv:Body>
</soapenv:Envelope>
```

The example removes one value, `purple`, of the `color` standard attribute assignment for all records whose values for the `price` attribute are higher than 45.

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

```
<ingest:ingestChangesResponse
xmlns:ingest="http://www.endeca.com/MDEX/ingest/3/0">
  <ingest:numPropertiesCreated>0</ingest:numPropertiesCreated>
  <ingest:numRecordsAffected>4</ingest:numRecordsAffected>
  <ingest:numRecordsDeleted>0</ingest:numRecordsDeleted>
```

```
</ingest:ingestChangesResponse>
```

The `numRecordsAffected` element in the response shows that four records were successfully modified.

wildcardDeleteAssignments request

The `wildcardDeleteAssignments` element of the `updateRecords` element in the `ingestChanges` operation removes all assignments from the same standard attribute or managed value at once. This operation, similar to assignment deletion, cannot be combined with any operation that modifies the assignments to be deleted.

The request must use EQL statement inside the `recordSpecifier` element to identify the record to be updated, and also list the attribute name from which all assignments must be deleted. The `wildcardDeleteAssignments` request format is:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
  xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:ingestChanges>
          <ns:updateRecords>
            <ns:recordSpecifier>?</ns:recordSpecifier>
               <ns:wildcardDeleteAssignments>
                <ns1:attribute name="?"/>
               </ns:wildcardDeleteAssignments>
            </ns:updateRecords>
       </ns:ingestChanges>
   </soapenv:Body>
</soapenv:Envelope>
```

To remove all assignments on one 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 for the record whose `partID` is `P4000`:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
  xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:ingestChanges>
          <ns:updateRecords>
            <ns:recordSpecifier>"partID" = 'P4000'</ns:recordSpecifier>
               <ns:wildcardDeleteAssignments>
                <ns1:attribute name="color"/>
               </ns:wildcardDeleteAssignments>
            </ns:updateRecords>
       </ns:ingestChanges>
   </soapenv:Body>
</soapenv:Envelope>
```

In the example, if record P4000 had six assignments from the `color` standard attribute, then all six assignments would be removed.

If successful, the operation returns the following `ingestChangesResponse`, indicating that one record was affected:

```
<ingest:ingestChangesResponse xmlns:ingest="http://www.endeca.com/MDEX/ingest/3/0">
   <ingest:numPropertiesCreated>0</ingest:numPropertiesCreated>
   <ingest:numRecordsAffected>1</ingest:numRecordsAffected>
   <ingest:numRecordsDeleted>0</ingest:numRecordsDeleted>
</ingest:ingestChangesResponse>
```

Replacing record assignments

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

The `updateRecords` sub-operation of `ingestChanges` includes the `replaceAssignments` element that replaces assignments from records in the data domain.

This element replaces standard attribute assignments as well as managed value assignments. Similar to record replacement, if there are no assignments already in the data domain that match the new assignments, the assignment is still added.

Assignment replacement cannot be combined with any other operation that would modify the assignments to be replaced.

The `updateRecords` operation allows you to have only one `recordSpecifier` element. This element is expected to contain an EQL expression identifying one or more records. Therefore, if you combine more than one updating element inside an `updateRecords` operation, it is worth remembering that all updating operations (additions, deletions, and replacements) will run only on records identified by the EQL expression inside `recordSpecifier`.

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 attribute type.
- Any standard attribute that is not specifically typed will be treated by default as a string type.

If an attribute is multi-select and has more than one assignment value on some of the records, the `replaceAssignments` element deletes all of these assignments and adds the one assignment that you specify.

The `replaceAssignments` request can also rename an attribute in your data domain. For information, see *Changing the name of a standard attribute on page 51*.

Important: Do not to use `replaceAssignments` to modify the system-created attributes in the data domain. For example, do not use it to set the system `mdex-property_IsSingleAssign` property to `false` for all attributes. Doing so may have some undesired effect on your application, such as not being able to create views correctly in Studio.

If you submit the `ingestChanges` 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.

replaceAssignments request

The `replaceAssignments` element of the `updateRecords` sub-operation removes individual standard attribute and/or managed value assignments from the specified records in your data domain, and replaces them with new assignment values. You can replace one or more assignments in the same request.

To replace individual assignments:

- Indicate which records should be affected by including the EQL filter inside the `recordSpecifier` element in your request.
- Specify the attributes for which assignment values should be replaced for these records.

The `replaceAssignments` request format is:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
  xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:ingestChanges>
         <ns:updateRecords>
            <ns:recordSpecifier>?</ns:recordSpecifier>
            <ns:replaceAssignments>
               <ns1:attribute name="?">?</ns1:attribute>
            </ns:replaceAssignments>
         </ns:updateRecords>
      </ns:ingestChanges>
   </soapenv:Body>
</soapenv:Envelope>
```

where the string that you add to a `recordSpecifier` element is an EQL expression identifying records.

The following example illustrates this request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:ingestChanges>
         <ns:updateRecords>
            <ns:recordSpecifier>"price" > 5</ns:recordSpecifier>
            <ns:replaceAssignments>
               <ns1:attribute name="color">black</ns1:attribute>
            </ns:replaceAssignments>
         </ns:updateRecords>
      </ns:ingestChanges>
   </soapenv:Body>
</soapenv:Envelope>
```

The example replaces all values of the `color` standard attribute assignment for all records whose values for the `price` attribute are higher than 5 with the value `black`.

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

```
<ingest:ingestChangesResponse xmlns:ingest="http://www.endeca.com/MDEX/ingest/3/0">
    <ingest:numPropertiesCreated>0</ingest:numPropertiesCreated>
    <ingest:numRecordsAffected>7</ingest:numRecordsAffected>
    <ingest:numRecordsDeleted>0</ingest:numRecordsDeleted>
</ingest:ingestChangesResponse>
```

The `numRecordsAffected` element in the response shows that seven records were successfully modified.

Changing the name of a standard attribute

The Data Ingest Web Service allows you to rename a standard attribute on records in a running data domain.

You use the `replaceAssignments` element in an `updateRecords` to rename attributes.

When you rename an attribute, the Data Ingest Web Service removes the name of the attribute that was previously specified in the `mdex-property_Key` of the PDR of this attribute, and replaces it with the new name. All records retain assignments on this attribute, and the Dgraph process of the Oracle Endeca Server continues to run.

Important: Before renaming a standard attribute, remove it from groups or entities (used for views in Studio). If you rename the attribute without removing it from groups or entities, you will invalidate these groups or entities.

You can rename any standard attribute on the records in your data domain. You cannot rename a managed attribute (except by renaming the associated standard attribute). You also cannot rename an attribute on any system record, such as PDRs, DDRs, or system attributes that define groups.

You cannot include any other updates inside the `updateRecords` request for renaming an attribute. Also, if the entire `ingestChanges` request contains additional `updateRecords` elements, they cannot be run on the same attribute that is being renamed.

The request must specify the EQL expression that correctly identifies the name of the attribute to be replaced, using this request format:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
 <soapenv:Header/><soapenv:Body>
  <ns:ingestChanges>
   <ns:updateRecords>
    <ns:recordSpecifier>"mdex-property_Key"='oldName'</ns:recordSpecifier>
     <ns:replaceAssignments>
      <ns1:attribute name="mdex-property_Key">newName</ns1:attribute>
    </ns:replaceAssignments>
   </ns:updateRecords>
  </ns:ingestChanges>
 </soapenv:Body>
</soapenv:Envelope>
```

where:

- `"mdex-property_Key"='oldName'` is the EQL expression that locates the PDR of your attribute, and identifies the current assignment value, `oldName`, on the `mdex-property_Key` of the PDR.
- `newName` is the new name for the attribute.

If you submit the `ingestChanges` 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 rename an attribute:

1. Ensure that both the data domain's Dgraph process and the Data Ingest Web Service are running.
2. Create an `updateRecords` request, similar to the example below that renames an attribute `partID` with an attribute `SKU`, and send the request to the Data Ingest Web Service.

   ```
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
   xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
      <soapenv:Header/>
      <soapenv:Body>
         <ns:ingestChanges>
            <ns:updateRecords>
               <ns:recordSpecifier>"mdex-property_Key"='partID'</ns:recordSpecifier>
               <ns:replaceAssignments>
                  <ns1:attribute name="mdex-property_Key">SKU</ns1:attribute>
   ```

```
            </ns:replaceAssignments>
         </ns:updateRecords>
      </ns:ingestChanges>
   </soapenv:Body>
</soapenv:Envelope>
```

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

A successful `ingestChangesResponse` returned from the above sample request should look like this, indicating that one record has been affected:

```
<ingest:ingestChangesResponse xmlns:ingest="http://www.endeca.com/MDEX/ingest/3/0">
   <ingest:numPropertiesCreated>0</ingest:numPropertiesCreated>
   <ingest:numRecordsAffected>1</ingest:numRecordsAffected>
   <ingest:numRecordsDeleted>0</ingest:numRecordsDeleted>
</ingest:ingestChangesResponse>
```

Chapter 5

Deleting or Replacing Records

This section describes how to delete or replace records in the Endeca data domain.

You can remove or replace existing records by using these `ingestChanges` operations:

- `deleteRecords` operation
- `addRecords` operation

Note that both operations use a record set specifier to determine the set of records to operate on. The record set specifier is evaluated from an EQL expression in the `recordSpecifier` element of the operation. For more information on the record set specifier, see *Identifying records with EQL on page 44*.

Deleting records

The Data Ingest Web Service allows you to delete records from a running data domain.

The `deleteRecords` operation of `ingestChanges` allows you to create a request to delete a record. The request must specify the EQL expression that correctly identifies one or more records to be deleted, using this request format:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
  xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:ingestChanges>
        <ns:deleteRecords>
            <ns:recordSpecifier>?</ns:recordSpecifier>
         </ns:deleteRecords>
       </ns:ingestChanges>
   </soapenv:Body>
</soapenv:Envelope>
```

Multiple records can be deleted in the same request. A records deletion operation cannot be combined in the same request with any operation that would try to modify the records to be deleted.

If you submit the `ingestChanges` 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 domain:

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

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:ingestChanges>
        <ns:deleteRecords>
            <ns:recordSpecifier>"partID" = 'P123'</ns:recordSpecifier>
          </ns:deleteRecords>
       </ns:ingestChanges>
   </soapenv:Body>
</soapenv:Envelope>
```

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

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

```
<ingest:ingestChangesResponse
    xmlns:ingest="http://www.endeca.com/MDEX/ingest/3/0">
   <ingest:numPropertiesCreated>0</ingest:numPropertiesCreated>
   <ingest:numRecordsAffected>0</ingest:numRecordsAffected>
   <ingest:numRecordsDeleted>1</ingest:numRecordsDeleted>
</ingest:ingestChangesResponse>
```

Note that when specifying a record with an invalid or missing primary key, the `deleteRecords` sub-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):

```
<ingest:ingestChangesResponse ...>
   ...
   <ingest:numRecordsDeleted>0</ingest:numRecordsDeleted>
</ingest:ingestChangesResponse>
```

Replacing records

The Data Ingest Web Service allows you to replace records in a running data domain.

The `updateRecords` operation of `ingestChanges` includes the `replaceRecords` element to replace records.

When you replace a record, the Data Ingest Web Service first deletes the record (along with all of its assignments on all attributes) and then adds a new record with the attribute assignments that you specify for the record, including a required assignment on a primary-key attribute.

The request must specify the EQL expression that correctly identifies one or more records to be replaced, using this request format:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:ingestChanges>
          <ns:replaceRecords>
            <ns:recordSpecifier>?</ns:recordSpecifier>
            <ns:record>
               <ns1:attribute name="?" type="mdex:string">?</ns1:attribute>
             </ns:record>
        </ns:replaceRecords>
      </ns:ingestChanges>
   </soapenv:Body>
</soapenv:Envelope>
```

Record replacement, since it is the combination of a deletion and an addition, cannot be combined in the same request with any operation that would modify the records to be deleted. Similarly, the record to be added is not available for any operations in the same request.

If you submit the `ingestChanges` 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 replace a record in the data domain:

1. Ensure that both the data domain's Dgraph process and the Data Ingest Web Service are running.
2. Create a `replaceRecords` request, similar to the example below that replaces one record, and send the request to the Data Ingest service.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:ingestChanges>
          <ns:replaceRecords>
            <ns:recordSpecifier>"partID" = 'P345'</ns:recordSpecifier>
            <ns:record>
              <ns1:attribute name="partID">P345</ns1:attribute>
               <ns1:attribute name="color" type="mdex:string">green</ns1:attribute>
             <ns1:attribute name="price">100</ns1:attribute>
            </ns:record>
         </ns:replaceRecords>
      </ns:ingestChanges>
   </soapenv:Body>
</soapenv:Envelope>
```

Note: When you are replacing records, you are first deleting a record and all its assignments on all attributes (including an assignment on a primary-key attribute, which is required), and then adding assignments on those attributes that are listed inside the `record` element. Therefore, to ensure that an update operation succeeds, always include the primary-key attribute in the description of the record that is being updated. In this example, the primary-key attribute is `partID`, and even though its assignment for this record is not being replaced, this attribute must be included in the record's description to make the update operation valid.

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

A successful `ingestChangesResponse` returned from the above sample request should look like this, indicating that one record has been affected:

```
<ingest:ingestChangesResponse xmlns:ingest="http://www.endeca.com/MDEX/ingest/3/0">
   <ingest:numPropertiesCreated>0</ingest:numPropertiesCreated>
   <ingest:numRecordsAffected>1</ingest:numRecordsAffected>
   <ingest:numRecordsDeleted>1</ingest:numRecordsDeleted>
</ingest:ingestChangesResponse>
```

Note that if you incorrectly identify records, the `replaceRecords` request will not fail, but instead will delete zero records and add one record (if using the example above). In this case, the `numRecordsDeleted` element in the response will have a value of 0, and `numRecordsAffected` will have a value of 1.

Chapter 6

Resetting the Data Domain and Updating Spelling Dictionaries

This section describes how to reset the data domain and update its spelling dictionaries.

Resetting the Data Domain

The `clearDataStore` and `provisionDataStore` operations of the Data Ingest Web Service are intended to be used together for removing the data and configuration from the data domain while keeping the Dgraph process of the data domain 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 domain.

Note: Clearing the data domain without provisioning it again leaves the data domain in an invalid state, preventing all further operations, including importing the configuration.

In a typical scenario, `clearDataStore` is run as part of updating an existing data domain 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 `clearDataStore`, provision the data domain with `provisionDataStore`, and import the configuration.

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

It is recommended to run `clearDataStore` and `provisionDataStore` in conjunction, using the **Reset Data Domain** 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 ETL.

Removing all records from the data domain

Use the `clearDataStore` operation in the Data Ingest Web Service to remove all data records and also the schema records that define the data domain 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 domain with a new set of records).

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

```
<ingest:clearDataStore/>
```

Note: If you submit this request after a Transaction Web Service request that starts an outer transaction, the `clearDataStore` 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 `clearDataStore` 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/3/0">
    <soapenv:Header/>
        <soapenv:Body>
           <ingest:clearDataStore>
              <ingest:OuterTransactionId>txId</ingest:OuterTransactionId>
           </ingest:clearDataStore>
         </soapenv:Body>
   </soapenv:Envelope>
   ```

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

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

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

Note that if you specify an outer transaction ID that does not match the ID of the currently running transaction, the `clearDataStore` 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 domain, you want to provision it again with the default configuration settings. To provision the data domain, run `provisionDataStore`.

Note: Clearing the data domain without provisioning it again leaves the data domain in an invalid state, preventing all further operations, including importing the configuration.

Provisioning the data domain

To provision the data domain, use the `provisionDataStore` 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 `clearDataStore`. It is also assumed that you have previously exported your configuration defined in the schema records, and will import it after you run the `provisionDataStore` operation.

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

```
<ingest:provisionDataStore/>
```

Note: If you submit the provision request after a Transaction Web Service request that starts 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 domain:

1. Create a `provisionDataStore` 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/3/0">
    <soapenv:Header/>
        <soapenv:Body>
           <ingest:provisionDataStore>
              <ingest:OuterTransactionId>txId</ingest:OuterTransactionId>
           </ingest:provisionDataStore>
         </soapenv:Body>
      </soapenv:Envelope>
   ```

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

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

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

After you have run the `provisionDataStore` operation, you can import your configuration and run the Endeca Server `update-spelling-dictionaries` command to update the spelling dictionary.

If you use **Reset Data Domain** in Integrator for running `clearDataStore` and `provisionDataStore` operations, then this connector also updates the spelling dictionary.

Updating spelling dictionaries for the data domain

Use the `updateSpellingDictionaries` operation of the Data Ingest Web Service to update the spelling dictionaries in the data domain. Run this operation after you have added data records to the data domain, to enable spelling correction in the Dgraph.

This operation rebuilds the spelling dictionaries for spelling correction from the data corpus while the Endeca Server continues to run. The Endeca Server can continue processing end-user queries and updates to the data domain's index, without stopping and restarting the Dgraph nodes.

Run this operation after you have added data records to the data domain, to enable spelling correction in the Dgraph processes for this data domain.

Additionally, during the data ingest process, you can run the request with the `updateSpellingDictionaries` operation periodically to update the spelling dictionary the data domain for one of the Endeca Server features known as Automatic Spelling Correction and DYM (Did You Mean?).

In a cluster of Dgraph nodes (a data domain running on multiple hosting Endeca Server machines), this request is routed to the leader node, because it is an updating operation.

The request performs the following actions:

- Crawls the text search portion of the index for the data domain for all terms that meet the constraint settings.

  The constraint settings include minimum word occurrences and maximum and minimum number of characters, for records and attribute values. The Dgraph nodes in the data domain use these constraints to update the spelling dictionaries. You can change them in the Global Configuration Record.
- Updates the spelling dictionaries stored in the Dgraph for processing of all queries arriving after this update to the index files. The Dgraph uses these updated dictionaries when processing all future queries. For detailed information on spelling dictionaries, see the *Oracle Endeca Server Developer's Guide*.

The Dgraph process applies the updated settings while continuing to run queries and without needing to restart.

To update the spelling dictionaries for the data domain:

1. Access the Data Ingest Web Service for the data domain:

   ```
   http://localhost:<port>/endeca-server/ws/ingest/<dataDomain>?wsdl
   ```

   where the `localhost` and `port` are the host and port of the running Oracle Endeca Server, `endeca-server` is the default context root for the Endeca Server Java application running in the WebLogic Server, and *dataDomain* is the name of the Endeca data domain.
2. Use the `updateSpellingDictionaries` operation in the request, as in this example:

   ```
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:ingest="http://www.endeca.com/MDEX/ingest/3/0">
    <soapenv:Header/>
        <soapenv:Body>
           <ingest:updateSpellingDictionaries>
              <ingest:OuterTransactionId>txId</ingest:OuterTransactionId>
           </ingest:updateSpellingDictionaries>
         </soapenv:Body>
   </soapenv:Envelope>
   ```

   where `txId` is the ID of an outer transaction (if one is running).

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

If the data domain exists and is enabled, the operation completes successfully, returning an empty web service success response (`updateSpellingDictionariesResponse`). This means that the spelling dictionaries have been updated successfully.

Chapter 7

Bulk Load API

This section 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 domain. 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 domain 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 domain, the new standard attribute is automatically created with system default values for the PDR. For these default values, see *Default values for new attributes on page 22*.
- There is a limit of 128MB on the maximum size of a source record. An attempt to ingest a source record larger than 128MB will fail and an error will be returned (with the record spec of the rejected record), but the bulk load ingest will continue after that rejected record.

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.

Implementing Logging

The `BulkIngester` class is the primary entry point for the client-side Bulk Load Interface for loading data into an Endeca data domain. `BulkIngester` uses the `java.util.logging` package to output some logging for debugging purposes. (Clients see the debugging messages through Callback classes and Exception messages.)

To implement logging for your program, you can use other logging libraries, such as Apache Commons logging, Simple Logging Facade for Java (SLF4J), and so on. For details, see the documentation that is provided with those logging components.

Implementing SSL connections

The `BulkIngester` constructor's `socketFactory` parameter takes in a socket factory that is used to create sockets for the bulk ingest operations. Clients can therefore specify their own pre-configured SocketFactory. This is especially useful in the case of SSL, so that you can use this parameter to specify an SSLSocketFactory for SSL connections.

The sample program imports two Java socket classes and then sets the `socketFactory` parameter as follows:

```
import javax.net.SocketFactory;
import javax.net.ssl.SSLSocketFactory;
...
public BulkLoad(String host, int port, boolean ssl) throws IOException {
        _host = host;
        _port = port;
        BulkLoadCallback blcb = new BulkLoadCallback();

        SocketFactory sf = ssl ? SSLSocketFactory.getDefault() : SocketFactory.getDefault();
...
```

If the value of the incoming `ssl` Boolean parameter is `true`, then `sf` is set to a copy of the environment's default SSL socket factory, otherwise it is set to a copy of the default non-SSL socket factory.

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 allows you to 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 spelling dictionaries are 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 dictionaries. For information, see *Updating spelling dictionaries for the data domain on page 60*.

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.

The source file is `BulkLoad.java` and is located in the `$ENDECA_HOME/aps/examples` directory.

```
package com.oracle.endeca.server.examples.ops;

import java.io.IOException;
import java.util.concurrent.TimeUnit;

import javax.net.SocketFactory;
import javax.net.ssl.SSLSocketFactory;

import com.endeca.BulkLoad.AbortCallback;
import com.endeca.BulkLoad.BulkIngester;
import com.endeca.BulkLoad.ErrorCallback;
import com.endeca.BulkLoad.FinishedCallback;
import com.endeca.BulkLoad.StatusCallback;
import com.endeca.BulkLoad.Msg.Data;
import com.endeca.BulkLoad.Msg.Data.Record;

/**
 * This class provides an example of working with the bulk loading interface.
 */
public class BulkLoad {

    /**
     * This callback is provided to the bulk loader for providing feedback to the client process.
     */
    public static class BulkLoadCallback implements AbortCallback, ErrorCallback, FinishedCallback,
StatusCallback
    {
        @Override
        public void handleAbort(String reason) {
            System.out.println("Abort: " + reason);
        }

        @Override
        public void handleError(String reason, Record reject) {
            System.out.println("Error: " + reason);
        }

        @Override
        public void handleStatus(long recordsAdded, long recordsQueued, long recordsRejected, String
state) {
            System.out.println("Status: " + recordsAdded);
        }

        @Override
        public void handleFinish() {
            System.out.println("Finished!");
        }
    }

    /**
     * Set up a basic bulk ingest connection.
     */
```

```
    public BulkLoad(String host, int port, boolean ssl) throws IOException {
        _host = host;
        _port = port;
        BulkLoadCallback blcb = new BulkLoadCallback();

        SocketFactory sf = ssl ? SSLSocketFactory.getDefault() : SocketFactory.getDefault();
        boolean doFinalMerge = false;
        boolean doUpdateSpellingDictionary = false;
        long socketTimeout = TimeUnit.SECONDS.toMillis(90);
        String transactionId = null;

        // don't do a final merge after finishing ingest or update the spelling dictionary
; timeout after 90s
        _bi
= new BulkIngester(_host, _port, sf, transactionId, doFinalMerge, doUpdateSpellingDictionary,
socketTimeout, blcb, blcb, blcb, blcb);
        _bi.begin();
    }

    /**
     * Add a new record with a handful of given assignments.
     */
    public void addRecord(int val, String words, boolean square, boolean prime) throws IOException {
        Data.Record.Builder recordBuilder = Data.Record.newBuilder();
        Data.Assignment.Builder assignmentBuilder = Data.Assignment.newBuilder();

        assignmentBuilder.clear();

assignmentBuilder.setName("Value").setDataType(Data.Assignment.DataType.INT).setInt32Value(val);
        recordBuilder.setSpec(assignmentBuilder.build());

        assignmentBuilder.clear();

assignmentBuilder.setName("Words").setDataType(Data.Assignment.DataType.STRING).setStringValue(words)
;
        recordBuilder.addAssignments(assignmentBuilder.build());

        assignmentBuilder.clear();

assignmentBuilder.setName("Square").setDataType(Data.Assignment.DataType.BOOLEAN).setBoolValue(square
);
        recordBuilder.addAssignments(assignmentBuilder.build());

        assignmentBuilder.clear();

assignmentBuilder.setName("Prime").setDataType(Data.Assignment.DataType.BOOLEAN).setBoolValue(prime);
        recordBuilder.addAssignments(assignmentBuilder.build());

        Data.Record r = recordBuilder.build();
        _bi.sendRecord(r);
    }

    public void done() throws IOException, InterruptedException {
        _bi.endIngest();
    }

    private String _host;
    private int _port;
    private BulkIngester _bi;

}
```

Records, assignments, and data types

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

Records and assignments

Client programs using the Bulk Load API to load data into an Endeca data domain 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:

```
Data.Record.Builder recordBuilder = Data.Record.newBuilder();
Data.Assignment.Builder assignmentBuilder = Data.Assignment.newBuilder();

assignmentBuilder.clear();
assignmentBuilder.setName("Value").setDataType(Data.Assignment.DataType.INT).setInt32Value(val);
recordBuilder.setSpec(assignmentBuilder.build());
```

In this example, the name of the Spec is "Value" and it has an integer value ("val") that is input to the program.

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

```
assignmentBuilder.clear();
assignmentBuilder.setName("Words").setDataType(Data.Assignment.DataType.STRING).setStringValue(words)
;
recordBuilder.addAssignments(assignmentBuilder.build());
```

The sample program creates one record with three non-Spec attributes (the String "Words" and the two Boolean "Square" and "Prime").

Builders

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

- The `Data.Record.Builder` has `setSpec()` methods for the Spec and `setAssignments()` methods for the `Assignment` objects.
- The `Data.Assignment.Builder` has the `setName()` method for its key and various setters for its 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 `Data.Assignment.Builder` class's setter methods as follows:

Enum DataType	Data.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 Dgraph 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:
assignmentBuilder.setBoolValue(false);
assignmentBuilder.setDataType(Assignment.DataType.BOOLEAN);

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

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

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

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

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

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

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

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

Sending records to the data domain

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

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

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 Dgraph.
5. When finished sending records, call `endIngest()` to terminate the response thread and close the socket.

Note that periodically you may want to call the `requestStatusUpdate()` method to poll the Dgraph on its current status.

Defining callback interfaces

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

- `ErrorCallback` handles error conditions. The `handleError()` method is called when errors occur during the ingest operation.
- `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. An example of defining this callback is:

```
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 eleven parameters, in this order:

Parameter	Meaning
`host`	The name (a `String`) of the machine on which the Dgraph is running. The machine name can be obtained by using the `allocateBulkLoadPort` operation of the Manage Web Service.
`port`	The bulk load port (an `int`) of the Dgraph. The port can be obtained by using the `allocateBulkLoadPort` operation of the Manage Web Service.
`socketFactory`	A SocketFactory to create sockets for the bulk ingest operations. The sample program uses `SocketFactory.getDefault()` for non-SSL connections or `SSLSocketFactory.getDefault()` for SSL connections. Both methods are from the imported `javax.net.SocketFactory` and `javax.net.ssl.SSLSocketFactory` classes.
`transactionId`	The ID (a `String`) of an outer transaction (if one has been started by the Transaction Web Service). Use `null` if an outer transaction is not being used.
`doFinalMerge`	A `boolean` that specifies whether a merge is forced immediately after ingest. Setting it to `true` degrades ingest performance. Therefore, if you are performing multiple bulk load ingests, you should set this to `false` for all but the last ingest job.
`doUpdateDictionary`	A `boolean` that specifies whether the aspell dictionary is updated immediately after ingest. Setting it to `true` degrades ingest performance. Therefore, if you are performing multiple bulk load ingests, you should set this to `false` for all but the last ingest job.
`socketTimeoutMillis`	The timeout in milliseconds (an `int`) for a connection to the Dgraph.
`errorCallback`	The `ErrorCallback` object.
`finishedCallback`	The `FinishedCallback` object.
`abortCallback`	The `AbortCallback` object.
`statusCallback`	The `StatusCallback` object.

The sample program constructs the `BulkIngester` parameters follows:

```
_host = host;
_port = port;
BulkLoadCallback blcb = new BulkLoadCallback();

SocketFactory sf = ssl ? SocketFactory.getDefault() : SSLSocketFactory.getDefault();
boolean doFinalMerge = false;
boolean doUpdateSpellingDictionary = false;
long socketTimeout = TimeUnit.SECONDS.toMillis(90);
String transactionId = null;
```

The call to set up a connection to the Dgraph is:

```
_bi = new BulkIngester(_host,
                       _port,
                        sf,
                        transactionId,
                        doFinalMerge,
                        doUpdateSpellingDictionary,
                        socketTimeout,
                        blcb,
                        blcb,
                        blcb,
                        blcb);
```

Beginning and ending the ingest

After the client program has made a connection to the Endeca data domain, 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 domain.
3. The `endIngest()` method terminates the ingest process.

The sample program, which ingests only one record, is coded as follows:

```
public BulkLoad(String host, int port, boolean ssl) throws IOException {
  ...
  _bi
= new BulkIngester(_host, _port, sf, transactionId, doFinalMerge, doUpdateSpellingDictionary,
socketTimeout, blcb, blcb, blcb, blcb);
  _bi.begin();
}

public void addRecord(int val, String words, boolean square, boolean prime) throws IOException {
        Data.Record.Builder recordBuilder = Data.Record.newBuilder();
        Data.Assignment.Builder assignmentBuilder = Data.Assignment.newBuilder();
  ...
  Data.Record r = recordBuilder.build();
  _bi.sendRecord(r);
}

public void done() throws IOException, InterruptedException {
  _bi.endIngest();
}
```

Although not used in the sample program, the `requestStatusUpdate()` method can be used to retrieve the status of the ingest operation.

Appendix A

The Deprecated ingestManagedAttributeValues Operation

This section describes how to add managed attribute values to the Endeca data domain using the deprecated operation `ingestManagedAttributeValues`.

ingestManagedAttributeValues operation

The `ingestManagedAttributeValues` operation for adding managed attribute values has been deprecated as of Endeca Server version 7.6.0.

Attention! The `ingestManagedAttributeValues` operation is supported in this release for backward compatibility (and, for this reason, is described in this section). However, Oracle recommends that you migrate away from using this operation. Instead of using this operation, use the `putManagedAttributeValues` operation of the Configuration Web Service. This is the preferred way to add managed attribute values.

Within the `ingestManagedAttributeValues` structure, the `ingestManagedAttributeValue` element specifies the managed attribute to which each managed value belongs. If the managed attribute does not exist in the data domain, the service automatically creates the managed attribute.

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

ingestManagedAttributeValues request syntax

An `ingestManagedAttributesValues` operation request uses this format:

```
<ingestManagedAttributeValues>
   <managedAttributeValue displayName="dName" spec="maValueSpec"
         parentSpec="pSpec" managedAttribute="maName">
      <synonym>synName</synonym>
      <properties>
         <propName type="mdex:string">propValue</propName>
      </properties>
   </managedAttributeValue>
</ingestManagedAttributeValues>
```

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

Element/Attribute	Purpose
`managedAttribute`	The name of the managed attribute to which the managed value belongs. The name must use the NCName format.
`displayName`	A name for the managed value that can be used for user interface display purposes. The name does not have to use the NCName format.
`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.

Example of adding managed attribute values

Assuming that the `Component` managed attribute exists, the following example adds three managed values (Derailleur, Tire, and Michelin) to it:

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0">
   <soapenv:Header/>
   <soapenv:Body>
     <ns:ingestManagedAttributeValues>
       <ns:managedAttributeValue displayName="derailleur gears" spec="Derailleur" parentSpec="/"
           managedAttribute="Component">
         <ns:synonym>Chain</ns:synonym>
         <ns:synonym>Gear</ns:synonym>
       </ns:managedAttributeValue>
       <ns:managedAttributeValue displayName="bicycle tires" spec="Tire" parentSpec="/"
           managedAttribute="Component">
         <ns:properties>
           <myStrProp type="mdex:string">source:CAS</myStrProp>
         </ns:properties>
       </ns:managedAttributeValue>
       <ns:managedAttributeValue displayName="Michelin brand" spec="Michelin"
           parentSpec="Tire" managedAttribute="Component">
       </ns:managedAttributeValue>
     </ns:ingestManagedAttributeValues>
   </soapenv:Body>
</soapenv:Envelope>
```

In the example, the `Derailleur` and `Tire` managed values are at the root of the `Component` 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.

Note that if the `Component` managed attribute did not previously exist, the request would create it with default settings for its PDR and DDR.

ingestManagedAttributeValues responses

The `ingestManagedAttributeValues` operation has success and failure responses.

Success response

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

```
<ingest:ingestManagedAttributeValuesResponse
    xmlns:ingest="http://www.endeca.com/MDEX/ingest/3/0">
   <ingest:numManagedAttributesCreated>1</ingest:numManagedAttributesCreated>
   <ingest:numManagedAttributeValuesCreated>3</ingest:numManagedAttributeValuesCreated>
</ingest:ingestManagedAttributeValuesResponse>
```

In the sample response, the `numManagedAttributesCreated` element shows that one managed attribute was created, while the `numManagedAttributeValuesCreated` 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:

```
<ns:ingestManagedAttributeValues>
 <ns:managedAttributeValue
  displayName="bicycle saddles" spec="Saddle" parentSpec="Parts" managedAttribute="Component">
 </ns:managedAttributeValue>
</ns:ingestManagedAttributeValues>
```

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

```
<ingest:errorDetail>
Error applying updates: Managed attribute value put refers to
parent spec "Parts", which does not exist in managed attribute "Component"
</ingest:errorDetail>
```

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

Index