Endeca MDEX Engine

Web Services and XQuery Developer's Guide

Version 6.2.2 • March 2012

Contents

Copyright and disclaimer

Preface

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

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

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

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

About this guide

This guide contains general information about the Web services and XQuery for Endeca implementation, as well as procedures for developers to get started working with the software.

Who should use this guide

This book is intended for Endeca application developers interested in adding Web services and XQuery functionality to their Endeca applications.

It assumes knowledge of Web services, the XQuery programming language, and Endeca application development.

Conventions used in this guide

This guide uses the following typographical conventions:

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

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

Contacting Oracle Endeca 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 overview information about Web services and XQuery for Endeca, and explains the scope of this release.

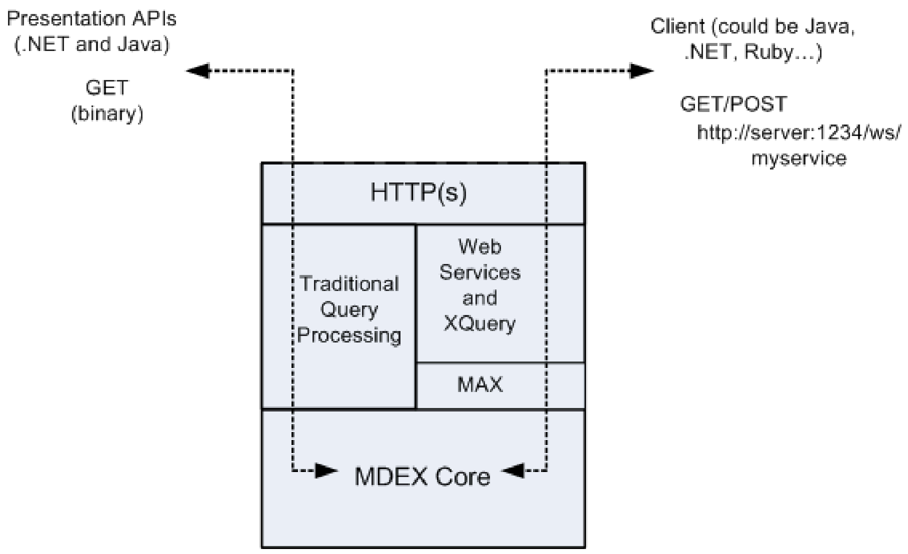
Overview of Web services and XQuery for Endeca

Web services and XQuery for Endeca provides Endeca application developers with a flexible, extensible, and standards-compliant query processing solution.

You can use Web services and XQuery for Endeca alongside the Endeca Presentation API to extend the functionality of your Endeca application. You can also write an entire application without using the binary Presentation APIs at all, as you would want to do if your site were built in a language environment other than Java or .NET.

The XQuery for Endeca query layer allows an unprecedented level of customization of the core MDEX Engine. By embedding a complete programmatic environment into the core engine, Endeca has made it possible for application developers to customize the behavior of and interface to the MDEX Engine, and share their solutions across platforms and application development environments. By encoding certain MDEX-specific logic into custom XQuery modules, applications can be made more flexible and easier to maintain and support.

As the diagram below illustrates, the XQuery for Endeca solution consists of an HTTP server and an XQuery evaluator embedded in the MDEX Engine framework. The XQuery processing module can receive a Web service request (such as a SOAP request) over HTTP and process it using XQuery. The XQuery evaluator interacts with the data store through a set of Endeca-specific functions that have access to the MDEX data. That is, the MDEX API through XQuery (or MAX) exposes the query API to the XQuery evaluator. The data returned is represented as XML and can therefore be manipulated quite readily in the XQuery execution engine.


Presentation APIs (.NET and Java)
GET (binary)
Client (could be Java, .NET, Ruby...)
GET/POST http://server:1234/ws/myservice
HTTP(s)
Traditional Query Processing
Web Services and XQuery
MAX
MDEX Core

When an HTTP request is received, the URL path is used to select an XQuery main module for evaluation. XQuery functions provide access to features of the HTTP request, including the POST body, and to navigation and other features of the MDEX Engine. The result of the XQuery evaluation is then returned as the HTTP response.

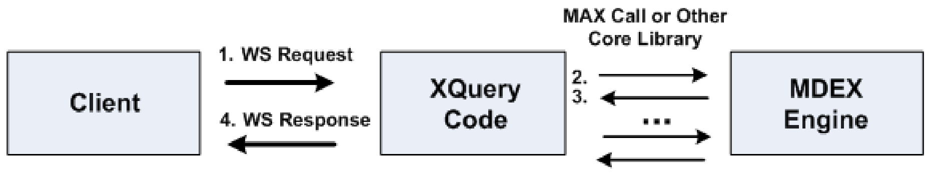
How the pieces of Web services and XQuery for Endeca work together

For Endeca application developers, XQuery is an additional way to write queries to the MDEX Engine and to extend MDEX Engine functionality.

The XQuery developer can write code in any XQuery development editor (including a regular text editor). The developer puts his or her code into an XQuery *main module*, which is analogous to a stored procedure in the RDBMS world. Each main module located in the directory specified in the `--xquery_path` flag is automatically enabled as a Web service. For example, the XQuery module `myquery.xq` is exposed as the URL `http://<mdex_host>:<port>/ws/myquery`.

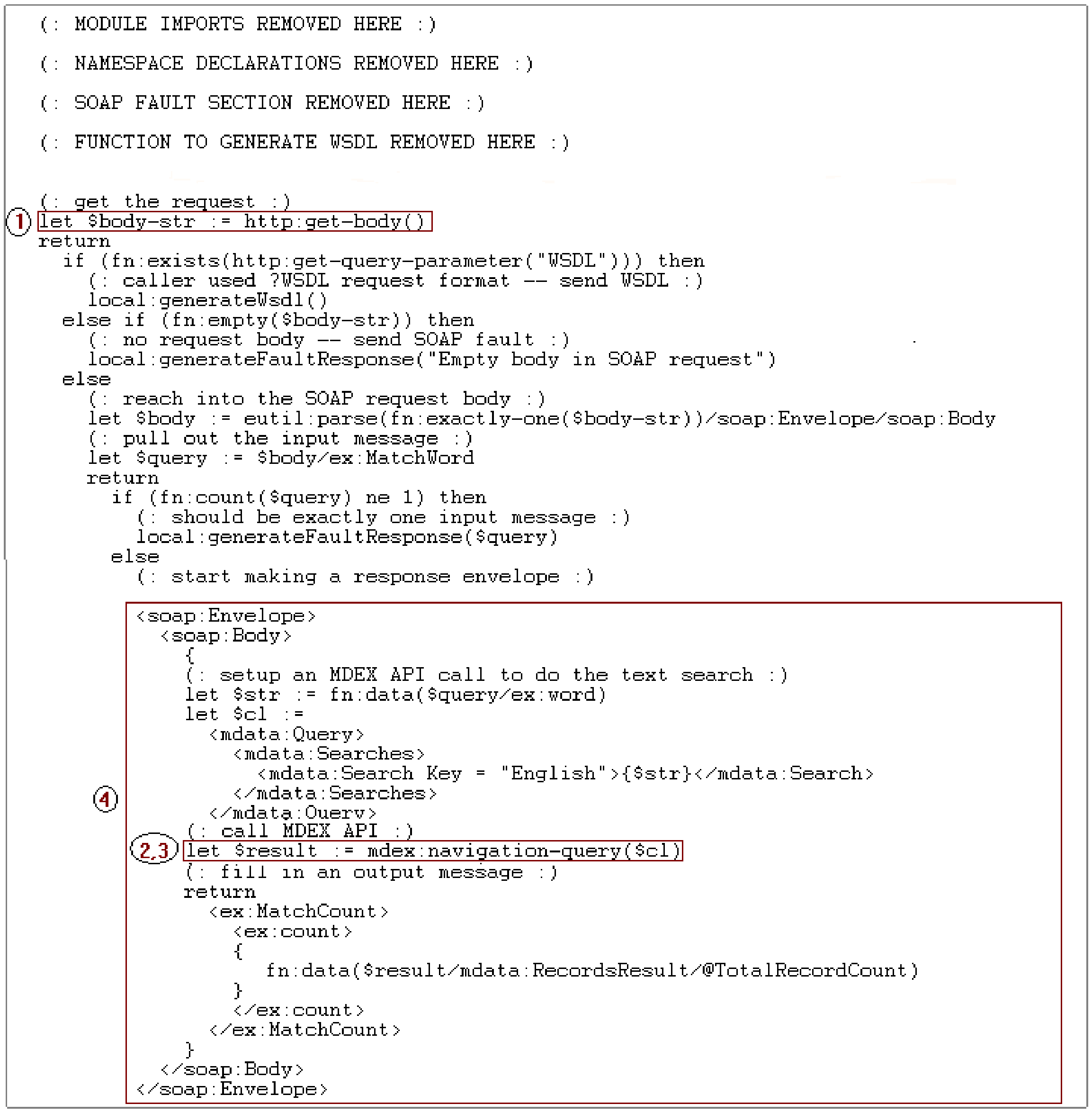
The MDEX Engine handles the HTTP part of the Web service, leaving the XQuery developer free to implement the request and response structure via any standard protocol such as SOAP, JSON, or XML-RPC. The developer accesses the HTTP request (the URL, the POST body, and so on) by calling the XQuery HTTP library. The Web service response is the XML result of the main module evaluation. Typically, the developer makes one or more calls to the MDEX library per Web services invocation, but this is not a requirement.

The diagram below illustrates query flow in Web services mode:



In this scenario, the client first sends the request via a Web services protocol like SOAP to the MDEX Engine. The request is dispatched to the body of an XQuery main module, which calls the MDEX API through XQuery (or MAX) some number of times to help it service the request. The final results of the main module are packaged as a Web service response (such as a SOAP envelope) and returned to the issuing client. The XQuery main module in the MDEX Engine can pre- and post-process application requests, and a single Web service call can send multiple requests to the MDEX Engine without requiring multiple round trips from the client to the MDEX Engine.

The following code extract walks through the same scenario, with highlights around the portions of the code that relate to each step in the diagram. In the interest of brevity, certain declarations have been omitted here. For the full version of this XQuery main module, see the section "Getting Started with Web Services and XQuery for Endeca."



```
(: MODULE IMPORTS REMOVED HERE :)

(: NAMESPACE DECLARATIONS REMOVED HERE :)

(: SOAP FAULT SECTION REMOVED HERE :)

(: FUNCTION TO GENERATE WSDL REMOVED HERE :)


(: get the request :)
let $body-str := http:get-body()
return
  if (fn:exists(http:get-query-parameter("WSDL"))) then
    (: caller used ?WSDL request format -- send WSDL :)
    local:generateWsdl()
  else if (fn:empty($body-str)) then
    (: no request body -- send SOAP fault :)
    local:generateFaultResponse("Empty body in SOAP request")
  else
    (: reach into the SOAP request body :)
    let $body := eutil:parse(fn:exactly-one($body-str))/soap:Envelope/soap:Body
    (: pull out the input message :)
    let $query := $body/ex:MatchWord
    return
      if (fn:count($query) ne 1) then
        (: should be exactly one input message :)
        local:generateFaultResponse($query)
      else
        (: start making a response envelope :)

          <soap:Envelope>
            <soap:Body>
              {
              (: setup an MDEX API call to do the text search :)
              let $str := fn:data($query/ex:word)
              let $cl :=
                <mdata:Query>
                  <mdata:Searches>
                    <mdata:Search Key = "English">{$str}</mdata:Search>
                  </mdata:Searches>
                </mdata:Query>
              (: call MDEX API :)
              let $result := mdex:navigation-query($cl)
              (: fill in an output message :)
              return
                <ex:MatchCount>
                  <ex:count>
                  {
                     fn:data($result/mdata:RecordsResult/@TotalRecordCount)
                  }
                  </ex:count>
                </ex:MatchCount>
              }
            </soap:Body>
          </soap:Envelope>
```

Using XQuery with the MDEX Engine

Starting with version 6.1, the MDEX Engine provides two ways for application developers to use XQuery: custom Web services, and the MDEX Web service.

Creating custom Web services with the Advanced Query Module

XQuery developers can write their own XQuery main and library modules that extend MDEX Engine functionality.

Developing a custom Web service allows you to write arbitrarily complex XQuery logic organized into any number of library modules in addition to the main module. The modules are loaded and compiled on Dgraph startup and ready to process Web service requests efficiently.

Typically, complex, performance-critical XQuery logic for an application in production will be implemented as a custom Web service, as this makes it easier to manage complexity and obtains the best performance. It also allows you to use the same XQuery logic from multiple application-tier applications.

Note: The creation of XQuery modules, aside from those provided with the MDEX distribution, requires the purchase of the Advanced Query Module. Contact your Endeca representative for details.

Using the MDEX Web service to access XQuery functionality

The MDEX Web service, which is included as part of this release, allows you to use XQuery with the MDEX Engine without writing custom Web services.

The MDEX Web service is not meant as a Web service replacement to the Presentation API, but rather as a data service similar to something that a relational database management system might provide. In an RDBMS scenario, SQL input transported via ODBC or JDBC returns tabular results. Similarly, in an MDEX scenario, XQuery input transported via the MDEX Web service returns XML results.

For more details and examples of getting started with the MDEX Web service, see the section "Using the MDEX Web service."

Note: In previous releases of version 6.0, the MDEX Web service was known as the Query Web service.

Why XQuery?

XQuery is a programming language designed for processing XML and other semi-structured data. Endeca has added an XQuery evaluator to the MDEX Engine for a number of reasons.

Principally, XQuery is a language designed to make it easy to manipulate XML fragments and other structured and semi-structured data that can be represented as XML. It is more convenient to manipulate XML (and therefore record results, dimension trees, and whatever else the MDEX Engine might return) in XQuery than it would be to do so in SQL, Java, .NET, or Perl.

XQuery is also a W3C standard, with rigorously defined and tested semantics. This allows MDEX developers to benefit from the body of practice surrounding XQuery. The XQuery language is extensible via external functions, pragmas (implementation-specific language extension facilities), and library modules.

In addition to the ease with which XQuery lets you manipulate XML, the benefits of XQuery include the following:

- It is compatible with Web services.
- It allows business logic to be implemented in the MDEX Engine, without requiring application modification.

- It supports code reuse between environments such as Java and .NET.

Exposing your custom XQuery main modules as Web services may also provide performance benefits. It can save the cost of network transfer in application instances where multiple round trips to the MDEX Engine are required. The data is handed to the XQuery runtime by the MDEX Engine by way of an in-memory data structure, which is faster than marshaling results, sending them over the wire, unmarshaling the results, and manipulating them in the client application. There may also be a caching benefit in cases where a load balancer directs queries at random to numerous Dgraphs, because all of the requests to the MDEX Engine from a single Web service call will execute in the same Dgraph.

Impact on Endeca application development

XQuery for Endeca provides application developers with a portable, standards-based query processing solution. In addition, XQuery features are cross-platform and reusable, so application developers can reuse their own work and share it with other developers.

With XQuery for Endeca, some application logic can be implemented in XQuery and executed within the MDEX Engine. That means that application developers do not need to implement that functionality in their application tier. (Exactly when it makes sense to move application logic into XQuery will depend on the case. See the section below for advice in this regard.)

Application-tier developers do not interact with XQuery at the API level. Instead, they use a Web service that accesses main modules written in XQuery in the MDEX Engine. This changes the way you build Endeca applications, because instead of being limited to the MDEX Presentation APIs, you define the API that makes sense for your application as a set of Web services.

Moving application logic into XQuery

The following guidance should help you decide what logic to move into XQuery and what to keep in the application tier.

The following logic fits well within XQuery:

- Endeca-specific functionality, such as cascading search.
- Application logic used to compute a query from other information (such as deciding whether to use a particular record filter, search interface, or type of sort).
- Query output that needs to be filtered down to the minimum amount needed for page rendering in order to minimize the amount of network traffic. (Presentation API queries tend to have large responses.)
- Page views that need more than one query to populate them. If possible, those multiple queries should be moved to the MDEX Engine behind a single Web service request.
- Application logic used to customize the behavior of the MDEX Engine (such as instructing the MDEX Engine to automatically run a more general query if there are few results, or to have breadcrumb removal implicitly select the parent).
- Application logic used to customize results (such as deciding whether to display or hide a dimension, or render or hide a property).
- Code to reformat queries or results to better fit the needs of a particular application.
- Record filters used for security purposes.

The following logic should be kept in the application tier:

- User session management (such as authentication, bookmark management, and shopping cart management).
- UI rendering logic (such as determining which widgets to re-render).

- State information.
- Authorization logic.
- Integration with third-party systems (such as pulling supplemental data from a database).

Obtaining more information about XQuery

This document assumes that you already have a good working knowledge of XQuery. If you do not, you may find the resources listed here helpful.

The W3C landing page for XQuery *http://www.w3.org/XML/Query/* contains overview information as well as a number of useful links to discussions and further reading. For information about general XQuery syntax, consult the XQuery specification here: *http://www.w3.org/TR/xquery/*. For information specifically about XQuery built-in XPath functions, consult the XQuery 1.0 and XPath 2.0 Functions and Operations specification here: *http://www.w3.org/TR/xpath_functions*. For information about XQuery update, consult the XQuery Update Facility 1.0 spec here: *http://www.w3.org/TR/xquery-update-10/*.

Books on XQuery

The following books provide useful information about the XQuery language:

Brundage, Michael. *XQuery: The XML Query Language.* Boston: Addison-Wesley, 2004.

Katz, Howard, ed. *XQuery from the Experts: A Guide to the W3C XML Query Language.* Boston: Addison-Wesley, 2004.

Walmsley, Priscilla: *XQuery.* Sebastopol, CA: O'Reilly Media, 2007.

Note: Some of these books were published before the XQuery specification, and so do not exactly reflect the final XQuery language. The technical reports for XQuery published by the W3C should be taken as the authoritative specifications.

Obtaining more information about Web services

This document assumes that you already have a good working knowledge of Web services. If you do not, you may find the resources listed here helpful.

- *Skillbuilders' Introduction to Web Services*
- *w3schools' Web Services Tutorial*
- *Microsoft's ASP .NET Quick Start Tutorial*
- *Skillbuilders' Introduction to SOAP, WSDL, and UDDI*
- *w3schools' SOAP Tutorial*
- *w3schools' WSDL Tutorial*
- *Axis2 Java SOAP Library*
- *Axis2 Code Generator Tool*

About this release of Web services and XQuery for Endeca

This section provides details about the release of Web services and XQuery for Endeca.

Changes since the 6.1.2 release

The following features have been added or modified since the version 6.1.2 release of Web services and XQuery for Endeca.

For details on any of these features, see elsewhere in this guide. For complete information about problems fixed in this release, as well as known issues, see the MDEX release notes.

Addition of language IDs to the MAX API

A `LanguageId` element has been added to the navigation, dimension search, and compound dimension search queries to allow per-query specification of the language to parse search terms in.

XQuery try/catch expression change

If exceptions are raised in a try block, any updates appended within that try block are removed from the pending update list. This rollback is applied even if the exception is not caught.

MDEX Web service change

When you request the WSDL for the MDEX Web service, `mdex?wsdl` now pulls the Host header directly from the HTTP request and uses the host and port information that it finds there.

Early Access features in this release

This release of Web services and XQuery for Endeca includes some features that are in Early Access state. The interface of these features is likely to change, and they are not supported for production use at this time. These features are documented in chapter 5 of this guide.

Data Update API for updating Web services

A Data Update API for updating Web services has been added. In this Early Access release, it is possible to add, modify, or delete a record or add a dimension value.

Statistics related to updating Web services have been added to the MDEX Engine Statistics page.

Persistent document storage with fn:put()

Persistent storage of XML documents has been implemented as an Early Access feature, using the updating XQuery function `fn:put()`.

Introduction of the dimension value spec

The dimension value spec an Early Access feature in this release. When used, the dimension value spec serves as a primary key for dimension values within a dimension.

Three utility functions have been added to allow for conversion between dimension value IDs and dimension value specs.

A Dgidx flag called `--autogenerate-dval-specs` enables the auto-generation of dimension value specs in cases where it is not possible to assign dimension value specs directly.

What's not in this release

This section outlines features that are not part of this release of Web services and XQuery for Endeca.

Dgraph features not supported in the MDEX API through XQuery:

- The Agraph (Aggregated MDEX Engine).
- Clustering queries.

XQuery features not supported:

- Schema imports.
- Schema validation.
- Certain XQuery functions have not been implemented yet. A list can be found in the section "About functions."
- `fn:collection()`: the only collection available at this time is the `mdex://documents` collection, which is an Early Access feature in this release.
- Ability to bind to a context item in a main module, except in the case of the `eutil:eval` function.
- Ability to bind to external variables in a main module.

Performance expectations for this release

The performance of XQuery for Endeca has improved significantly in this release.

Nonetheless, when using Web services and XQuery for Endeca, keep in mind the following performance expectations. The time XQuery takes to process a request is directly proportional to the amount of data it needs to process and return. The Endeca XQuery evaluator is able to process data of the size of a normal result set (containing records, refinements, and supplements) in a reasonable amount of time (a few tens of operations per second on contemporary server hardware). Reducing the amount of data processed and returned by XQuery will increase throughput; correspondingly, increasing the amount of data processed and returned will reduce throughput.

Details on XQuery for Endeca performance limitations and tips can be found in the "XQuery Components and Features" section of this guide.

About the examples used in this guide

This topic discusses the conventions used in examples in this guide and points you to additional examples.

In addition to the special character used for hard line breaks in code examples, mentioned in the Preface, placeholders in example code that need to be replaced by an actual name appear in angle brackets. For example, `<prefix>:<function-name>` might be replaced by `fn:string`. The topic "EBNF for the URL format" contains more information about formatting.

The function names in the examples used in this guide are sometimes qualified with the namespace prefix `fn` and sometimes not. This makes no difference to the meaning or behavior of the function. For example, the function `fn:concat` is the same as the function `concat`.

About connecting Web browsers to your MDEX Engine

For security reasons, you should never allow user Web browsers to connect directly to your MDEX Engine (although an administrator may choose to connect directly to the MDEX Engine using proper precautions).

Non-administrator browsers should always connect to your application through an application server.

Chapter 2

Getting Started with Web Services and XQuery for Endeca

This section provides installation, setup, and workflow information to get you started on your first Web services and XQuery for Endeca project.

Dgraph flags that control XQuery use

The following Dgraph flags control various aspects of XQuery for Endeca.

Flag	Description
`--xquery_fndoc <option>`	Optional. Specifies the handling of the `fn:doc()` function within XQuery. The following three values are supported: • *none* causes all calls to `fn:doc()` to fail. • *sandbox* allows `fn:doc()`, but interprets its argument as a relative path within the `xml` subdirectory of the XQuery service directory. • *open* allows `fn:doc()` and interprets its argument as a URL. At this time, `fn:doc()` with a setting of *open* is not supported for use in production. If this flag is not specified, it defaults to `none`.
`--xquery_path <directory>`	Optional. The directory in which XQuery Web service resources are located. XQuery main modules are loaded from this directory. (Associated library modules are loaded from the `lib` subdirectory of the directory specified here.) If `--xquery_path` is not specified, a user XQuery path is not used, and user-supplied XQuery modules are not loaded. **Note:** You can specify the path multiple times pointing to different folders, and the paths will be searched in the order they are provided.

Setting up your XQuery for Endeca directory

When you start the MDEX Engine, built-in XQuery library modules are automatically loaded.

Application-specific XQuery modules are loaded from the directory you specify in `--xquery_path`. The MDEX Engine creates a Web service for each file in the `--xquery_path` directory with an `.xq` extension that contains an XQuery main module.

Application-specific library modules are loaded from files with an `.xq` extension that are located in the `lib` subdirectory of the directory specified in `--xquery_path`. If `--xquery_fndoc` is set to `sandbox`, XML files can be loaded from an `xml` subdirectory of the directory specified in `--xquery_path`, and can then be accessed using the `fn:doc()` built-in function.

Note: To suppress the automatic loading of XQuery modules at startup, use the Dgraph `--disable_web_services` flag.

Determining file location in module import

When you import an XQuery library module, you specify the namespace on the import line. The Endeca implementation of XQuery also requires the use of a file location hint in library module import.

In the Endeca XQuery implementation, modules are considered concrete files. Therefore, in addition to the required namespace, location hints must be provided to import library module files for that namespace. For example, the file `typeahead.xq`, downloadable from the Endeca Developer Network at *http://eden.endeca.com/wiki/display/feature/XQuery*, contains the following import lines. As you can see, a location hint consisting of the module name follows the namespace declarations.

```
import module namespace http = "http://www.endeca.com/XQuery/http/2008" at
 "http.xq";
import module namespace mdex = "http://www.endeca.com/XQuery/mdex/2008" at
 "mdex.xq";
import module namespace eutil = "http://www.endeca.com/XQuery/eutil/2008"
at "eutil.xq";
import module namespace esoap = "http://www.endeca.com/XQuery/esoap/2008"
at "esoap.xq";
import module namespace typeahead-wsdl = "http://www.endeca.com/XQuery/ser¬
vice/typeahead/wsdl/2008" at "typeahead-wsdl.xq";
```

Library module files are found by looking up the specified name in the library search path. If `--xquery_path` is specified, its `lib` subdirectory is included in the library search path.

Developer checklist

Before you begin writing a Web service main module, you need to decide which of the supported protocols (such as SOAP or XML-RPC) your service will conform to.

Regardless of the protocol you choose, the code you write needs to meet the following requirements:

- It must use the `http:*` external functions to get the URL, any related parameters, or any POST data and process these items as request data according to your choice of Web service protocol. An example in SOAP would be unpacking a SOAP request.
- It must contain the actual functionality of your service, which may include one or more calls to the MDEX API through XQuery.

- You must make the return value of your module be a valid response value according to your choice of Web service protocol. For example, in SOAP, it would have to be a valid SOAP response.

Tools to support Web services and XQuery for Endeca development

You can write your XQuery main modules in any XQuery editor or text editor. In order to test your Web services, you need a tool like Progress Software's Stylus Studio®, Eviware's soapUI, or Altova's XMLSpy® that you can use to send requests and debug responses.

Performance expectations for Web services

The performance of Web services written in XQuery can vary considerably from service to service, but it is possible to write a service that performs well.

For example, the Typeahead SOAP Web service can run at over 1000 requests/second. A full Navigation Web service, in which each request to XQuery contains at least three underlying queries to the data in the MDEX Engine (that is to say, the one Web service request results in three Presentation API-equivalent requests), can run at over 150 requests/second.

The fact that the MDEX Engine is serving XML or SOAP, or doing extra processing in XQuery, does not prohibit fast services from being built, though there are considerations that a developer should keep in mind when building a service for speed. Some things that affect the performance of a service include:

- The amount of data examined in XQuery.
- The speed and number of MAX queries.
- The complexity of the XML transformations performed.
- Use of `fn:doc()` to retrieve data from external sources, which blocks query evaluation for the duration of the call.
- Use of `eutil:eval()`, which adds a significant delay while the arguments to `eval` are compiled.

For more detailed guidance on performance, see the *Performance Tuning Guide*.

Performance expectations for the MDEX Web service

Because the MDEX Web service compiles a large main module for each query, its performance may be slower than that of custom Web services.

Exposing XQuery main modules as Web services

To add your own functionality to the MDEX Engine, you can create your own XQuery main modules and supporting library modules, and then expose the main modules as Web services.

The high-level steps for Web services and XQuery for Endeca development are outlined below. Following the outline, we provide a simple example.

1. Create your XQuery main module in a text editor.
2. Save the file with an `.xq` extension.
3. Restart the Dgraph, specifying `--xquery_path` with the path to the directory containing the new main module file.
4. Look at the Dgraph error log to uncover any errors.

5. Try your function in soapUI or another similar tool, sending in requests and debugging responses. Dynamic errors resulting from the call to the module are reported in the Dgraph error log.
6. Each time you change your main module, reload it using the `admin?op=reload-services` administrative option. When you run `reload-services`, errors in syntax are reported in the Dgraph error log.

Web services and XQuery for Endeca example

The following example implements a simple SOAP Web service that searches through the English property in a sample MDEX data set and returns the number of occurrences of a single search term.

An example XQuery main module

This topic contains the XQuery main module `example.xq`.

The main module is broken down to highlight its component sections, including the following:

- Module import and namespace declaration.

```
import module namespace http = "http://www.endeca.com/XQuery/http/2008"
 at "http.xq";
import module namespace eutil = "http://www.endeca.com/XQuery/eutil/2008"
 at "eutil.xq";
import module namespace mdex = "http://www.endeca.com/XQuery/mdex/2008"
 at "mdex.xq";

declare namespace soap = "http://schemas.xmlsoap.org/soap/envelope/";
declare namespace mdata = "http://www.endeca.com/MDEX/data/IR600";
declare namespace ex = "http://endeca.com/example.xsd";
```

- SOAP processing.

```
declare function local:generateFaultResponse($message as xs:string*) as
 element(soap:Envelope, xs:untyped)
{
   <soap:Envelope>
      <soap:Body>
         <soap:Fault>
            <mdata:query-fault>
               <mdata:fault-string>{$message}</mdata:fault-string>
            </mdata:query-fault>
         </soap:Fault>
      </soap:Body>
   </soap:Envelope>
};
```

- Generating embedded WSDL.

```
declare function local:generateWsdl()
{
<definitions name="ExampleMDEXQuery"
             targetNamespace="http://endeca.com/example.wsdl"
             xmlns:es="http://endeca.com/example.wsdl"
             xmlns:esxsd="http://endeca.com/example.xsd"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">
```

```
  <types>
    <xsd:schema
      targetNamespace="http://endeca.com/example.xsd"
      elementFormDefault="qualified"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">

      <!-- request contains just a word to be matched -->
      <xsd:element name="MatchWord">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="word" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <!-- response contains just the number of matches -->
      <xsd:element name="MatchCount">
        <xsd:complexType>
          <xsd:all>
            <xsd:element name="count" type="xsd:int"/>
          </xsd:all>
        </xsd:complexType>
      </xsd:element>

    </xsd:schema>
  </types>

  <!-- request message uses an element from the above schema -->
  <message name="CountMatchesInput">
    <part name="word" element="esxsd:MatchWord"/>
  </message>

  <!-- response message uses an element from the above schema -->
  <message name="CountMatchesOutput">
    <part name="count" element="esxsd:MatchCount"/>
  </message>

  <!-- port type specifies the above request and response message -->
  <portType name="ExampleMDEXPortType">
    <operation name="CountMatches">
      <input message="es:CountMatchesInput"/>
      <output message="es:CountMatchesOutput"/>
    </operation>
  </portType>

  <!-- binding specifies the above port type and declares this
       web service to be of the document/literal variety -->
  <binding name="ExampleMDEXSoapBinding" type="es:ExampleMDEXPortType">
    <soap:binding style="document" transport="http://schemas.xml¬
soap.org/soap/http"/>
    <operation name="CountMatches">
      <soap:operation soapAction="http://endeca.com/Example"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <!-- associate the above port type and binding -->
  <service name="ExampleMDEXService">
    <documentation>Endeca Example MDEX Query Service</documentation>
   <port name="ExampleMDEXPortType" binding="es:ExampleMDEXSoapBinding">

      <soap:address location="http://endeca.com/mdex"/>
    </port>
  </service>

</definitions>
};
```

- MDEX API through XQuery calls, wrapped in `fn:trace()` for ease of debugging.

```
(: get the request :)
let $body-str := fn:trace(http:get-body(), "REQUEST")
return
  if (fn:exists(http:get-query-parameter("WSDL"))) then
    (: caller used ?WSDL request format -- send WSDL :)
    local:generateWsdl()
  else if (fn:empty($body-str)) then
    (: no request body -- send SOAP fault :)
    local:generateFaultResponse("Empty body in SOAP request")
  else
    (: reach into the SOAP request body :)
    let $body := eutil:parse(fn:exactly-one($body-str))/soap:Enve¬
lope/soap:Body
    (: pull out the input message :)
    let $query := $body/ex:MatchWord
    return
      if (fn:count($query) ne 1) then
        (: should be exactly one input message :)
        local:generateFaultResponse($query)
      else
        (: start making a response enveleope :)
        fn:trace(
        <soap:Envelope>
          <soap:Body>
            {
            (: setup an MDEX API call to do the text search :)
            let $str := fn:data($query/ex:word)
            let $cl :=
              <mdata:Query>
                <mdata:Searches>
                  <mdata:Search Key = "English">{$str}</mdata:Search>
                </mdata:Searches>
              </mdata:Query>
            (: call MDEX API :)
            let $result := mdex:navigation-query($cl)
            (: fill in an output message :)
            return
                <ex:MatchCount>
                  <ex:count>
                  {
                 fn:data($result/mdata:RecordsResult/@TotalRecordCount)

                  }
                  </ex:count>
                </ex:MatchCount>
            }
          </soap:Body>
        </soap:Envelope>, "RESPONSE")
```

Note: After writing your main module, make sure you save it to the directory specified in the `--xquery_path` flag.

Example request and response bodies

This example request searches the English property for the word *two*.

Sending a request message like this one:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:exam="http://endeca.com/example.xsd">
   <soapenv:Body>
      <exam:MatchWord>
         <exam:word>two</exam:word>
      </exam:MatchWord>
   </soapenv:Body>
</soapenv:Envelope>
```

Returns a response message like this:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ex:MatchCount xmlns:ex="http://endeca.com/example.xsd">
         <ex:count>19</ex:count>
      </ex:MatchCount>
   </soap:Body>
</soap:Envelope>
```

Using the MDEX Web service

This section describes how to get started with the MDEX Web service, which is included with Web services and XQuery for Endeca.

About the MDEX Web service

The MDEX Web service, which is included as part of this release, allows you to use XQuery with the MDEX Engine without writing custom Web services.

The MDEX Web service consists of an `mdex.xq` main module and `mdex.wsdl` file. These files implement a service that provides a SOAP wrapper for the MAX API.

The MDEX Web service is a WS-I compliant, WSDL/SOAP 1.x Web service. Along with any other Web services included in the installation, it is loaded automatically upon MDEX Engine startup, unless the engine is started with the `--disable_web_services` flag.

Bulk export is not supported in this release of the MDEX Web service.

Note: The MDEX Web service included as part of this release was called the Query Web service in release 6.1.1 and earlier versions.

What the MDEX Web service does

The MDEX Web service is not meant as a Web service replacement to the Presentation API, but rather as a data service similar to something that a relational database management system might provide. In an RDBMS scenario, SQL input transported via ODBC or JDBC returns tabular results. Similarly, in an MDEX scenario, XQuery input transported via the MDEX Web service returns XML results.

The MDEX Web service accepts an XQuery main module as its request argument, and returns the result of evaluating that module in the MDEX. The main module is a string that can be constructed programatically in your application. The module contained in each request is compiled in the MDEX Engine on the fly. Such main modules may call library modules that have been loaded into the MDEX Engine, including custom library modules.

If your XQuery logic is very simple, your application has modest performance requirements, or you need complex application-tier logic to generate your XQuery code on the fly, you will benefit from the MDEX Web service. In particular, if your XQuery logic consists simply of MDEX API through XQuery (or MAX) calls, the MDEX Web service has an important advantage: the WSDL for it will allow stub generators to provide you with bindings for all of the MAX return types. In general, generated stubs will return results in this form whenever the query's return type is a MAX return type. When you run XQuery code that does not return MAX types, or embeds them in other elements, the generated stubs will provide you with an XML DOM tree, which you will have to traverse to obtain the result data.

Using the MDEX Web service in conjunction with custom Web services

The MDEX Web service is useful during XQuery development. It provides a convenient way to try out an ad hoc XQuery code fragment against the MDEX Engine, including ad hoc calls to library modules that you have written earlier. A convenient way to develop XQuery applications is to start with simple MDEX Web service calls that you gradually evolve by factoring out function declarations and moving them into library module files loaded into the MDEX Engine. In this scenario, at any point in time your well-tested and stable code lives in library module files, and your new and experimental function declarations and query body are sent to the MDEX Engine using the MDEX Web service. You may then choose to move the library module to a file loaded into the MDEX Engine as well, completing the transition to a custom Web service.

Note: The creation of XQuery modules, aside from those provided with the MDEX distribution, requires the purchase of the Advanced Query Module. Contact your Endeca representative for details.

Updating with the MDEX Web service

The MDEX Web service uses `eutil:eval()`, which is a non-updating function, and cannot, therefore, invoke updating expressions.

If you want to use XQuery update functionality, you must create custom Web services.

Inputs and outputs of the MDEX Web service

This topic describes inputs and outputs to the MDEX Web service.

The MDEX Web service takes as its input a the contents of a SOAP envelope in the form of a string that is an XQuery main module. That element can consist of a MAX function, pathing, or some other string. Main modules sent to the MDEX Web service have the MAX libraries imported automatically.

The service can return two kinds of results:

- MAX function calls produce typed results. These typed results are defined in the `mdex.wsdl`, which can be used to generate stub files.
- Any result that is not a direct output of a MAX function call, including modified MAX results, is wrapped in an untyped XML response, with each element of a sequence returned as a separate response. Untyped results contain a single `xs:any` element containing two elements: a root result element, and a child element containing the actual result of the XQuery, wrapped in the result element.

Examples

The following examples illustrate both a MAX query and a non-MAX query. In both cases, the input is a Request element wrapped in a SOAP envelope that takes a string which is valid XQuery.

In the following MAX example, a CDATA is used to avoid complicated escaping:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
  soap/envelope/" xmlns:ns="http://www.endeca.com/MDEX/data/IR600">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:Request>
     <![CDATA[
        mdex:navigation-query(<mdata:Query/>)
        ]]>
      </ns:Request>
   </soapenv:Body>
</soapenv:Envelope>
```

The output of this example would consist of a single `Response` element. Evaluated XQuery returns a sequence of nodes, with the `Response` element containing one `Result` element for each item in the return sequence. (The following code sample is truncated for the sake of brevity).

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
  soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <mdata:Response xmlns:mdata="http://www.endeca.com/MDEX/
        data/IR600">
         <mdata:TypedResult>
            <NavigationResults xmlns="http://www.endeca.com/
              MDEX/data/IR600">
               <Dimensions>
                  ...
                  <Dimension Name="Designation" Id="7"
                    MultiSelect="None" GroupName="Ratings">
                     <DimensionValue Name="Designation" Id="7"
                       IsLeaf="false" IsNavigable="false">
                        <DimensionValues>
                           <DimensionValue Name="Highly Recommended"
                             Id="8029" IsLeaf="true" IsNavigable="true"/>
                           <DimensionValue Name="Best Buy" Id="8031"
                             IsLeaf="true" IsNavigable="true"/>
                        </DimensionValues>
                     </DimensionValue>
                  </Dimension>
                   ...
                 </Dimensions>
               <NavigationStatesResult>
                  <DimensionStates>
                     <DimensionState DimensionName="Wine Type"
                       DimensionId="6200">
                        <Refinements ParentName="Wine Type"
```

```
                          ParentId="6200" HasMore="false"
                          IsRefinable="true"/>
                     </DimensionState>
                      ...
                   </DimensionStates>
               </NavigationStatesResult>
               <RecordsResult Offset="0" RecordsPerPage="10"
                 TotalRecordCount="57076">
                  <Records>
                     <Record Id="34699">
                        <Attributes>
                           <AssignedDimensionValue Key="Review Score"
                             DimensionId="9" Id="21">0 to 10
                           </AssignedDimensionValue>
                           <Property Key="P_DateReviewed">
                             08/31/95</Property>
                           <Property Key="P_Description">Supple
                             and polished cedar, coffee, cherry and berry

                            flavors. This is elegant, finishing with firm

                             tannins and good length. Drinkable now.
                           </Property>
                           <Property Key="P_Name">A Red Blend
                             Alexander Valley</Property>
                           <Property Key="P_Price">18.000000</Property>
                           <Property Key="P_Region">Sonoma</Property>
                           <Property Key="P_Score">5</Property>
                           <Property Key="P_WineID">34699</Property>
                           <Property Key="P_Winery">Lyeth</Property>
                           <Property Key="P_WineType">Cabernet Blend
                           </Property>
                           <Property Key="P_WineType">Red</Property>
                           <Property Key="P_Year">1992</Property>
                        </Attributes>
                     </Record>
                      ...
                   </Records>
               </RecordsResult>
                ...
            </NavigationResults>
         </mdata:TypedResult>
      </mdata:Response>
   </soapenv:Body>
</soapenv:Envelope>
```

The following non-MAX example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
  soap/envelope/" xmlns:ns="http://www.endeca.com/MDEX/data/IR600">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:Request>for $x in (1 to 5) return $x</ns:Request>
   </soapenv:Body>
</soapenv:Envelope>
```

Returns this:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/
  soap/envelope/">
   <soap:Body>
```

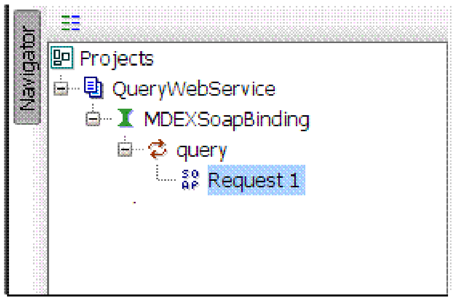
```
      <mdex:Response xmlns:mdata="http://www.endeca.com/MDEX/
        data/IR600">
         <mdata:UntypedResult>
            <mdata:Result>1</mdata:Result>
         </mdata:UntypedResult>
         <mdata:UntypedResult>
            <mdata:Result>2</mdata:Result>
         </mdata:UntypedResult>
         <mdata:UntypedResult>
            <mdata:Result>3</mdata:Result>
         </mdata:UntypedResult>
         <mdata:UntypedResult>
            <mdata:Result>4</mdata:Result>
         </mdata:UntypedResult>
         <mdata:UntypedResult>
            <mdata:Result>5</mdata:Result>
         </mdata:UntypedResult>
      </mdata:Response>
   </soap:Body>
</soap:Envelope>
```

Invoking the MDEX Web service

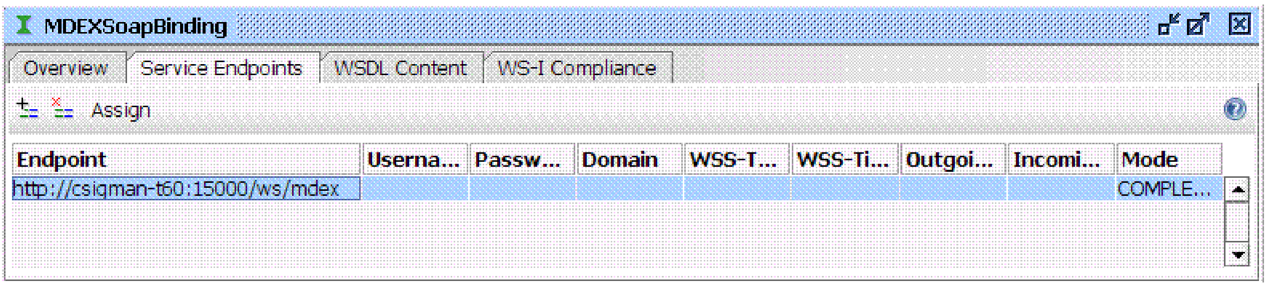
This very simple example demonstrates how to create a Web services call to the MDEX service.

You can perform this task in the WSDL tool of your choice, such as Stylus Studio or soapUI. We use soapUI here. A free version of this tool can be found at *http://www.soapui.org*.

1. Create a new project in the soapUI tool. This includes giving the project a name and pointing it at the `mdex.wsdl` file, which is returned from `http://<hostname:port>/ws/mdex?wsdl`.

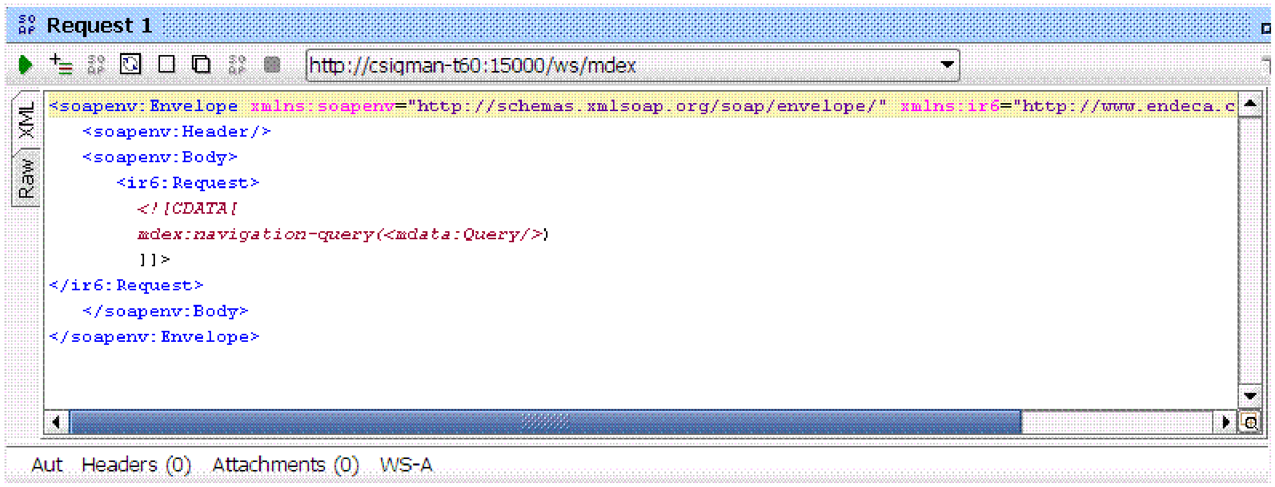


2. In the MDEXSoapBinding element, click the Service Endpoints tab and replace <hostname:port> with your information to set the service endpoint.



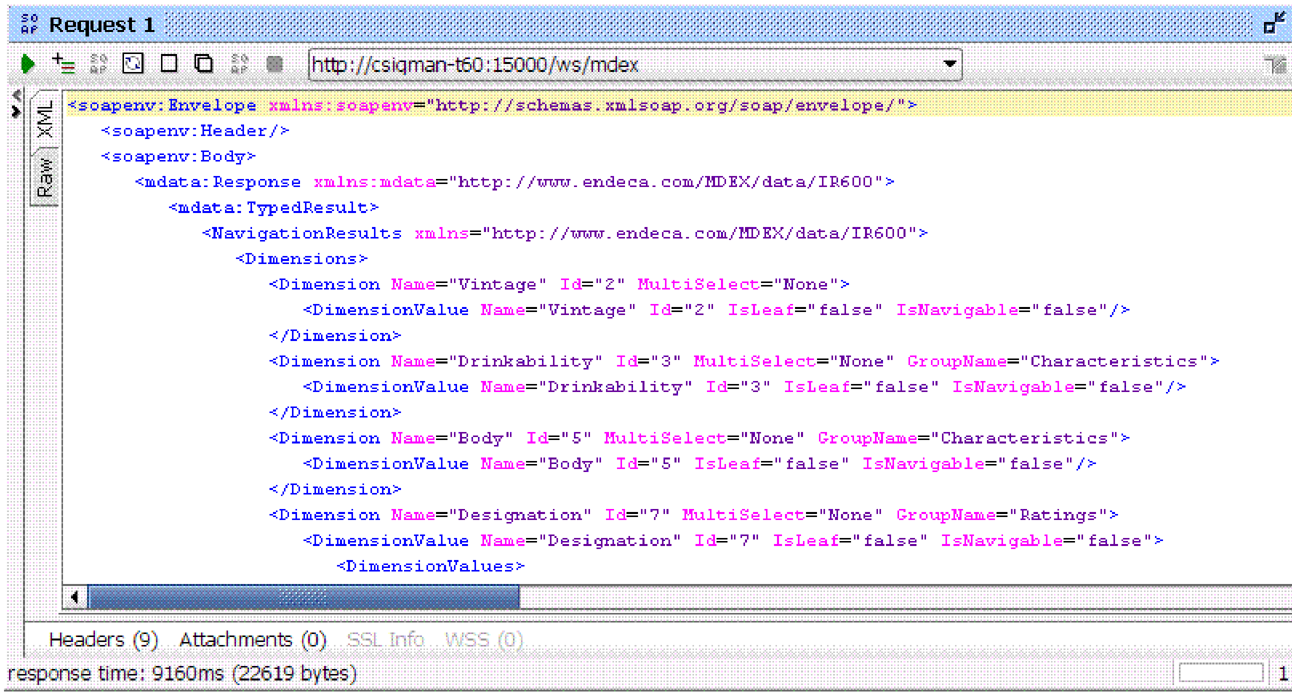
Endpoint	Userna...	Passw...	Domain	WSS-T...	WSS-Ti...	Outgoi...	Incomi...	Mode
http://csiqman-t60:15000/ws/mdex								COMPLE...

3. Put a SOAP request to the port and execute the service by clicking the green arrow in the upper left corner of the SOAP Request window. Here, we make a simple navigation query request.


Request 1
http://csigman-t60:15000/ws/mdex
XML
Raw
Aut Headers (0) Attachments (0) WS-A

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ir6="http://www.endeca.c
   <soapenv:Header/>
   <soapenv:Body>
      <ir6:Request>
        <![CDATA[
        mdex:navigation-query(<mdata:Query/>)
        ]]>
</ir6:Request>
   </soapenv:Body>
</soapenv:Envelope>
```

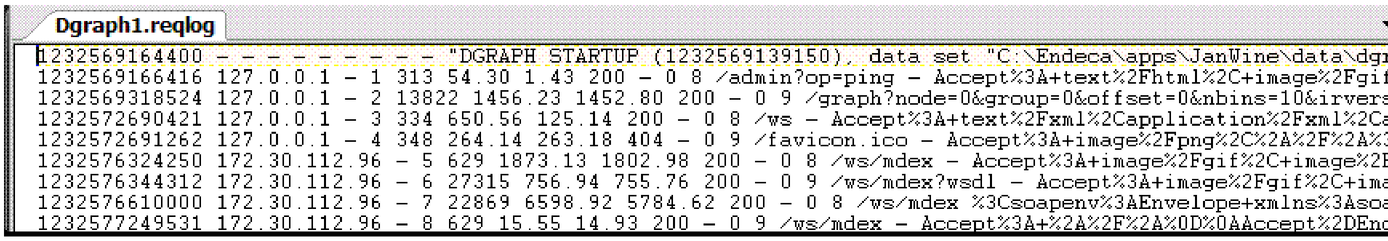
4. If the service executes successfully, soapUI displays the response message on the right side of the window and posts the response time in the status bar at the bottom of the screen. (If the service encounters any problems, details appear in the status bar.)


Request 1
http://csigman-t60:15000/ws/mdex
XML
Raw
Headers (9) Attachments (0) SSL Info WSS (0)

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <mdata:Response xmlns:mdata="http://www.endeca.com/MDEX/data/IR600">
         <mdata:TypedResult>
            <NavigationResults xmlns="http://www.endeca.com/MDEX/data/IR600">
               <Dimensions>
                  <Dimension Name="Vintage" Id="2" MultiSelect="None">
                     <DimensionValue Name="Vintage" Id="2" IsLeaf="false" IsNavigable="false"/>
                  </Dimension>
                  <Dimension Name="Drinkability" Id="3" MultiSelect="None" GroupName="Characteristics">
                     <DimensionValue Name="Drinkability" Id="3" IsLeaf="false" IsNavigable="false"/>
                  </Dimension>
                  <Dimension Name="Body" Id="5" MultiSelect="None" GroupName="Characteristics">
                     <DimensionValue Name="Body" Id="5" IsLeaf="false" IsNavigable="false"/>
                  </Dimension>
                  <Dimension Name="Designation" Id="7" MultiSelect="None" GroupName="Ratings">
                     <DimensionValue Name="Designation" Id="7" IsLeaf="false" IsNavigable="false">
                        <DimensionValues>
```

response time: 9160ms (22619 bytes)

5. Confirm the result in the Dgraph request log. In the final four lines shown below, you can see a record of Web service execution.


Dgraph1.reqlog

```
1232569164400 - - - - - - - - - - "DGRAPH STARTUP (1232569139150), data set "C:\Endeca\apps\JanWine\data\dgr
1232569166416 127.0.0.1 - 1 313 54.30 1.43 200 - 0 8 /admin?op=ping - Accept%3A+text%2Fhtml%2C+image%2Fgif
1232569318524 127.0.0.1 - 2 13822 1456.23 1452.80 200 - 0 9 /graph?node=0&group=0&offset=0&nbins=10&irvers
1232572690421 127.0.0.1 - 3 334 650.56 125.14 200 - 0 8 /ws - Accept%3A+text%2Fxml%2Capplication%2Fxml%2Ca
1232572691262 127.0.0.1 - 4 348 264.14 263.18 404 - 0 9 /favicon.ico - Accept%3A+image%2Fpng%2C%2A%2F%2A%3
1232576324250 172.30.112.96 - 5 629 1873.13 1802.98 200 - 0 8 /ws/mdex - Accept%3A+image%2Fgif%2C+image%2B
1232576344312 172.30.112.96 - 6 27315 756.94 755.76 200 - 0 9 /ws/mdex?wsdl - Accept%3A+image%2Fgif%2C+ima
1232576610000 172.30.112.96 - 7 22869 6598.92 5784.62 200 - 0 8 /ws/mdex %3Csoapenv%3AEnvelope+xmlns%3Asoa
1232577249531 172.30.112.96 - 8 629 15.55 14.93 200 - 0 9 /ws/mdex - Accept%3A+%2A%2F%2A%0D%0AAccept%2DEnc
```

About the mdex.wsdl file

The `mdex` schema used by MDEX Web service is defined in the `mdex.wsdl` file.

The `mdex.wsdl` file is located in `$ENDECA_MDEX_ROOT/xquery/xml` directory. You can use the WSDL as an API to write your query in the language of your choice (such as Java, C#, or Perl).The WSDL for the MDEX Web service is retrievable directly from the MDEX Engine by way of an HTTP GET request.

Requesting the WSDL

You can get the WSDL for the MDEX Web service by adding `?wsdl` to the URL for the service.

The syntax of the request is `/ws/mdex?wsdl`.

Note: The `mdex?wsdl` service pulls the `Host` header directly from the HTTP request.

Namespaces used by the MDEX Web service

The MDEX Web service uses two pre-defined namespaces: `mdex` and `mdata`.

MDEX lib modules and MAX functions are automatically imported under the prefix `mdex` (http://www.endeca.com/mdex/data/IR600).

Any MAX results are returned under the prefix `mdata`.

Note: Do not attempt to redefine these two namespaces.

Supported binding generators for the MDEX Web service

If you want to generate bindings for MDEX Web service client stubs, you must use one of the supported binding generators.

The following are supported:

- Axis2 version 1.4.1, using XMLBeans. Note that bindings generated with the ADB will not function properly.
- .NET 2.0 Framework (or later) `wsdl.exe` tool.

Exception handling in the MDEX Web service

The MDEX Web service runs user-supplied XQuery code and generates SOAP responses.

If the code throws an exception it cannot handle, the generated SOAP response contains a SOAP fault structure that includes detailed information about the exception. If a SOAP error occurs while processing the request, the SOAP HTTP server issues an HTTP 500 "Internal Server Error" response and includes a SOAP message in the response. This SOAP message contains a SOAP Fault element indicating the SOAP processing error.

The MDEX Web service uses try/catch expressions to catch exceptions. Exceptions thrown can be one of the following types:

- Dynamic errors, which emerge during processing.
- Static errors, which include typos and parsing errors.
- Type-checking errors.

For more information about Endeca's implementation of try/catch, see the topic "XQuery try/catch expressions."

SOAP Fault details

The SOAP Faults generated by the MDEX Web service are SOAP 1.1 compliant.

A SOAP Fault is an optional SOAP element that provides information about errors that prevent a request from being processed successfully. The information in the SOAP Fault makes it easier to debug problems in a SOAP Web service. In the case of an error, the MDEX Web service returns a SOAP Fault that wraps a detailed `mdata:Fault` element.

The structure of a fault is as follows:

```
<soap-envelope:Fault>
     <soap-envelope:faultcode> ... </soap-envelope:faultcode>
     <soap-envelope:faultstring> ... </soap-envelope:faultstring>
     <soap-envelope:detail>
          <mdata:Fault>
               <mdata:ErrorCode> ... </mdata:ErrorCode>
               <mdata:ErrorDetail> ... </mdata:ErrorDetail>
               <mdata:ErrorSequence> ... </mdata:ErrorSequence>
               <mdata:StackTrace> ... </mdata:StackTrace>
          </mdata:Fault>
     </soap-envelope:detail>
</soap-envelope:Fault>
```

- The `faultcode` element is one of the following: "MustUnderstand", "Client", or "Server".
- The `faultstring` element contains a description of the error.
- The `detail` element contains the `mdata:Fault` element, which holds the following details about the exception:
  - The `ErrorCode` element contains the fault code generated by the MAX API. For details about these codes, see the topic "External function error codes."
  - The `ErrorDetail` element contains the same description of the error found in `faultstring`.
  - The `ErrorSequence` element contains the items provided in the third argument to the `fn:error()` function (if the exception was generated by a call to the three-argument form of `fn:error()`). If you only call MAX API functions, this element will always be empty, because MAX API functions do not use the three-argument form of `fn:error()`.
  - The `StackTrace` element represents the stack trace associated with the current exception. The stack trace, by identifying the expression where an error occurred and the expressions it

was called from, makes it easier to debug errors in your XQuery code. For more information about stack traces, see the topic "eutil:get-stack-trace()."

SOAP fault schema used by the MDEX Web service

This fragment of the `mdex.wsdl` contains the SOAP fault schema used by the MDEX Web service, along with other information.

```
<complexType name="Location">
        <attribute name="line" type="string" use="required" />
        <attribute name="column" type="string" use="required" />
      </complexType>

      <complexType name="Span">
        <sequence>
          <element name="Start" type="tns:Location"/>
          <element name="End" type="tns:Location"/>
        </sequence>
        <attribute name="uri" type="string" use="required" />
      </complexType>

      <complexType name="StackTrace">
        <sequence>
          <element name="Span" type="tns:Span"
            maxOccurs="unbounded" />
        </sequence>
      </complexType>

      <complexType name="ErrorSequence">
        <sequence>
           <any processContents="lax" minOccurs="0"
             maxOccurs="unbounded" />
        </sequence>
      </complexType>

      <element name="Fault">
          <complexType>
               <sequence>
                   <element name="ErrorCode"
                     type="string" />
                   <element name="ErrorDetail"
                     type="string" />
                   <element name="ErrorSequence"
                     type="tns:ErrorSequence" />
                   <element name="StackTrace"
                     type="tns:StackTrace"/>
               </sequence>
         </complexType>
      </element>

    <message name="MDEXFault">
        <part name="fault" element="tns:Fault"/>
    </message>

    <portType name="MDEXPort">
        <operation name="query">
            <input name="request"
              message="tns:MDEXRequest"/>
            <output name="response"
              message="tns:MDEXResponse"/>
            <fault name="fault"
              message="tns:MDEXFault"/>
        </operation>
    </portType>
```

Examples using SOAP Faults within the MDEX Web service

This topic contains two examples of SOAP Faults returned for exceptions thrown in the MDEX Web service.

Exception thrown due to a typo in query

In the following example, a misspelling of the attribute `RelevanceRankingStrategy` causes an exception to be thrown. This is a static error.

Here is the query:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
  envelope/" xmlns:ir6="http://www.endeca.com/MDEX/data/IR600">
   <soapenv:Header/>
   <soapenv:Body>
      <ir6:Request><![CDATA[mdex:dimension-search-query(<mdata:Query>
        <mdata:DimensionSearch RelevanceRankingSrategy="Exact">1
        </mdata:DimensionSearch></mdata:Query>)]]></ir6:Request>
   </soapenv:Body>
</soapenv:Envelope>
```

Here is the response:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org
  /soap/envelope/">
 <soapenv:Header/>
 <soapenv:Body>
    <soapenv:Fault>
       <faultcode>soapenv:Client</faultcode>
       <faultstring>exception encountered while executing external
         function 'internal:schema-validate-query-input', caused
         by error endeca-client-err:MAXF0001 : Schema Validation
         Error. : Invalid xml: Attribute 'RelevanceRankingSrategy'
         is not declared for element 'DimensionSearch'
       </faultstring>
       <detail>
          <mdata:Fault xmlns:mdata="http://www.endeca.com/MDEX/data/
            IR600">
             <mdata:ErrorCode>endeca-client-err:MAXF0001
             </mdata:ErrorCode>
             <mdata:ErrorDetail>exception encountered while
               executing external function 'internal:schema-
               validate-query-input', caused by error
               endeca-client-err:MAXF0001 : Schema
               Validation Error. : Invalid xml: Attribute
               'RelevanceRankingSrategy' is not declared for
               element 'DimensionSearch'</mdata:ErrorDetail>
             <mdata:ErrorSequence/>
             <mdata:StackTrace>
                <mdata:Span uri="%28INTERNAL%29%2Fmdex_internal_
                  dimension_search.xq">
                   <mdata:Start line="20" column="75"/>
                   <mdata:End line="20" column="143"/>
                </mdata:Span>
                <mdata:Span uri="%28INTERNAL%29%2Fmdex.xq">
                   <mdata:Start line="44" column="2"/>
                   <mdata:End line="44" column="58"/>
                </mdata:Span>
                <mdata:Span uri="">
                   <mdata:Start line="2" column="0"/>
                   <mdata:End line="2" column="136"/>
                </mdata:Span>
                <mdata:Span uri="<ENDECA_MDEX_ROOT>%2Fxquery%2Fmdex.xq">
                   <mdata:Start line="93" column="15"/>
                   <mdata:End line="93" column="46"/>
                </mdata:Span>
                <mdata:Span uri="<ENDECA_MDEX_ROOT>%2Fxquery%2Fmdex.xq">
                   <mdata:Start line="165" column="4"/>
                   <mdata:End line="171" column="5"/>
                </mdata:Span>
                <mdata:Span uri="<ENDECA_MDEX_ROOT>%2Fxquery%2Fmdex.xq">
                   <mdata:Start line="183" column="2"/>
                   <mdata:End line="183" column="24"/>
                </mdata:Span>
             </mdata:StackTrace>
          </mdata:Fault>
       </detail>
    </soapenv:Fault>
 </soapenv:Body>
</soapenv:Envelope>
```

An exception thrown by an external function passing an invalid argument

In this example, an exception is thrown by external function `dimension-value-id-from-path` when an invalid dimension name is passed as argument. This is a dynamic error.

Here is the query:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org
  /soap/
  envelope/" xmlns:ir6="http://www.endeca.com/MDEX/data/IR600">
   <soapenv:Header/>
   <soapenv:Body>
      <ir6:Request><![CDATA[mdex:dimension-value-id-from-path
       (("NoSuchDimension"))]]></ir6:Request>
   </soapenv:Body>
</soapenv:Envelope>
```

Here is the response:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
  soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
     <soapenv:Fault>
        <faultcode>soapenv:Client</faultcode>
        <faultstring>exception encountered while executing external
          function 'mdex:dimension-value-id-from-path', caused by
          error endeca-client-err:MDEX0007 : Dimension not found
        </faultstring>
        <detail>
           <mdata:Fault xmlns:mdata="http://www.endeca.com/MDEX/
             data/IR600">
```

```
              <mdata:ErrorCode>endeca-client-err:MDEX0007
                </mdata:ErrorCode>
              <mdata:ErrorDetail>exception encountered while
                executing external function 'mdex:dimension-
                value-id-from-path', caused by error
                endeca-client-err:MDEX0007 : Dimension not found
              </mdata:ErrorDetail>
              <mdata:ErrorSequence/>
              <mdata:StackTrace>
                 <mdata:Span uri="">
                    <mdata:Start line="2" column="0"/>
                    <mdata:End line="2" column="42"/>
                 </mdata:Span>
                 <mdata:Span uri="<ENDECA_MDEX_ROOT>%2Fxquery%2Fmdex.xq">
                    <mdata:Start line="93" column="15"/>
                    <mdata:End line="93" column="46"/>
                 </mdata:Span>
                 <mdata:Span uri="<ENDECA_MDEX_ROOT>%2Fxquery%2Fmdex.xq">
                    <mdata:Start line="165" column="4"/>
                    <mdata:End line="171" column="5"/>
                 </mdata:Span>
                 <mdata:Span uri="<ENDECA_MDEX_ROOT>%2Fxquery%2Fmdex.xq">
                    <mdata:Start line="183" column="2"/>
                    <mdata:End line="183" column="24"/>
                 </mdata:Span>
              </mdata:StackTrace>
           </mdata:Fault>
        </detail>
     </soapenv:Fault>
  </soapenv:Body>
</soapenv:Envelope>
```

Returning non-MAX XML in the MDEX Web service

If you are returning non-MAX XML, ensure that your query does not return a result where the root element name is the same as a return type of MAX.

If you do, the client bindings will fail to parse the result and the query will generate an error.

Using the exquery command-line tool

Endeca provides a command-line developer tool, exquery, that you can use to run XQuery. It can be used in interactive or file-based mode to test and debug the XQuery functions that you write yourself.

Exquery is run from the `$ENDECA_MDEX_ROOT/bin` directory.

Important: This version of the exquery tool is for development use only. It is not intended or supported for production use. The MDEX API through XQuery is not available in the exquery tool. In terms of external functions, `http` functions are not available in the exquery tool, but `eutil` functions are.

Application debugging with exquery

When building an application, you can use the exquery tool to develop and debug much of your XQuery code before running it in the Dgraph.

However, exquery neither implements nor resolves the `http` and `mdex` external functions provided by the Endeca XQuery implementation. That means that if your code references these functions, it will not compile correctly in exquery.

In order to take advantage of the exquery debugging environment, you can temporarily replace the unimplemented external functions with local functions that refer to dummy XML data, so that the rest of the code can be validated. You can supply the dummy data in two ways: inline, or by using `fn:doc()` to return XML data that is located in a separate file.

For example, if your code refers to the `mdex:dimension-search-query` external function, you might declare a stand-in function similar to the following:

```
declare function local:mydimension-search-query($query as element(mda¬
ta:Query, xs:untyped)) as element(mdata:Results, xs:untyped)
{
    fn:exactly-one(fn:doc("path-to-sample-output.xml")/*)
};
```

Then, anywhere your code calls `mdex:dimension-search-query`, modify it to call `local:mydimension-search-query` instead. When you have finished developing and testing your code, you can replace the references to `mdex:dimension-search-query` before running the query in the Dgraph.

Note: Remember to declare the `local` namespace in your main module.

Exquery usage

This topic defines the usage of the exquery command-line tool.

You can access the usage message in the tool by using the `--help` option.

```
Usage:
  Evaluate expression: exquery <expr>
  Print AST: exquery -a <expr>
  Normalize expression: exquery -n <expr>
  Expand and normalize expression: exquery -e <expr>
  Print expression static type after expansion and normalization: exquery
-t <expr>

  -f <fname>: executes the query in the XQuery main module in <fname>
  -fn <fname>: normalizes the query in the XQuery main module in <fname>
  -c <path-to-xml-file>: makes XML file available via fn:collection()
  -i <path-to-xml-file>: makes XML file available via .: the context item
```

Exquery access to environment variables

The exquery tool can use user-defined environment variables as external variables in XQuery.

In exquery, any variable in the namespace `http://www.endeca.com/XQuery/exquery/2008/env` and declared to be external will be an `xs:untypedAtomic` variable. Its value will be taken from its environment, if it is defined.

For example, consider the following session in the bash shell:

```
$ export my_shell_var=saffron
$ exquery
Endeca XQuery!
%% declare namespace env = "http://www.endeca.com/XQuery/exquery/2008/env";
 declare variable $env:my_shell_var external; $env:my_shell_var
>> saffron
>>
%%
```

On Windows, use `'set varname=value'` instead of `'export'`.

Notes

- This only works in exquery, and not in the MDEX Engine.
- Environment variables are traditionally uppercase, and XQuery variables are traditionally lowercase, but both upper and lower case are allowed in each one. However, in Unix shells, you may not use dashes in an environment variable name, so another separator, like an underscore, must be used. On Windows, environment variables are case-insensitive, so the external variable in XQuery may be of any case. Once the variable is defined, the same capitalization must be used consistently in the program.

Chapter 3

Web Services and XQuery Components and Features

This section discusses the components and features that make up Web services and XQuery for Endeca. If you choose to copy XQuery snippets from this document, be aware that in some cases hard line breaks have been added to improve legibility. These extra line breaks must be removed in order for the XQuery snippets to work properly.

Implementation-defined behavior in XQuery for Endeca

The XQuery specification identifies implementation-defined features, the behavior of which may vary in different implementations. This section provides details about implementation-defined behavior in the Endeca XQuery implementation.

Static type checking

This topic discusses what static type checking is, why it is important, and how Endeca has implemented this feature.

Note: Several of the examples in this topic are shown in the format used by the command-line developer tool, exquery, that is provided by Endeca. You can use exquery in interactive or file-based mode to test and debug the XQuery functions that you write yourself. More information about exquery appears in the section "Getting Started with Web Services and XQuery for Endeca."

Static type checking attempts to determine at compile time whether an XQuery expression can encounter a type error at runtime. Endeca's XQuery implements restrictive static type checking. It performs static type checking when compiling XQuery modules and fails at compile time unless it can prove that a type error *cannot* occur at runtime, except in cardinality functions or `treat as` expressions, which perform explicit runtime type-checking. In contrast, some other XQuery implementations use permissive static type checking, which only fails when it can prove that a type error *must* occur at runtime.

Because Endeca's static type checking is stricter than that done by many other existing XQuery implementations, it is conservative, meaning that the fact that a program compiles means that it could not contain a type error. If an expression could fail for type reasons, it is marked. Therefore, if you are testing your XQuery code with another, more optimistic tool, such as Saxon or eXist, that implements static type checking differently, XQuery code that works in another tool may not work in the Endeca XQuery evaluator.

Usually, an optimistically valid program can be turned into a conservatively valid program by adding `treat as` statements and adding cardinality functions (such as `exactly-one()`) that make explicit the places where type errors could occur.

Type checking example

The following example shows a simple type checking error:

```
%% 1 + 'one'
>> Query failure: at 1:0-1:9: XPST0017: In function: _fs:plus() with 2 pa¬
rameters: Argument types '((xs:integer), (xs:string)) (where xs =
"http://www.w3.org/2001/XMLSchema")' do not match any signature for operator;
 possible signatures are '((xs:integer), (xs:integer)) (where xs =
"http://www.w3.org/2001/XMLSchema"); ((xs:decimal), (xs:decimal)) (where
xs = "http://www.w3.org/2001/XMLSchema"); ((xs:float), (xs:float)) (where
xs = "http://www.w3.org/2001/XMLSchema"); ((xs:double), (xs:double)) (where
 xs = "http://www.w3.org/2001/XMLSchema")'
```

To analyze this message, note the following:

- The span `1:0-1:9` shows that the type error is in the addition operation (which, in this case, is the whole program).
- The + operator is internally normalized to a function called `_fs:plus()`.
- The + operator takes either two integers, two decimals, two floats, or two doubles as arguments. (Expressions like `1 + 1.0e0` are valid because of type promotion.)
- The notes `(where xs = "http://www.w3.org/2001/XMLSchema")` are included to help fix errors where prefixes and namespaces in different contexts do not correspond exactly.

Static type checking and occurrence indicators

In XQuery, all values are considered to be sequences. The Endeca implementation of XQuery regards the occurrence indicators ? (that is, the item can appear zero times or one time), * (the item can appear zero or more times), and + (the item must appear one or more times), which regulate the sequence length of a valid value, as part of the type. The Endeca XQuery evaluator handles these indicators intelligently, knowing that anything that is a valid `xs:integer?` is also a valid `xs:integer*`. However, the Endeca XQuery evaluator will catch a misused quantifier if it does not match, as in the case of using an `xs:string+` as an `xs:string`. In the Endeca implementation of XQuery, whenever it cannot be statically determined that a sequence value has the cardinality required in a given context, a function such as `one-or-more()` or `exactly-one()` must be used to make run-time checking explicit. By contrast, many other XQuery implementations statically permit an expression if it could possibly work in any case.

Cardinality examples

In the following example:

```
%% string-to-codepoints('X') + 1
>> Query failure: at 1:0-1:29: XPTY0004: In function:
>> _fs:convert-operand() with 2 parameters: Expected type
>> 'xs:anyAtomicType?' and found type 'xs:integer*' (where xs =
>> "http://www.w3.org/2001/XMLSchema")
```

`convert-operand` is introduced in normalization around the arguments to the + operator, which does not allow sequences longer than one element. The compiler cannot prove that `string-to-codepoints('X')` returns a sequence with only one element, but we know that it will, so we can do the following:

```
%% exactly-one(string-to-codepoints('X')) + 1
>> 89
```

In this next example:

```
%% let $e := <wrapper><elem>val</elem></wrapper> return concat(data($e/elem),
 'ue')
>> Query failure: at 1:53-1:80: XPTY0004: In function: fn:concat() with
>> 2 parameters: Expected type 'xs:anyAtomicType?' and found type
>> 'xs:untypedAtomic*' (where xs = "http://www.w3.org/2001/XMLSchema")
```

(Note that `data()` is a special function whose return type is dependent on the type that is passed to it—it always returns values of the same cardinality that was passed in.)

Here, the compiler knows that `concat()` needs to take a sequence of length zero or one for both arguments. The second argument is a sequence with one element, so it is fine. But `data()` says it will return a sequence of length zero-or-more. However, in this case, we know that `data()` will always return a single element, so we can write:

```
%% let $e := <wrapper><elem>val</elem></wrapper> return concat(exactly-
one(data($e/elem)), 'ue')
>> value
```

Troubleshooting static type system errors

This topic provides some guidance for avoiding XQuery static type errors.

Declare return types for functions

If a function is declared without a return type, its return type is assumed to be `item()*`. Usually, this is not useful, so you should specify a more specific return type.

Use the type `element(*, xs:untyped)` for prototypes and quick projects

The type `element(*, xs:untyped)` is useful for passing XML in a function element or return value. In addition, using an element name instead of `*` in the first argument to the `element()` type (that is, `element(<name>, xs:untyped)`) allows the typechecking system to verify that you are using XML data as you intend to.

The following example illustrates the use of `element(*, xs:untyped)`:

```
let $e-ut as element(*, xs:untyped) := <e a='42'>v</e>
return (
  data($e-ut), (: fn:data() always returns an atomic value, so the result
in this case is of type xs:untypedAtomic :)
  concat(data($e-ut), 'alue'),  (: xs:untypedAtomic can be used in any
function call where an atomic value is required :)
  data($e-ut/@a) div 6 (: operators are function calls, too.  Note that
pathing down preserves untypedness :)
)
```

Note: In terms of performance, using untyped variables is slower than using specific types.

Use `fn:doc() treat as`

`fn:doc()` is of type `document-node()?`. You can use `document-node()?` in a path, so it is possible to use `fn:doc($uri)/my/data`. However, `fn:data(doc($uri)/my/data)` will cause a static type error, because the path expression does not have a typed value. You can solve this by using `(fn:doc($uri) treat as document-node(element(*, xs:untyped))` in place of an ordinary call to `fn:doc()`.

Do not use `element()` as a type

It is not a good idea to use `element()` as a type. `element()` is equivalent to `element(*, xs:anyType nillable)`. Instead, use `xs:untyped`.

`xs:anyType` and `xs:untyped` both provide no type information about the contents of an element. However, marking an element as containing `xs:untyped` data makes the data eligible for extraction with `fn:data`. Any node, including one that contains `xs:anyType`, can have its data extracted as a string with `fn:string`, but `fn:data` can directly return values in the appropriate types such as doubles and integers.

Consider using `fn:exactly-one()` on path expressions

Consider the following example:

```
let $e as element(x, xs:untyped) := <x><y>21</y></x>
return exactly-one(data($e/y)) + 21
```

An element might have more than one child named `y`. Because `$e/y` has the type `element(y, xs:untyped)*`, `fn:data($e/y)` has the type `xs:untypedAtomic*`. The + operator requires both of its operands to have a cardinality of `exactly-one`. To eliminate the `*`, a function call to `fn:exact¬ly-one()` is required.

fn:doc() behavior

The `fn:doc()` function retrieves an external document identified by a URI. In order to help system administrators manage possible security threats, Endeca has implemented a stepped enforcement scheme to control its use.

The Dgraph flag `--xquery_fndoc` has three possible values that specify the handling of the `fn:doc()` function within XQuery.

The following values are supported:

- `none:` Causes all calls to `fn:doc()` to fail. This is the default if `--xquery_fndoc` is not provided.
- `sandbox:` Allows `fn:doc()`, but controls the environment within which it is used. When you call `fn:doc()`, its argument is always interpreted as a relative path to the `xml` subdirectory of the directory specified with the Dgraph flag `--xquery_path`. The path is screened for ".." steps, so if the function tries to go outside of the `xml` subdirectory, `fn:doc()` fails.
- `open:` At the `open` level, arguments to `fn:doc()` are interpreted as URLs and given to a URL resolver. The URL is not restricted to the local host or local domain. The `open` setting may impact security and performance significantly.

Note: At this time, `fn:doc()` with a setting of `open` is not supported for use in production.

Using fn:doc() with document nodes

Regardless of the Dgraph flag setting, `fn:doc()` always works when used to retrieve documents stored at any URI starting with `mdex://documents`.

fn:error() behavior

The Endeca implementation of `fn:error()` signals an error by throwing an exception.

If an exception is thrown during the evaluation of an XQuery main module, either from the MDEX Engine or by an explicit call to `fn:error()`, and that exception is not caught in the main module, an HTML response, with the details of the exception, is returned, with a status code of `500: Internal Server Error`.

As defined in the XQuery spec, the `fn:error()` function can take zero, one, two, or three optional arguments. In the case of XQuery for Endeca, the third argument in the three-argument version is used to specify the error sequence. If an error raised by `fn:error()` is not caught, the contents of the third argument, if any, are serialized and returned as the body of the HTTP response. In the case of SOAP faults generated by the MDEX Web service, this appears in the `ErrorSequence` element of the `mdata:Fault`.

Note: MAX API functions do not use the three-argument form of `fn:error()`, so if the XQuery code passed in the request to the MDEX Web service does not itself call the three-argument form of `fn:error()`, the `ErrorSequence` element in the SOAP Fault will always be empty.

fn:trace() behavior

The `fn:trace()` function is useful for debugging XQuery modules. It wraps an expression and writes its value to an implementation-defined log. In the Dgraph, output generated by calls to `fn:trace()` is sent to the Dgraph log file, or to `stdout`, if the Dgraph is run from a shell.

`fn:trace()` takes two arguments: the expression you are tracing and a label. The XQuery evaluator serializes the value of the expression (that is, turns a typed value into a string) and returns the same value again, so you can transparently annotate your query.

The following example illustrates `fn:trace()` usage:

```
trace(<parent><child1/><child2/></parent>/*, "my_label")
dgraph log output:
Request: 1 - trace(<child1/><child2/>, my_label)
```

Note: An XQuery processor is only required to evaluate as much of a query as is necessary to compute the final result. That means that unused sub-expressions are not guaranteed to be evaluated, so their values may not be directed to the log. For more information, see the topic "Endeca XQuery evaluator expression-skipping behavior."

fn:replace() behavior

Endeca's implementation of `fn:replace()` does not handle subexpression references in the replacement string exactly as specified in the XQuery specification.

In particular, it fails if it encounters a reference for which there is no corresponding subexpression in the pattern, instead of replacing it by an empty string. Endeca's implementation of `fn:replace()` therefore imposes several constraints, to confine usage of subexpression references to those which will yield correct results according to XQuery. The following are not permitted:

1. Multiple-digit subexpression references.
2. References to non-existent subexpressions.
3. Digits immediately following a subexpression.

xs:integer precision details

The Endeca implementation of the type `xs:integer` limits values to those representable by 64-bit integers.

xs:decimal precision details

The Endeca implementation of the type `xs:decimal` limits the precision of values to 18 digits.

xs:string length limitation

The length of `xs:string` is limited by available memory.

System response to integer overflow

In the event of integer overflow, XQuery for Endeca raises an error (`FOAR0002`).

Default ordering mode for an empty sequence

The default ordering mode for an empty sequence is `empty greatest`, which means that when sorting, an empty sequence is considered the largest of all.

Treatment of external functions

If an external function is found, it is guaranteed either to succeed or to raise an error as defined in the specification.

If an external function is not found, it raises an `err:XPST0017 "function not found"` error, and causes the compilation of the module containing the call to that function to fail.

Treatment of options

The Endeca implementation of XQuery does not recognize any option declarations, as defined by the XQuery specification.

Treatment of extra digits in numeric operations

Extra digits in numeric operations are rounded according to IEEE round-to-nearest, except in the case of decimals.

Decimals are rounded away from zero.

Treatment of collation

The Endeca implementation of XQuery supports only default collation, as defined in the XQuery specification.

The default collation is the Unicode code point collation.

Treatment of illegal characters in XML

The Endeca implementation of XQuery raises an error (`SERE0003`) if the result of a query includes XML nodes, and the serialization of these nodes would generate characters that are not legal within XML documents.

For details on illegal characters, see `http://www.w3.org/TR/REC-xml/#charsets`.

Casting strings to xs:decimal

If there are too many decimal digits when casting a string to `xs:decimal`, XQuery for Endeca truncates them.

XQuery try/catch expressions

This topic describes how XQuery try/catch is implemented in XQuery for Endeca.

The Endeca XQuery implementation supports the try/catch facility specified in XQuery 1.1 Working Draft 3 (*http://www.w3.org/TR/xquery-11*). Try/catch expressions provide a mechanism for handling errors raised during the evaluation of an XQuery main module.

Try/catch simplifies the development of Web services in XQuery by enabling the author to handle application errors appropriately in the service. For example, an XQuery main module implementing a SOAP Web service might contain an outermost try/catch to intercept any errors raised during evaluation of the service and produce a response containing an appropriate SOAP Fault.

Errors that are unhandled by a service, and thus escape the XQuery evaluator, are treated as follows:

- The HTTP return code is set to 500.
- If the error is associated with a non-empty error sequence (via the third argument to the fn:error function), the response body is computed by serializing the error sequence.
- Otherwise, an HTML body is constructed containing the error code (QName), a description, and any associated details about the error.

The interaction of XQuery try/catch expressions with updating expressions

When try/catch expressions are used with the Endeca implementation of XQuery update, if exceptions are raised in a try block, any updates appended within that try block are removed from the pending update list. This rollback is applied regardless of whether the exception is caught at that point, caught further up the stream, or escapes the program.

Optimizations and error message details

This section provides information about the behavior of optimizations and error messages in the Endeca XQuery evaluator.

Related Links

Impact of optimizations in the Endeca XQuery evaluator on page 47

The Endeca XQuery evaluator may reorder or change expressions in a way that is guaranteed not to change the computed result of an expression.

Treatment of unused variables in let statements on page 48
Unused variables in `let` statements are pruned completely.

Error message details on page 48
This topic describes the XQuery error message format and points you to additional information about XQuery error handling.

Tips for optimizing XQuery

This topic contains tips for optimizing XQuery.

Note: Several of the examples in this topic are shown in the format used by the command-line developer tool, exquery, that is provided by Endeca. You can use exquery in interactive or file-based mode to test and debug the XQuery functions that you write yourself. More information about exquery appears in the chapter "Getting Started with Web Services and XQuery for Endeca."

- Use the `ep:time` pragma as a profiling tool to find hotspots. Only optimize code that is expensive to execute. However, keep in mind that the `ep:time` pragma may produce unintuitive results because of XQuery optimization. For example, consider the following code:

```
declare namespace ep = "http://www.endeca.com/XQuery/pragmas/2008";
 (#ep:time all#) {
 let $n := (#ep:time seqstuff#) { reverse(1 to 1000000)[1] }
    return (#ep:time add#) { $n + 1 }
 }
```

  which, run in exquery, emits the following:

```
TRACE{seqstuff}: 821.809 ms
 TRACE{add}: 822.199 ms
 TRACE{all}: 822.229 ms
 1000001
```

  Not only does this imply that the `all` block take less time than the sum of its children, but also that creating and reversing a million-element list takes as long as performing addition. The cost of addition is actually trivial, but the XQuery compiler performed `let-inlining` to produce code structured like this:

```
declare namespace ep = "http://www.endeca.com/XQuery/pragmas/2008";
 (#ep:time all#) {
    return (#ep:time add#) { (#ep:time seqstuff#) { reverse(1 to
1000000)[1] } + 1 }  }
```

  To investigate cases where the optimizer may have interfered with the time pragma, you can put the time pragma around larger blocks of code to look out for inconsistencies, or use the `!o` modifier in exquery to see what the optimizer does to various kinds of expressions.
- Do not calculate the same value, or walk down the same path, repeatedly in performance-critical code. For example, instead of:

```
<packed>
  <fish>{$x/categories/foodstuffs/[@type = 'fish']}</fish>
  <bread>{$x/categories/foodstuffs/[@type = 'bread']}</bread> </packed>
```

use:

```
 let $foodstuffs := $x/categories/foodstuffs return
 <packed>
   <fish>{$foodstuffs/[@type = 'fish']}</fish>
   <bread>{$foodstuffs/[@type = 'bread']}</bread>
 </packed>
```

This can save a significant amount of time, if `$x` or `$x/categories` or `$x/categories/foodstuffs` has many children. (Common sub-expression elimination has not yet been implemented.)

- The XQuery evaluator is not optimized to work on large results sets. If you bring a large data set into XQuery, it will have to be internally materialized and processed, even if XQuery does not emit a lot of data. Do as much work like filtering and searching as possible in the queries executed by the Dgraph, rather than in XQuery code.
- Because sorting is expensive, you should not use the `order by` clause in FLOWR expressions more than is necessary. In addition, you should minimize the size of the sequence values that are sorted.
- The Endeca XQuery evaluator does not perform tail-call optimization or limit recursion depth. Therefore, recursive functions may execute slowly, and deep recursion may cause a stack overflow.

Impact of optimizations in the Endeca XQuery evaluator

The Endeca XQuery evaluator may reorder or change expressions in a way that is guaranteed not to change the computed result of an expression.

In some cases, evaluation of an expression may be skipped altogether if it does not impact the results. Such optimizations may cause counter-intuitive side effects, such as the printing of items by `fn:trace` and the timing reported by the `ep:time` pragma.

Consider the following example:

```
let $foo := fn:trace(“x”, “y”)
return <nothing/>
```

Since the return value does not depend on the binding of `fn:trace(“x”, “y”)` to the variable `$foo`, the call to `fn:trace` will be elided.

As another example, consider the following expression in which we attempt to time the evaluation of the path expression `$results//DimVal`:

```
let $results := mdex:query(...)
let $dvals := (#ep:time#) {
  $results//DimVal
}
return $dvals[1], $dvals[2]
```

Because the variable `$results` is referenced only once, its value may be inlined at the site of reference as in the following code:

```
let $dvals := (#ep:time#) {
  mdex:query(...)//DimVal
}
return $dvals[1], $dvals[2]
```

This can impact the time reported by the `ep:time` pragma, which will now time both the call to `mdex:query` and the path expression.

Treatment of unused variables in let statements

Unused variables in `let` statements are pruned completely.

Keep this in mind when you are debugging related issues, because it is possible to comment out more than you intend to in the chain of execution.

Error message details

This topic describes the XQuery error message format and points you to additional information about XQuery error handling.

Errors reported by XQuery for Endeca follow standard XQuery coding conventions. For example, the message `err:XPST0017` "function not found" is issued if an external function is missing. This error can be broken down into the following parts:

- `err` indicates that it is an error.
- XP denotes it as an error defined by XPath.
- ST denotes it as a static error, rather than a dynamic error or a type error.
- `0017` is the unique numeric code for this error.

Information about error handling in XQuery can be found here: `http://www.w3.org/TR/2007/REC-xquery-20070123/#errors`.

Namespace for proprietary errors

The namespace used for Endeca-proprietary XQuery errors is:

```
http://www.endeca.com/XQuery/errors/2008
```

About functions

The XQuery language uses functions to query XML data.

Every XQuery function must have a namespace-qualified name and a return type.

A function may also have one or more typed arguments. XQuery supports three kinds of functions: built-in functions, external functions, and user-defined functions.

In addition, the Endeca XQuery implementation allows library modules. Library modules make it easier to group and share related functions, including those you develop yourself.

Related Links

Built-in functions on page 49
The XQuery language defines over 100 built-in functions defined in the XQuery 1.0 and XPath 2.0 Functions and Operators specification.

External functions on page 49
The Endeca XQuery implementation contains numerous external functions for use by application queries. These external functions add Endeca-specific features to the Endeca XQuery implementation.

User-defined functions on page 57
You can create your own functions, declare them in library or main modules, and access them with Web services.

Built-in functions

The XQuery language defines over 100 built-in functions defined in the XQuery 1.0 and XPath 2.0 Functions and Operators specification.

For this release, the Endeca XQuery implementation supports a large subset of this built-in function library.

Note that for the fn:normalize-unicode function, NFC (Unicode Normalization Form C) is the only normalization form supported. That is, the NFKC, NFD, or NFKD normalization forms are not supported.

Unimplemented functions in this release

The built-in functions listed here are not implemented in this release of XQuery for Endeca.

Each function or group of functions is listed with the section of the XQuery 1.0 and XPath 2.0 Functions and Operators specification that refers to it. See *http://www.w3.org/TR/xpath-functions* for more information about these functions.

2.3 fn:nilled

7.4.11 fn:iri-to-uri

7.4.12 fn:escape-html-uri

8.1 fn:resolve-uri

10. Functions and Operators on Durations, Dates and Times

11.1.1 fn:resolve-QName

11.2.5 fn:namespace-uri-for-prefix

11.2.6 fn:in-scope-prefixes

12. Operators on base64Binary and hexBinary

13. Operators on NOTATION

15.5.2 fn:id

15.5.3 fn:idref

16.3 fn:current-dateTime

16.4 fn:current-date

16.5 fn:current-time

16.6 fn:implicit-timezone

External functions

The Endeca XQuery implementation contains numerous external functions for use by application queries. These external functions add Endeca-specific features to the Endeca XQuery implementation.

External functions are not implemented using XQuery syntax, nor are they defined by the XQuery specification. For this reason, a query must include declarations of the external functions it uses, either directly or by importing the library modules that contain the declarations of these functions.

XQuery function declaration syntax

This topic describes the syntax used to declare XQuery functions. It is only necessary to declare external functions and user-defined functions.

To declare a function, use the following syntax:

```
declare function prefix:function-name($parameter-name as parameter-type,
...)
as return-type external;
```

For more information, see *http://www.w3.org/TR/xquery/#FunctionDeclns*.

External function library modules

In order to use the Endeca external functions, you must import the correct library modules, or include equivalent declarations in your query.

These modules are located in `$ENDECA_MDEX_ROOT/lib/xquery/internal`.

- The library module for eutil functions is `eutil.xq`.
- The library module for http functions is `http.xq`.
- The library module for mdex functions, `mdex.xq`.

Important: The modification of these files without the express prior written consent of Endeca is not permitted.

Endeca http functions

The external functions declared in the `http.xq` library module provide access to the HTTP request within XQuery. The names of these functions belong to the `http` namespace.

Note: For related information, see the topic, "About the HTTP URL format."

Declaring and using http functions

To use any of the `http` functions in a module, the import statement that follows, or equivalent declarations, must be included in that module.

```
import module namespace http =
"http://www.endeca.com/XQuery/http/2008" at "http.xq";
```

Http function namespace

This topic provides the namespace URI for `http` functions. The module import statement implicitly declares the namespace.

The namespace for http functions is:

```
http://www.endeca.com/XQuery/http/2008
```

http:get-method()

The `http:get-method()` function returns the HTTP request method as a string, such as "GET" or "POST".

Function Signature	`http:get-method() as xs:string`

Function Summary	The `http:get-method()` function returns the HTTP request method as a string, such as "GET" or "POST". The HTTP request method is generally sent from the HTTP client to the Web server and communicates the purpose of the request.
Parameters	none
Example	`http:get-method()` might return the following: "POST"

http:get-path()
The `http:get-path()` function returns the request path as a string, such as `/ws/myservice`.

Function Signature	`http:get-path() as xs:string`
Function Summary	The `http:get-path()` function returns the request path as a string, such as `/ws/myservice`.
Parameters	none
Examples	Consider an `echo` service. You would use `http://server:1234/ws/echo` to execute the service. Consider a "GET" request to `http://server:1234/ws/echo/some/path/here`. In this case, within the XQuery for the `echo` service, a call to `http:get-path()` would return the string "/ws/echo/some/path/here". In addition, here is an example using query parameters: `http://server:1234/ws/ser¬ vice?param1=val1&param2=val2&param3=val3` In this case, `http:get-path()` would return "/ws/service".

http:get-header(header-name)
The `http:get-header(header-name)` function returns the value of the named HTTP request header as a string, or an empty sequence if the named header was not present in the current request.

Function Signature	`http:get-header($name as xs:string) as xs:string?`
Function Summary	The `http:get-header(header-name)` function returns the value of the named HTTP request header, such as "Content-Type", as a string, or as an empty sequence if the named header was not present in the current request.
Parameters	`$header-name as xs:string` (this is not case sensitive)
Example	`http:get-header("Content-Type")`might return the following: "text/plain"

http:get-header-names()

The `http:get-header-names()` function returns the names of the headers present in the current request as a sequence of strings.

Function Signature	`http:get-header-names() as xs:string*`
Function Summary	The `http:get-header-names()` function returns the names of the headers present in the current request.
Parameters	none
Example	`http:get-header-names()` might return a sequence such as the following: ("Content-Type","Content-Length")

http:get-body()

The `http:get-body()` function returns the HTTP request body as a string, or an empty sequence if no body was provided.

`http:get-body()`, together with `eutil:parse(xml-string)`, can be used to access the SOAP body or other POST body in main modules.

Function Signature	`http:get-body() as xs:string?`
Function Summary	The `http:get-body()` function returns the HTTP request body as a string, or an empty sequence if no body was provided.
Parameters	none
Example	You could use `eutil:parse(fn:exactly-one(http:get-body()))`to get the request body as an XML document.

http:get-id()

The `http:get-id()` function returns the HTTP request ID as a numeric string. This is an internally-generated ID that is unique for each request during a single run of the Dgraph.

Function Signature	`http:get-id() as xs:string`
Function Summary	The `http:get-id()` function returns the HTTP request ID. This is an internally-generated ID that is unique for each request during a single run of the Dgraph.
Parameters	none
Example	`http:get-id()` might return the following: "9700"

http:get-query()

The `http:get-query()` function returns the query part of the URL as a string, or an empty sequence if the URL does not include a query part.

The query portion of an HTTP URL string is formatted as a series of parameter name=value pairs separated by ampersands, such as `name1=value1&name2=value2`. In the example URL that follows, `http://host:port/ws/service?name1=value1&name2=value2#tag`, the query part is `name1=value1&name2=value2`.

Function Signature	`http:get-query() as xs:string?`
Function Summary	The `http:get-query()` function returns the query part of the URL as a string.
Parameters	none
Example	`http:get-query()` returns something similar to `"name1=value1&name2=value2"`.

http:get-query-parameter-names()

The `http:get-query-parameter-names()` function returns the query parameter names as a sequence of strings.

The query portion of an HTTP URL string is formatted as a series of parameter name=value pairs separated by ampersands, such as `n1=v1&n2=v2`. The order of the sequence of names returned by `http:get-query-parameter-names()` is not specified. Each name in the URL appears only once, even if it was repeated multiple times in the URL.

Function Signature	`http:get-query-parameter-names() as xs:string*`
Function Summary	The `http:get-query-parameter-names()` function returns the query parameter names as a sequence of strings.
Parameters	none
Example	`http:get-query-parameter-names()` might return something like `"value1"`.

http:get-query-parameter()

The `http:get-query-parameter()` function returns the values of a query parameter as a sequence of strings.

It is possible to have multiple values associated with one parameter name. These values are always returned in the same order that they appeared in the URL. If the name appears in the query string with no value, (that is, as "name" rather than "name=value"), the corresponding value appears in the value sequence as an empty string. The argument is case sensitive.

Function Signature	`http:get-query-parameter($name as xs:string) as xs:string*`
Function Summary	The `http:get-query-parameter()` function returns the value of a query parameter as a sequence of strings.
Parameters	`$name as xs:string`
Example	`http:get-query-parameter()` might return something like `("value1","value2")`.

Endeca eutil functions

The external functions declared in the `eutil.xq` library modules provide general-purpose extensions to the Endeca XQuery implementation. These functions do not depend on the Dgraph. The names of these functions are bound to the `eutil` namespace.

Declaring and using eutil functions

To use any of the eutil functions, the import statement that follows, or equivalent declarations, must be included in the query.

```
import module namespace eutil =
"http://www.endeca.com/XQuery/eutil/2008" at "eutil.xq";
```

Eutil function namespace

In order to use the `eutil` functions, you have to declare the namespace URI. The module import statement implicitly declares the namespace.

The namespace for `eutil` functions is `http://www.endeca.com/XQuery/eutil/2008`. In addition, the `eutil:get-stack-trace` function requires the stack-trace namespace (`http://www.endeca.com/XQuery/stacktrace/2009`).

eutil:eval(xquery-string)

The `eutil:eval(xquery-string)` function evaluates `xquery-string` as an XQuery main module and returns a result. The evaluation is performed with no context item defined. It throws an exception if `xquery-string` is not a valid main module, or if the evaluation causes a dynamic error.

Important: Because `xquery-string` is compiled each time `eutil:eval()` is called, use of this function can be time-consuming. It was designed for development use, and is not supported for production use.

Function Signature	`eutil:eval($xquery-string as xs:string) as item()*`
Function Summary	The `eutil:eval(xquery-string)` function evaluates `xquery-string` as an XQuery main module and returns a result. It throws an exception if `xquery-string` is not a valid main module, or if the evaluation causes a dynamic error.
Parameters	`$xquery-string as xs:string`
Example	`eutil:eval("1 to 4")` returns the sequence (1, 2, 3, 4)
Errors Thrown	• `endeca-err:EVAL0001` if the input to the function was a library module rather than a main module. • `endeca-err:EVAL0002` if the input to the function was an updating main module. • `endeca-err:EXTF0001` if an unexpected internal error of a different nature occurred. For more details about the errors thrown by external functions, see the topic "Error code listing."

eutil:eval(xquery-string,context-item)

The `eutil:eval(xquery-string,context-item)` function is similar to `eutil:eval(xquery-string)` except that the evaluation is performed with the context item set to the value of the second argument. The static type of the context item is defined to be "item()". The context item is a value that the XQuery evaluator makes available to the query that can be set by the implementation.

Important: Because `xquery-string` is compiled on the fly each time `eutil:eval()` is called, it can be time-consuming. It was designed for development use, and is not supported for production use.

Function Signature	`eutil:eval($xquery-string as xs:string, $context-item as item()) as item()*`
Function Summary	The `eutil:eval(xquery-string,context-item)` function is similar to `eutil:eval(xquery-string)` but in addition sets the context item to the provided item.
Parameters	`$xquery-query as xs:string, $context-item as item()`
Example	`eutil:eval("let $item as node() := . treat as node() return $item/text()", <something>some text</something>)` returns: "some text"
Errors Thrown	• `endeca-err:EVAL0001` if the input to the function was a library module rather than a main module. • `endeca-err:EVAL0002` if the input to the function was an updating main module. • `endeca-err:EXTF0001` if an unexpected internal error of a different nature occurred. For more details about the errors thrown by external functions, see the topic "Error code listing."
Note	The `context-item` type is `item()`, so you need a `treat` expression or something similar to use it as a value with a more specific type.

eutil:parse(xml-string)

The `eutil:parse(xml-string)` function parses `xml-string` and returns the resulting document node. It throws an exception if `xml-string` is either empty or is not well-formed XML.

`eutil:parse(xml-string)` can be used with `http:get-body()` to access the SOAP body or other POST body in main modules.

Function Signature	`eutil:parse($xml-string as xs:string) as document-node(element(*, xs:untyped))`

Function Summary	The `eutil:parse(xml-string)` function parses `xml-string` and returns the resulting document node. It throws an exception if `xml-string` is either empty or is not well-formed XML.
Parameters	`$xml-string as xs:string`
Example	You could use `eutil:parse(fn:exactly-one(http:get-body()))`to get the request body as an XML document.
Errors Thrown	• `endeca-err:PARS0001` if the input to the function was not valid XML. • `endeca-err:EXTF0001` if an unexpected internal error of a different nature occurred. For more details about the errors thrown by external functions, see the topic "Error code listing."

eutil:get-stack-trace()

The `eutil:get-stack-trace()` function returns a sequence of span elements representing the stack trace associated with the current exception. The stack trace, by identifying the expression where an error occurred and the expressions it was called from, makes it easier to debug errors in your XQuery code.

This function should only be used within a catch block in a try/catch expression. That expression should be generating human-readable diagnostic information, such as a SOAP fault. If `eutil:get-stack-trace()` is used outside of a try/catch expression, it returns an empty sequence.

Function Signature	`eutil:get-stack-trace() as element(stack-trace:span, xs:untyped)*`
Function Summary	The `eutil:get-stack-trace()` function returns a sequence of `Span` elements. Each `Span` element contains a `uri` attribute that identifies the file in which the error occurs, as well as elements for the starting and ending location of the error. In a sequence of `Span` elements, the first one is the innermost expression where the error occurred, while any subsequent ones locate enclosing expressions.
Parameters	none

Example	This example shows an embedded stack trace:
	`  <stack-trace:Span uri="%28INTERNAL%29%2Fmdex_internal_dimen¬`
	`sion_search.xq">`
	`          <stack-trace:Start line="20" column="75"/>`
	`          <stack-trace:End line="20" column="143"/>`
	`   </stack-trace:Span>`
	`   <stack-trace:Span uri="%28INTERNAL%29%2Fmdex.xq">`
	`          <stack-trace:Start line="44" column="2"/>`
	`          <stack-trace:End line="44" column="58"/>`
	`    </stack-trace:Span>`
	`    <stack-trace:Span uri="">`
	`          <stack-trace:Start line="2" column="0"/>`
	`          <stack-trace:End line="2" column="136"/>`
	`    </stack-trace:Span>`
	`    <stack-trace:Span uri="%2Flocaldisk%2Fvvaradha%2Fsand¬`

```
box%2FDEV%2Ftrunk%2Flinux-x86_64-release%2Finstall%2Fx¬
query%2Fmdex.xq">
          <stack-trace:Start line="91" column="15"/>
          <stack-trace:End line="91" column="46"/>
    </stack-trace:Span>
    <stack-trace:Span uri="%2Flocaldisk%2Fvvaradha%2Fsand¬
box%2FDEV%2Ftrunk%2Flinux-x86_64-release%2Finstall%2Fx¬
query%2Fmdex.xq">
          <stack-trace:Start line="157" column="4"/>
          <stack-trace:End line="163" column="5"/>
    </stack-trace:Span>
    <stack-trace:Span uri="%2Flocaldisk%2Fvvaradha%2Fsand¬
box%2FDEV%2Ftrunk%2Flinux-x86_64-release%2Finstall%2Fx¬
query%2Fmdex.xq">
          <stack-trace:Start line="175" column="2"/>
          <stack-trace:End line="175" column="24"/>
    </stack-trace:Span>
```

User-defined functions

You can create your own functions, declare them in library or main modules, and access them with Web services.

Note: The creation of XQuery modules, aside from those provided with the product, requires the purchase of the Advanced Query module. Contact your Endeca representative for details.

About pragmas

Pragmas are implementation-specific language extension facilities that are used to provide information, such as additional parameters, to the XQuery evaluator.

Pragmas wrap an expression and have complete control over static type checking and evaluation. The syntax for pragmas is defined in the "Extension Expressions" section of the XQuery specification, at *http://www.w3.org/TR/xquery/#id-extension-expressions*.

Endeca provides several pragmas that were designed to speed application development. All of these pragmas pass through the static type and dynamic value of the enclosed expression unchanged and have side effects related to logging.

Pragma namespace

The pragma namespace must be declared in any query that uses pragmas.

The namespace for all of the supported pragmas is `http://www.endeca.com/XQuery/prag¬ mas/2008`.

ep:emit-dynamic-type

The `ep:emit-dynamic-type` pragma writes out the dynamic type of the enclosed expression.

When evaluated, `ep:emit-dynamic-type` causes the dynamic type of the result of evaluating the enclosed expression to be emitted via the same mechanism as `fn:trace()`. The pragma contents are used as the label for the trace.

The following example of `ep:emit-dynamic-type`:

```
declare namespace ep = "http://www.endeca.com/XQuery/pragmas/2008";
(#ep:emit-dynamic-type my_label#) {
    <parent><child1/><child2/></parent>/*
}
```

would result in the following Dgraph log output:

```
Request: 1 - fn:trace(element child1 , element child2, my_label)
```

ep:emit-static-type

The `ep:emit-static-type` pragma writes out the static type of the enclosed expression.

When evaluated, `ep:emit-static-type` causes the statically computed type of the enclosed expression to be emitted via the same mechanism as `fn:trace()`. The pragma contents are used as the label for the trace.

The following example of `ep:emit-static-type`:

```
declare namespace ep = "http://www.endeca.com/XQuery/pragmas/2008";
(#ep:emit-static-type my_label#) {
    <parent><child1/><child2/></parent>/*
}
```

would result in the following Dgraph log output:

```
Request: 1 - fn:trace(( element * of type xs:untyped | none )*, my_label)
```

ep:stats-timing

The `ep:stats-timing` pragma tracks the time in milliseconds used to evaluate the enclosed expression and writes this information in list form to the MDEX Engine Statistics page. This pragma makes it easier to determine what part of a Web services query is impacting performance.

The `ep:stats-timing` pragma collects most expensive query statistics for MAX. If you are using MAX functions, the most expensive MAX queries are collected automatically.

The `ep:stats-timing` pragma can also be used to collect custom timing data. In such cases, you use the pragma to create your own lists. The custom timing list appears below the most expensive MAX invocation list in the XQuery Server Statistics section on the MDEX Engine Statistics page.

When evaluated, `ep:stats-timing` causes the amount of time spent evaluating the enclosed expression to be emitted via the same mechanism as `fn:trace()`. The pragma contents are used as the label for the trace. The `ep:stats-timing` pragma is similar to `ep:time`, except that `ep:stats-timing` puts its output into a list.

The format of the `ep:stats-timing` pragma is as follows:

```
(#ep:stats-timing my_list#)
{
   "identifying expression",
   <parent><child1/><child2/></parent>
}
```

The `ep:stats-timing` pragma must take a sequence expression, joined by a comma, as the enclosed expression.

- `my_list` is the name of the list to add to.
- `"identifying expression"` is the expression to use to identify the evaluation of the expression.
- `<parent><child1/><child2/></parent>` is the expression to time and evaluate.

If the expression inside the `ep:stats-timing` pragma is not a sequence, an `endeca-err:STAT0001` error is thrown.

The following example uses the `ep:stats-timing` pragma to keep track of the top ten most expensive calls to a function.

```
declare namespace ep = "http://www.endeca.com/XQuery/pragmas/2008";
declare function local:example ($input as xs:string)
as xs:string
{
   (#ep:stats-timing Example Function List#)
   {
      $input,
      concat("Hello, ",$input, "!")
   }
}
```

This pragma would add an entry to a list entitled Example Function List, with an ID matching the string value of `$input`, and a value equal to the time it took to evaluate `concat("Hello, ",$input, "!")`.

ep:time

The `ep:time` pragma writes out the time in milliseconds used to evaluate the enclosed expression. The `ep:time` pragma writes this information to both the Dgraph log and the MDEX Engine Statistics page.

When evaluated, `ep:time` causes the amount of time spent evaluating the enclosed expression to be emitted via the same mechanism as `fn:trace()`. The pragma contents are used as the label for the trace.

The `ep:stats-timing` pragma is similar to `ep:time`, except that `ep:stats-timing` puts its output into a list.

The following example of `ep:time`:

```
declare namespace ep = "http://www.endeca.com/XQuery/pragmas/2008";
(#ep:time my_label#) {
    <parent><child1/><child2/></parent>/*
}
```

might result in the following Dgraph log file output:

```
Request: 1 - fn:trace(1.008 ms, my_label)
```

Note: For information about how optimization can make using `ep:time` confusing, see the topic "Tips for optimizing XQuery."

Working with Web services in the MDEX Engine

This section contains details about using Web services with your Endeca application.

The HTTP URL format

This topic defines the format used by the MDEX Engine for Web services URLs.

This format is based on the standard URL format of `http://<host>:<port>/<path>`. In addition to the required path, an MDEX Engine Web services URL can include optional query parameters and fragment identifiers.

For example, in the URL `http://host:port/ws/service?n1=v1&n2=v2#tag`:

- `service` is a service name that corresponds to an XQuery main module. Service names are case sensitive.
- `n1=v1&n2=v2` is the optional query part of the URL. These parameters may be specified after the `<path>` element by appending a `?` followed by parameters. Parameters are `<name>=<value>` pairs, separated by ampersands `(&)`.
- `tag` is the optional fragment identifier part of the URL.

EBNF for the URL format

The following EBNF grammar describes the syntax for the URL format.

The EBNF uses the same Basic EBNF notation as the W3C specification of XML, located here: *http://www.w3.org/TR/xml/#sec-notation*.

```
<URL> ::= "http://" <host> ":" <port> <path> ["?" <query>] ["#" <fragment-
id>]

<path> ::= "/" <prefix> ["/" <service-name> ["/" <path-step-list>]]

<prefix> ::= "ws"

<path-step-list> ::= <path-step-name> ["/" <path-step-list>]

<query> ::= <query-parameter-list>

<query-parameter-list> ::= <query-parameter> ["&" <query-parameter-list>]

<query-parameter> ::= <query-parameter-name> ["=" <query-parameter-value>]
```

Note these important items about the syntax:

- Reserved items in this syntax include terms used by the grammar, such as `ws`.
- If `<path> == <prefix>`, then the list operation is executed to list currently available services.
- All quoted strings are case-insensitive.
- If `<path>` includes a service name, the corresponding XQuery main module will be evaluated. If there is no corresponding main module, then a 404 Not Found message is returned.
- If `<path>` does not include a service name, an administrative operation identified by the query parameter `op` will be performed.

Viewing a list of available Web services

The following command returns a list of available Web services modules:

```
http://<host>:<port>/ws
```

About URL paths

URL paths are used to invoke various HTTP operations.

There are two kinds of HTTP operations:

- Administrative operations
- Configuration operations

Note: For details on these operations, see the *Oracle Endeca Guided Search Administrator's Guide*.

Web service request and response headers

This topic describes how Web services and XQuery for Endeca handles request and response headers.

For POST requests, the request body is always converted to a string. The client can specify any encoding supported by ICU, using the charset field of the Content-Type header. Depending upon the situation, the following behavior occurs:

- If there is no Content-Type header, or the Content-Type header doesn't specify the charset, the encoding defaults to iso-8859-1, as required by the HTTP spec.
- If a specified encoding is not recognized, an Unsupported Media Type status is returned, with an HTML body describing the error.
- If the encoding is recognized, but the conversion to string fails, a Bad Request status is returned, with an HTML body describing the error.
- If a Content-Type header is present, but its value cannot be parsed (including the case where the media type part is not of the form `"<mime-type>/<mime-subtype>"`), an Unsupported Media Type status is returned, with an HTML body explaining the error.
- The media type specified in the request Content-Type header has no effect on request processing, except in so far as the Web service itself depends on the value of this header.

Within the Web service, the request body can be accessed as an `xs:string` using `http:get-body()`, and this can be parsed as XML using `eutil:parse()`. The value of the Content-Type header, if present, can be accessed using `http:get-header("Content-Type")`. No functions are provided to access the media type and charset fields separately. For GET requests, `http:get-body()` returns an empty sequence.

Web service text encoding

All Web service HTTP response bodies are strings encoded as UTF-8, with a Content-Type of "text/xml; charset=UTF-8".

The string is only well-formed XML if the result of evaluating the XQuery main module is a single element node, or a document node with a single element node child. The element node can itself have arbitrary child and descendent nodes.

Main and library modules loaded from disk files must be UTF-8. Files loaded using `fn:doc()` in `sandbox` mode must be UTF-8. POST bodies with Content-Type "charset=UTF-8" must be valid

UTF-8. Validation errors during module load generate warnings, but do not prevent other modules from being loaded, or the MDEX Engine from starting up. UTF-8 validation errors during request processing cause the request to fail with non-OK status.

Dgraph Web service clients that use non-ASCII characters in their URLs (whether in service names or in query parameter names or values) must first convert these characters to UTF-8 and then URL encode the separate UTF-8 bytes to form the actual URL. When a Web service request is received, the path and the fragment ID are URL decoded. If any of these generates invalid UTF-8, the request fails with a Bad Request status, and a warning is logged. In addition, all query parameter names and values are URL decoded. If the name or value of any query parameter cannot be URL decoded to valid UTF-8, that name-value pair is ignored, and a warning is written to the error log.

WSDL support

There is no explicit support for WSDL in Web services and XQuery for Endeca.

However, it is possible to create a Web service that returns a WSDL document, instead of the normal Web service result, based on the query parameters in the request. To do so, you can embed the WSDL text directly in the Web service XQuery code.

In the example that follows, the WSDL is embedded in the XQuery main module:

```
import module namespace http = "http://www.endeca.com/XQuery/http/2008" at
 "http.xq";
import module namespace eutil = "http://www.endeca.com/XQuery/eutil/2008"
at "eutil.xq";
import module namespace mdex = "http://www.endeca.com/XQuery/mdex/2008" at
 "mdex.xq";

declare namespace soap = "http://schemas.xmlsoap.org/soap/envelope/";
declare namespace mdata = "http://www.endeca.com/MDEX/data/IR600";
declare namespace ex = "http://endeca.com/example.xsd";

declare function local:generateFaultResponse($message as xs:string*) as
element(soap:Envelope, xs:untyped)
{
   <soap:Envelope>
      <soap:Body>
         <soap:Fault>
            <mdata:query-fault>
               <mdata:fault-string>{$message}</mdata:fault-string>
            </mdata:query-fault>
         </soap:Fault>
      </soap:Body>
   </soap:Envelope>
};

declare function local:generateWsdl()
{
<definitions name="ExampleMDEXQuery"
             targetNamespace="http://endeca.com/example.wsdl"
             xmlns:es="http://endeca.com/example.wsdl"
             xmlns:esxsd="http://endeca.com/example.xsd"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    <xsd:schema
      targetNamespace="http://endeca.com/example.xsd"
      elementFormDefault="qualified"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">

      <!-- request contains just a word to be matched -->
      <xsd:element name="MatchWord">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="word" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <!-- response contains just the number of matches -->
      <xsd:element name="MatchCount">
        <xsd:complexType>
          <xsd:all>
            <xsd:element name="count" type="xsd:int"/>
          </xsd:all>
        </xsd:complexType>
      </xsd:element>

    </xsd:schema>
  </types>

  <!-- request message uses an element from the above schema -->
  <message name="CountMatchesInput">
    <part name="word" element="esxsd:MatchWord"/>
  </message>

  <!-- response message uses an element from the above schema -->
  <message name="CountMatchesOutput">
    <part name="count" element="esxsd:MatchCount"/>
  </message>

  <!-- port type specifies the above request and response message -->
  <portType name="ExampleMDEXPortType">
    <operation name="CountMatches">
      <input message="es:CountMatchesInput"/>
      <output message="es:CountMatchesOutput"/>
    </operation>
  </portType>

  <!-- binding specifies the above port type and declares this web service
 to
       be of the document/literal variety -->
  <binding name="ExampleMDEXSoapBinding" type="es:ExampleMDEXPortType">
    <soap:binding style="document" transport="http://schemas.xml¬
soap.org/soap/http"/>
    <operation name="CountMatches">
      <soap:operation soapAction="http://endeca.com/Example"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>

  <!-- associate the above port type and binding -->
  <service name="ExampleMDEXService">
```

```
    <documentation>Endeca Example MDEX Query Service</documentation>
    <port name="ExampleMDEXPortType" binding="es:ExampleMDEXSoapBinding">
      <soap:address location="http://endeca.com/mdex"/>
    </port>
  </service>

</definitions>
};


(: get the request :)
let $body-str := fn:trace(http:get-body(), "REQUEST")
return
  if (fn:exists(http:get-query-parameter("WSDL"))) then
    (: caller used ?WSDL request format -- send WSDL :)
    local:generateWsdl()
  else if (fn:empty($body-str)) then
    (: no request body -- send SOAP fault :)
    local:generateFaultResponse("Empty body in SOAP request")
  else
    (: reach into the SOAP request body :)
    let $body := eutil:parse(fn:exactly-one($body-str))/soap:Enve¬
lope/soap:Body
    (: pull out the input message :)
    let $query := $body/ex:MatchWord
    return
      if (fn:count($query) ne 1) then
        (: should be exactly one input message :)
        local:generateFaultResponse($query)
      else
        (: start making a response enveleope :)
    fn:trace(
        <soap:Envelope>
          <soap:Body>
            {
            (: setup an MDEX API call to do the text search :)
            let $str := fn:data($query/ex:word)
            let $cl :=
              <mdata:Query>
                <mdata:Searches>
                  <mdata:Search Key = "English">{$str}</mdata:Search>
                </mdata:Searches>
              </mdata:Query>
    (: call MDEX API :)
    let $result := mdex:navigation-query($cl)
            (: fill in an output message :)
            return
                <ex:MatchCount>
                  <ex:count>
                  {
        fn:data($result/mdata:RecordsResult/@TotalRecordCount)
                  }
                  </ex:count>
                </ex:MatchCount>
            }
          </soap:Body>
        </soap:Envelope>, "RESPONSE")
```

About request IDs

When the Dgraph is operating in Web services mode, it maintains a sequence number that starts with the value 1 each time the Dgraph is started. Each request that is received gets the next sequential value appended to its URL. These numbers are known as request IDs or HTTP Exchange IDs.

The request ID is copied in each MDEX Engine request log entry, in certain tables in the MDEX Engine statistics page, and in certain other sources of logging information, to help correlate this information with the associated request.

Coordinating logging details for Web services invocations

For Web services invocations, logging and debugging information about queries needs to be gathered from multiple sources. You can use the request ID to correlate information across logs.

The MDEX Engine provides several tools for tracking and monitoring engine and query performance, including the following:

- The MDEX Engine request log (also called the Dgraph request log) captures query information, which you can use to analyze application performance.
- The MDEX Engine Statistics page displays MDEX Engine (Dgraph) performance statistics. It provides a detailed breakdown of what the Dgraph is doing, and is a useful source of information about your Endeca implementation's configuration and performance.
- The Dgraph error log tracks errors at the engine level.

When Web services invocations are used in the MDEX Engine, requests are logged and statistics generated just as they are for Presentation API queries. However, collecting query details for Web services invocations is a multi-part process, because the Web services URL contains only the service name. The bulk of the query is contained in the POST body. To make it easier to correlate queries across logs for Web services invocations, the MDEX Engine appends a request ID to each query.

For example, if you wanted to gather more information about the most expensive queries in your application, you would need to look at both the request log and the statistics page. On the MDEX Engine Statistics page, the Most Expensive Queries tab lists query URLs by service name and request ID, as in the example `/ws/myservice:57`. This request ID corresponds to the HTTP Exchange ID in the MDEX Engine request log. By looking for HTTP Exchange ID 57, you could retrieve additional information about the contents of the query from the request log's Query Body field.

The same HTTP Exchange ID is used by the Dgraph error log and so can be used to track errors as well.

For detailed information about using both the MDEX Engine Statistics page and the MDEX Engine Request Log, see the *Performance Tuning Guide*.

Error code listing

External functions in the Endeca implementation of XQuery raise useful errors. You can use try/catch expressions to handle such errors.

In almost all cases, the actual errors thrown provide additional detail describing the specific condition that led to the error. For example, referencing a dimension named Sales that does not exist would lead to an MDEX0001 error with the detail message "Dimension 'Sales' does not exist."

Errors use the prefix `endeca-err` and the URI `http://www.endeca.com/XQuery/errors/2009`.

Note: This table does not list the errors you might encounter if you attempt to use internal-only interfaces. Such use is not supported.

Code	Message	When Thrown	Meaning
SCHM0001	`"Schema validation failed"`	This error can be thrown from any of the following functions: • `mdex-data:put-record` • `mdex-data:put-dimension-value` • `mdex:navigation-query` • `mdex:dimension-search-query` • `mdex:compound-dimension-search-query` • `mdex:record-details-query` • `mdex:aggregate-record-details-query`	The input to a function did not pass schema validation.
PARS0001	`"It is a dynamic error if the argument to eutil:parse cannot be parsed as XML."`	This error is thrown from the `eutil:parse` function.	The input to `eutil:parse` was not valid XML.
MDEX0001	`"Invalid input"`	This error can be thrown from any of the following functions: • `mdex-data:put-record` • `mdex-data:put-dimension-value` • `mdex:get-record` • `mdex-data:delete-record` • `mdex:navigation-query` • `mdex:dimension-search-query` • `mdex:compound-dimension-search-query` • `mdex:record-details-query` • `mdex:aggregate-record-details-query`	The input to a function was contextually invalid—that is, it passed schema validation, but failed some further logical validation. For example, a non-existent dimension name was specified, or a non-existent record spec was provided.

Code	Message	When Thrown	Meaning
EVAL0001	`"It is a dynamic error if the query argument to eutil:eval compiles to a library module."`	This error is thrown from the `eutil:eval` function.	The input to `eutil:eval` was a library module (as opposed to a main module).
EVAL0002	`"eutil:eval cannot evaluate an updating main module."`	This error is thrown from the `eutil:eval` function.	The input to `eutil:eval` was an updating main module.
STAT0001	`"It is a static error if the expression enclosed within a stats-timing pragma is not a sequence expression."`	This error is thrown at compilation time.	The expression inside a `stats-timing` pragma was not a sequence.
EXTF0001	`"External function error."`	This error can be thrown from any external function.	An unexpected internal error occurred in an external function. This error is thrown when more specific errors listed above do not apply.

Chapter 4

About the MDEX API through XQuery

The MDEX API through XQuery (or MAX) exposes MDEX Engine features in XQuery.

MDEX API through XQuery details

MAX is easy to use natively in XQuery, allowing you to perform common tasks for library building such as combining the results of several queries together, building several related queries for a single click, restructuring results, and so on.

MAX contains a set of XQuery functions that execute queries of different types. There are six query types:

- `navigation`
- `dimension-search`
- `compound-dimension-search`
- `record-details`
- `aggregate-record-details`
- `metadata`

The naming convention used is `<query-type>-query`. These functions take, as input, a document that represents a query. The input schema for each query type defines the structure of these input documents.These functions return another document representing the results of the query. The output schemas define the structure of these documents.

MDEX API through XQuery naming scheme

The naming scheme used for the data types in MAX provides some insight into the relationships between elements.

All elements used in MAX are in the namespace `http://www.endeca.com/MDEX/data/IR600`. In examples, this is usually assigned the prefix `mdata`.

The input of each query function (except `mdex:metadata-query()`) is an element with local name `Query` that conforms to a data type ending in `Query`. For example, the input of `mdex:navigation-query` is an element named `Query` that conforms to the `NavigationQuery` data type.

The result of each function is an element with local name `<QueryType>Results` that conforms to a data type ending in `Result`. For example, the result of `mdex:navigation-query` is an element named `NavigationResults` that conforms to the `NavigationResult` data type.

Collections of similar elements are contained in an outer element with a plural name. For example, to specify which dimension values to select in a navigation query, you create a `SelectedDimension¬ValueIds` element that contain a `DimensionValueId` element for each dimension value you are selecting. For example:

```
<Query xmlns="http://www.endeca.com/MDEX/data/IR600">
   <SelectedDimensionValueIds>
      <DimensionValueId>8026</DimensionValueId>
   </SelectedDimensionValueIds>
</Query>
```

To find out which filters were applied from the results of a query, look under the `AppliedFilters` element. For example:

```
let $numFiltersApplied := fn:count($results/mdata:AppliedFilters/*)
```

Any data type ending in `List` can only contain elements of the same type (or that are extensions of the same type). In addition, all elements that conform to a type ending in `List` have a plural name. They may also contain attributes that relate to all of the elements that they contain.

In the following example, the records in a navigation query are contained in an element named `Records`, which conforms to the data type `RecordList`. This element contains only elements of the type `Record`.

```
for $record in $results/mdata:RecordsResult/mdata:Records/*
return
   $record/@Id
```

Note that in the example above, you could use either `*` or `mdata:Record` after `mdata:Records`, because all children of the `Records` element can be treated as a record.

All attributes of a dimension value and properties on a record are contained in an element named `Attributes`, which conforms to the data type `AttributeList`. All children of this element can be treated as simple key-value pairs, because an `AttributeDimensionValue` is an extension of `Property`. For example:

```
for $attribute in $record/mdata:Attributes/*
return
   fn:concat($attribute/@Key, "-", fn:data($attribute))
```

Note that the `*` after `mdata:Attributes` will give you both the properties and the dimension values.

Any element with a type ending in `List` will not appear in the results if it is empty, although it may appear if it contains relevant attributes. The element will appear, even if it is empty, if it is a direct child of the `Results` element.

Empty elements with a type ending in `List` are allowed in the input of a function.

MDEX API through XQuery schema location

The MAX installation contains the following schemas.

- `aggregate_record_details_query.xsd`
- `compound_dimension_search_query.xsd`
- `dimension_search_query.xsd`

- `mdex.xsd`
- `navigation_query.xsd`
- `record_details_query.xsd`

The schemas are located in the `$ENDECA_MDEX_ROOT/conf/schema` directory.

Note: This directory also contains other schemas intended for internal use only.

About the internal namespace

Anything in the namespace `internal` is not intended for public consumption and may change from release to release. Direct usage of these features by custom XQuery logic is not supported.

Understanding error messages in the MDEX API through XQuery

This topic explains some MAX error messages.

Passing invalid Dgraph arguments in Web services mode

If your query includes arguments that pass schema validation but are invalid to the Dgraph, you will see the following message:

```
"exception encountered while executing external function 'internal:query',
 caused by Error:[MDEX] There was a problem processing the requested query."
```

This might appear if you specify a rollup key for a dimension that is not enabled for rollup, or if you specify a record ID that does not exist in a RecordDetailsQuery. If you sent the equivalent query in Presentation API mode, you would receive a 404 message.

To get more information about the problem that caused this error, look at the Dgraph error log.

Schema validation errors

When schema validation fails, you may receive a 500 error similar to the following:

```
WARN 09/23/09 15:08:09.949 UTC (1222182489949) DGRAPH {dgraph}: Exchange
38 returned "500 Internal Server Error": exception encountered while execut¬
ing external function 'internal:schema-validate-query-input', caused by
Invalid xml: Element 'SelectedDimensionValueIds' should be qualified
```

In order to obtain more specific information, you can check your query input against the schema. You may want to use a schema validator like the one freely available from W3C at *http://www.w3.org/2001/03/webdata/xsv* to do this.The schemas for MAX, `mdex_internal_*.xq` and `mdex.xq`, are located in the `$ENDECA_MDEX_ROOT/lib/xquery/internal` directory.

About mdex functions

The functions declared in the `mdex.xq` library provide access to the internal API of the MDEX Engine. The names of these functions are bound to the `mdex` namespace.

About declaring and using mdex functions

To use any of the `mdex` functions, the import statement that follows, or equivalent declarations, must be included in the query.

```
import module namespace mdex =
"http://www.endeca.com/XQuery/mdex/2008" at "mdex.xq";
```

Mdex function namespace

In order to use `mdex` functions, you have to declare the namespace URI. The module import statement implicitly declares the namespace.

The namespace declaration for `mdex` functions is the following.

```
declare namespace mdex = "http://www.endeca.com/XQuery/mdex/2008";
```

In addition, you need to declare the namespace for the XML used in arguments and return values for `mdex` functions as follows:

```
declare namespace mdata = 'http://www.endeca.com/MDEX/data/IR600';
```

mdex:add-navigation-descriptors()

The `mdex:add-navigation-descriptors` function allows you to determine the navigation state produced by following a navigation or refinement from an initial state.

This function might be used, for example, to allow refinements to be rendered as unique URL paths in an application.

Function Signature	`mdex:add-navigation-descriptors($currentDescriptors as xs:string*, $refinements as xs:string+) as xs:string*`
Function Summary	Returns the navigation state that would result from starting at the navigation state represented by the dimension values in `$currentDescriptors`, and following the refinements represented by the dimension values in `$refinements`.
Parameters	`$currentDescriptors`: a sequence of one or more dimension value IDs, representing the starting navigation state `$refinements`: a sequence of one or more dimension value IDs, representing the refinements to follow from the starting navigation state.
Return Values	Returns a sequence of dimension value IDs, representing the resulting navigation state.
Example	The following example returns the navigation state that would result from following a refinement for dimension value 10025 from a navigation state in which dimension values 10002 and 18003 were already selected.

	`mdex:add-navigation-descriptors((10002,18003), (10025))`

mdex:aggregate-record-details-query()

The `mdex:aggregate-record-details-query()` function executes an aggregate record details query.

Function Signature	`mdex:aggregate-record-details-query($query as element(mdata:Query, xs:untyped)) as element(mdata:Results, xs:untyped)`
Function Summary	Executes an aggregate record details query.
Parameters	`$query`: An element representing an aggregate record details query. This element should be of type `mdata:AggregateRecordDetailsQuery`.
Return Values	Returns a `Results` element of type `mdata:AggregateRecordDetailsResult` containing the results of the query.
Example	The following example returns the results from an aggregate record details query for the record with ID 10000, rolled up on the property WineType. `mdex:aggregate-record-details-query(<Query Id="10000" AggregationKey="WineType"/>)`
Errors Thrown	• `endeca-err:SCHM0001` if the input to the function did not pass schema validation. • `endeca-err:MDEX0001` if the input to the function contextually invalid. • `endeca-err:EXTF0001` if an unexpected internal error of a different nature occurred. For more details about the errors thrown by external functions, see the topic "Error code listing."

mdex:compound-dimension-search-query()

The `mdex:compound-dimension-search-query()` function executes a compound dimension search query.

Note: Dimension names have to match the NCName syntax rule defined in the XML specification. In particular, there can be no spaces or colons in names. If dimension names are not NCName compliant, Dgidx issues a warning to its log file, but execution is not halted.

Function Signature	`mdex:compound-dimension-search-query($query as element(mdata:Query, xs:untyped)) as element(mdata:Results, xs:untyped)`
Function Summary	Executes a compound dimension search query.
Parameters	`$query`: An element representing a compound dimension search query. This element should be of type `mdata:CompoundDimensionSearchQuery`.
Return Values	Returns a `Results` element of type `mdata:CompoundDimensionSearchResult` containing the results of the query.

Example	The following example returns the results from a compound dimension search query for the term Merlot.
	`mdex:compound-dimension-search-query(`
	`<Query>`
	`   <CompoundDimensionSearch>`
	`      Merlot`
	`   </CompoundDimensionSearch>`
	`</Query>)`

Errors Thrown	• `endeca-err:SCHM0001` if the input to the function did not pass schema validation. • `endeca-err:MDEX0001` if the input to the function contextually invalid. • `endeca-err:EXTF0001` if an unexpected internal error of a different nature occurred. For more details about the errors thrown by external functions, see the topic "Error code listing."

mdex:dimension-search-query()

The `mdex:dimension-search-query()` function executes a dimension search query.

Note: Dimension names have to match the NCName syntax rule defined in the XML specification. In particular, there can be no spaces or colons in names. If dimension names are not NCName compliant, Dgidx issues a warning to its log file, but execution is not halted.

Function Signature	`mdex:dimension-search-query($query as element(mdata:Query, xs:untyped)) as element(mdata:Results, xs:untyped)`
Function Summary	Executes a dimension search query.
Parameters	`$query`: An element representing a dimension search query. This element should be of type `mdata:DimensionSearchQuery`.
Return Values	Returns a `Results` element of type `mdata:DimensionSearchResult` containing the results of the query.

Example	The following example would return the results from a dimension search query for the term Merlot.
	`mdex:dimension-search-query(`
	`<Query>`
	`   <DimensionSearch>`
	`      Merlot`
	`   </DimensionSearch>`
	`</Query>)`

Errors Thrown	• `endeca-err:SCHM0001` if the input to the function did not pass schema validation. • `endeca-err:MDEX0001` if the input to the function contextually invalid. • `endeca-err:EXTF0001` if an unexpected internal error of a different nature occurred. For more details about the errors thrown by external functions, see the topic "Error code listing."

mdex:dimension-value-id-from-path(dimension-value-path)

The `mdex:dimension-value-id-from-path` function takes the path of a dimension value and returns the dimension value ID as a string.

Note: Dimension names have to match the NCName syntax rule defined in the XML specification. In particular, there can be no spaces or colons in names.

Function Signature	`mdex:dimension-value-id-from-path($path as xs:string+) as xs:string`
Function Summary	The `mdex:dimension-value-id-from-path` function is useful when you don't know a precise dimension value ID. It takes a single sequence of string values corresponding to the path to a dimension value, and returns the dimension value ID as a string.
Parameters	`$dimension-value-path as xs:string+`

Examples	**Note:** The `mdata` namespace used below contains all of the elements in the output. The following example would return the dimension value ID for Merlot. `mdex:dimension-value-id-from-path(("Wine_Type","Red","Mer¬lot"))`
Notes	The extra set of parentheses around `$dimension-value-path` is necessary because it represents a single sequence argument, and not string of separate arguments.
Errors Thrown	• `endeca-err:SCHM0001` if the input to the function did not pass schema validation. • `endeca-err:MDEX0001`if the dimension value is not under the specified parent or does not exist. • `endeca-err:EXTF0001` if an unexpected internal error of a different nature occurred.

mdex:metadata-query()

The `mdex:metadata-query` function retrieves static MDEX Engine metadata such as property data, sort keys, search keys, and aggregation keys.

Function Signature	`mdex:metadata-query() as element(mdata:Results, xs:untyped)`
Function Summary	Executes a metadata query.
Parameters	none
Return Values	Returns a `Results` element of type `mdata:MetadataResult` containing static MDEX Engine metadata such as property data, sort keys, search keys, and aggregation keys.
Example	The following example would return the MDEX Engine metadata. `mdex:metadata-query()`

mdex:navigation-query()

The `mdex:navigation-query()` function executes a navigation query.

Function Signature	`mdex:navigation-query($query as element(mdata:Query, xs:untyped)) as element(mdata:Results, xs:untyped)`

Function Summary	Executes a navigation query.
Parameters	`$query`: An element representing a navigation query. This element should be of type `mdata:NavigationQuery`.
Return Values	Returns a Results element of type `mdata:NavigationResult` containing the results of the query.
Example	The following example return the results from a root query. `mdex:navigation-query(<Query/>)`
Errors Thrown	• `endeca-err:SCHM0001` if the input to the function did not pass schema validation. • `endeca-err:MDEX0001` if the input to the function contextually invalid. • `endeca-err:EXTF0001` if an unexpected internal error of a different nature occurred. For more details about the errors thrown by external functions, see the topic "Error code listing."

mdex:record-details-query()

The `mdex:record-details-query()` function executes a record details query.

Function Signature	`mdex:record-details-query($query as element(mdata:Query, xs:untyped)) as element(mdata:Results, xs:untyped)`
Function Summary	Executes a record details query.
Parameters	`$query`: An element representing a record details query. This element should be of type `mdata:RecordDetailsQuery`.
Return Values	Returns a `Results` element of type `mdata:RecordDetailsResult` containing the results of the query.
Example	The following example would return the results from a record details query for the record with ID 10000. `mdex:record-details-query(<Query Id="10000"/>)`
Errors Thrown	• `endeca-err:SCHM0001` if the input to the function did not pass schema validation.

	• `endeca-err:MDEX0001` if the input to the function contextually invalid. • `endeca-err:EXTF0001` if an unexpected internal error of a different nature occurred. For more details about the errors thrown by external functions, see the topic "Error code listing."

MDEX API through XQuery data types

The following data types make up the schema(s).

Keep in mind the following points:

- Any data type that appears in the input can also be found in the output.
- If a `minOccurs` is specified for an element but no `maxOccurs` is specified, the default `maxOccurs` is 1, and not unbounded.

AdjustmentType data type

`AdjustmentType` is a simple type that enumerates the possible adjustments to a search term, such as word-break analysis or spelling correction, made by the MDEX Engine while processing a search.

```
<simpleType name="AdjustmentType">
  <restriction base="string">
    <enumeration value="Phrasing"/>
    <enumeration value="SpellingCorrection"/>
    <enumeration value="WordBreak"/>
  </restriction>
</simpleType>
```

AggregateRecord data type

The `AggregateRecord` data type represents an aggregated record in the MDEX Engine.

An aggregated record represents a collection of one or more records that have been grouped by a property or dimension value, referred to as the aggregation key, or rollup key.

```
<complexType name="AggregateRecord">
  <complexContent>
    <extension base="tns:Record">
      <sequence>
        <element name="DerivedProperties" type="tns:PropertyList" minOc¬
curs="0" maxOccurs="1" />
        <element name="ConstituentRecords" type="tns:RecordList" />
      </sequence>
      <attribute name="RecordsInAggregate" type="unsignedLong" use="re¬
quired"/>
    </extension>
  </complexContent>
</complexType>
```

Attributes:

`RecordsInAggregate`	Number of records in the aggregate record.

AggregateRecordDetailsAppliedFilters data type

The `AggregateRecordDetailsAppliedFilters` data type represents the filters that were applied to the navigation state by the `AggregateRecordDetailsQuery`.

```
<complexType name="AggregateRecordDetailsAppliedFilters">
  <sequence>
    <element name="SelectedDimensionValueIds" type="tns:DimensionValueIdList"
 minOccurs="0" />
    <element name="EqlExpression" type="tns:NonEmptyString" minOccurs="0"
/>
    <element name="RecordFilter" type="tns:NonEmptyString" minOccurs="0"
/>
    <element name="RangeFilters" type="tns:RangeFilterList" minOccurs="0"
/>
  </sequence>
</complexType>
```

AggregateRecordDetailsQuery data type

The `AggregateRecordDetailsQuery` data type contains the elements used by aggregate record queries.

In the list of elements below:

- `EqlExpression` is a string written in the Endeca Query Language (EQL). It cannot be URL-escaped as it is in the Navigation API. For more information about EQL, see the *Advanced Development Guide*.
- `RecordFilter` is an expression that restricts the results of the query.

For details on the other elements, see the corresponding data type entry.

```
<complexType name="AggregateRecordDetailsQuery">
  <all>
    <element name="SelectedDimensionValueIds" type="tns:DimensionValueIdList"
 minOccurs="0" />
    <element name="EqlExpression" type="tns:NonEmptyString" minOccurs="0"
/>
    <element name="RecordFilter" type="tns:NonEmptyString" minOccurs="0"
/>
    <element name="RangeFilters" type="tns:RangeFilterList" minOccurs="0"
/>
    <element name="Sorts" type="tns:SortList" minOccurs="0" />
  </all>
  <attribute name="AggregationKey" type="tns:NonEmptyString" use="required"
 />
  <attribute name="Id" type="tns:NonEmptyString" use="required" />
</complexType>
```

Attributes:

`AggregationKey`	The property upon which the records are aggregated.
`Id`	The ID of the aggregate record.

AggregateRecordDetailsResult data type

The `AggregateRecordDetailsResult` data type represents the results of a query for a single aggregate record retrieved from the MDEX Engine.

```
<complexType name="AggregateRecordDetailsResult">
  <sequence>
    <element name="Dimensions" type="tns:DimensionList" />
    <element name="AggregateRecord" type="tns:AggregateRecord" />
   <element name="AppliedFilters" type="tns:AggregateRecordDetailsApplied¬
Filters" />
  </sequence>
  <attribute name="AggregationKey" type="tns:NonEmptyString" use="required"
 />
</complexType>
```

Attributes:

`AggregationKey`	The property upon which the records are aggregated.

AggregationKey data type

The `AggregationKey` data type represents the name of the property by which records should be aggregated.

```
<complexType name="AggregationKey">
  <simpleContent>
    <extension base="tns:NonEmptyString">
     <attribute name="RecordsPerAggregateRecord" type="tns:RecordsPerAggre¬
gateRecord" />
    </extension>
  </simpleContent>
</complexType>
```

Attributes:

`RecordsPerAggregateRecord`	The number of records returned per aggregated record: all, none, or one.

AggregationKeyList data type

The `AggregationKeyList` data type represents a list of names of the property by which records should be aggregated.

```
<complexType name="AggregationKeyList">
  <sequence>
    <element name="AggregationKey" type="tns:NonEmptyString" minOccurs="1"
 maxOccurs="unbounded" />
  </sequence>
</complexType>
```

AlternativePhrasingMode data type

The `AlternativePhrasingMode` data type enumerates the possible values for alternative phrasing mode.

This indicates whether the MDEX Engine uses one of the alternative phrasings it has computed instead of the end user's original query when computing the set of documents to return.

```
<simpleType name="AlternativePhrasingMode">
  <restriction base="string">
    <enumeration value="ComputeAlternativePhrasing" />
    <enumeration value="RewriteWithAlternativePhrasing" />
    <enumeration value="Disabled" />
  </restriction>
</simpleType>
```

AnalyticsResult data type

The `AnalyticsResult` data type represents the Analytics results returned by the MDEX Engine.

An `AnalyticsResult` element contains `AnalyticsStatementResult` elements if an `Analytic¬sExpression` was used. There is one `AnalyticsStatementResult` per Analytics statement in the request.

```
<complexType name="AnalyticsResult">
  <sequence>
   <element name="AnalyticsStatementResult" type="tns:AnalyticsStatementRe¬
sult" minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>
```

Note: When an Analytics statement is syntactically invalid, a dynamic error is raised within XQuery.

AnalyticsStatementResult data type

The `AnalyticsStatementResult` data type represents the business rule results returned by the MDEX Engine.

Each `AnalyticsStatementResult` represents an Analytics table. There is one per Analytics statement. AnalyticsStatementResult elements contain records in the same formation as records in navigation.

```
<complexType name="AnalyticsStatementResult">
  <complexContent>
    <extension base="tns:RecordList">
      <attribute name="Name" type="tns:NonEmptyString"  use="required" />
     <attribute name="TotalRecordCount" type="unsignedLong" use="required"
 />
    </extension>
  </complexContent>
</complexType>
```

Attributes:

`Name`	The name of the result, as specified by the Analytics statement. In the example `RETURN "Best Wineries" AS`, "Best Wineries" is the value of the `Name` attribute.
`TotalRecordCount`	The total record count.

AttributeDimensionValue data type

The `AttributeDimensionValue` data type represents a dimension value.

```
<complexType name="AttributeDimensionValue">
  <simpleContent>
    <extension base="tns:Property">
      <attribute name="DimensionId" type="tns:DimensionValueId" use="re¬
quired" />
      <attribute name="Id" type="tns:DimensionValueId" use="required" />
    </extension>
  </simpleContent>
</complexType>
```

Attributes:

`DimensionId`	The ID of the dimension this dimension value belongs to.
`Id`	The ID of the dimension value itself.

AttributeList data type

The `AttributeList` data type is used to encapsulate dimension values and property values.

```
<complexType name="AttributeList">
  <choice minOccurs="1" maxOccurs="unbounded">
   <element name="AssignedDimensionValue" type="tns:AttributeDimensionValue"
 />
    <element name="Property" type="tns:Property" />
  </choice>
</complexType>
```

AttributeMetadata data type

The `AttributeMetadata` data type represents metadata describing a property's type and whether it can be searched, sorted, or can act as a aggregation key.

```
<complexType name="AttributeMetadata">
  <sequence>
   <element name="Properties" type="tns:PropertyList" minOccurs="0" maxOc¬
curs="1" />
  </sequence>
  <attribute name="Name" type="tns:NonEmptyString" use="required" />
</complexType>
```

Attributes:

`Name`	The name of the property that the metadata applies to.

AttributeMetadataList data type

The `AttributeMetadataList` data type represents static metadata about dimensions and properties.

```
<complexType name="AttributeMetadataList">
  <sequence>
    <element name="AttributeMetadata" type="tns:AttributeMetadata" minOc¬
curs="1" maxOccurs="unbounded" />
```

```
  </sequence>
</complexType>
```

BetweenFilter data type

The `BetweenFilter` data type represents the conditions that can be set on a "between" range filter.

The two bounds in `BetweenFilter` are non-inclusive, and can be used with geocode properties.

```
<complexType name="BetweenFilter">
  <complexContent>
    <extension base="tns:RangeFilter">
      <sequence>
        <element name="GeocodeReference" type="tns:Geocode" minOccurs="0"
maxOccurs="1" />
      </sequence>
      <attribute name="LowerBound" type="double" />
      <attribute name="UpperBound" type="double" />
    </extension>
  </complexContent>
</complexType>
```

Attributes:

`LowerBound`	The lower bound of the filter.
`UpperBound`	The upper bound of the filter.

BusinessRule data type

The `BusinessRule` data type is a result data type representing a dynamic business rule. A business rule allows certain records to be promoted and displayed to users as they search and navigate within a data set.

```
<complexType name="BusinessRule">
  <sequence>
      <element name="SelectedDimensionValueIds" type="tns:DimensionValueI¬
dList" minOccurs="0" maxOccurs="1" />
      <element name="Properties" type="tns:PropertyList" minOccurs="0"
maxOccurs="1" />
      <element name="Records" type="tns:RecordList" minOccurs="0" maxOc¬
curs="1" />
  </sequence>
  <attribute name="Id" type="string" use="required" />
  <attribute name="NavigationStateRecordCount" type="unsignedLong"
use="required" />
  <attribute name="SortAttribute" type="string" use="optional" />
  <attribute name="Style" type="string" use="required" />
  <attribute name="Title" type="string" use="required" />
  <attribute name="Zone" type="string" use="required" />
  <attribute name="SortDirection" type="tns:SortDirection" use="optional"
/>
</complexType>
```

Attributes:

`Id`	The rule ID.

`NavigationStateRecordCount`	The number of records in the navigation state.
`SortAttribute`	Attribute upon which to sort rules (typically name or priority).
`Style`	The name of the rule's style.
`Title`	The rule's title.
`Zone`	The rule's zone.
`SortDirection`	Ascending or Descending.

BusinessRuleList data type

The `BusinessRuleList` data type represents a collection of dynamic business rules.

```
<complexType name="BusinessRuleList">
  <sequence>
    <element name="BusinessRule" type="tns:BusinessRule" minOccurs="1"
maxOccurs="1" />
  </sequence>
</complexType>
```

BusinessRulePreviewTime data type

The `BusinessRulePreviewTime` data type represents the preview time for a business rule time trigger.

```
<simpleType name="BusinessRulePreviewTime">
  <restriction base="dateTime">
   <pattern value="[0-9]{4}-[0-9]{2}-[0-9]{2}T[0-9]{2}:[0-9]{2}:[0-9]{2}"/>

  </restriction>
</simpleType>
```

BusinessRulesResult data type

The `BusinessRulesResult` data type represents the business rule results returned by the MDEX Engine.

```
<complexType name="BusinessRulesResult">
  <sequence>
    <element name="BusinessRulesFilter" type="tns:NonEmptyString" minOc¬
curs="0" />
    <element name="BusinessRulesPreviewTime" type="dateTime" minOccurs="0"
 />
   <element name="BusinessRules" type="tns:BusinessRuleList" minOccurs="0"
 maxOccurs="unbounded" />
  </sequence>
</complexType>
```

CompoundDimensionSearch data type

The `CompoundDimensionSearch` data type represents the search terms used to perform a keyword search and retrieve dimension values with matching names.

```
<complexType name="CompoundDimensionSearch">
  <simpleContent>
    <extension base="tns:NonEmptyString">
      <attribute name="Mode" type="tns:SearchMode" />
    </extension>
  </simpleContent>
</complexType>
```

Attributes:

`Mode`	The search mode, one of: All, AllAny, AllPartial, Any, Boolean, Partial, PartialMax, or Unknown.

CompoundDimensionSearchAppliedFilters data type

The `CompoundDimensionSearchAppliedFilters` data type represents the query properties defined by the `CompoundDimensionSearchQuery`.

```
<complexType name="CompoundDimensionSearchAppliedFilters">
  <sequence>
    <element name="EqlExpression" type="tns:NonEmptyString" minOccurs="0"
/>
    <element name="RangeFilters" type="tns:RangeFilterList" minOccurs="0"
/>
    <element name="RecordFilter" type="tns:NonEmptyString" minOccurs="0"
/>
    <element name="SearchReport" type="tns:SearchReport" minOccurs="0" />
   <element name="SearchWithinDimensionValueIds" type="tns:DimensionValueI¬
dList" minOccurs="0" />
   <element name="SelectedDimensionValueIds" type="tns:DimensionValueIdList"
 minOccurs="0" />
    <element name="LanguageId" type="tns:NonEmptyString" minOccurs="0" />
  </sequence>
</complexType>
```

CompoundDimensionSearchQuery data type

The `CompoundDimensionSearchQuery` data type contains the elements used by compound dimension search queries.

In the list of elements below:

- `DimensionValuesPerDimension` is the number of dimension value results to return per dimension. This is useful when the number of matching dimension values is high and you want to limit it.
- `EqlExpression` is a string written in the Endeca Query Language (EQL). It cannot be URL-escaped as it is in the Navigation API. For more information about EQL, see the *Advanced Development Guide*.
- `RecordFilter` is an expression that restricts the results of the query.
- `SearchWithinDimensionValueIds` is a list of the IDs of the dimension values within which you want to limit your search. For `DimensionSearchQuery`, this is a single value, whereas for `CompoundDimensionSearchQuery`, it is a list of values.
- `LanguageId` is a string that allows per-query specification of the language to parse search terms in.

For details on the other elements, see the corresponding data type entry.

```
<complexType name="CompoundDimensionSearchQuery">
  <all>
    <element name="DimensionValuesPerDimension" type="unsignedLong" minOc¬
curs="0" />
    <element name="EqlExpression" type="tns:NonEmptyString" minOccurs="0"
/>
    <element name="RangeFilters" type="tns:RangeFilterList" minOccurs="0"
/>
    <element name="RecordFilter" type="tns:NonEmptyString" minOccurs="0"
/>
    <element name="CompoundDimensionSearch" type="tns:CompoundDimension¬
Search" />
   <element name="SearchWithinDimensionValueIds" type="tns:DimensionValueI¬
dList" minOccurs="0" />
   <element name="SelectedDimensionValueIds" type="tns:DimensionValueIdList"
 minOccurs="0" />
    <element name="LanguageId" type="tns:NonEmptyString" minOccurs="0" />
  </all>
</complexType>
```

CompoundDimensionSearchResult data type

The `CompoundDimensionSearchResult` data type represents the results of a compound dimension search query.

```
<complexType name="CompoundDimensionSearchResult">
  <sequence>
    <element name="Dimensions" type="tns:DimensionList" />
   <element name="MatchingCompoundDimensionsResult" type="tns:MatchingCom¬
poundDimensionsResult" />
   <element name="AppliedFilters" type="tns:CompoundDimensionSearchApplied¬
Filters" />
  </sequence>
</complexType>
```

CompoundDimensionValueList data type

The `CompoundDimensionValueList` data type represents a list of dimension values associated with a result from a compound dimension search query.

```
<complexType name="CompoundDimensionValueList">
  <sequence>
   <element name="MatchingCompoundDimensionValue" type="tns:MatchingDimen¬
sionValueList" minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>
```

Dimension data type

The `Dimension` data type is a core component of a navigation state query. It does not represent the full dimension—just the portion of it that is relevant to the query.

```
<complexType name="Dimension">
  <sequence>
    <element name="DimensionValue" type="tns:DimensionValue" />
  </sequence>
  <attribute name="Name" type="string" use="required" />
  <attribute name="Id" type="tns:DimensionValueId" use="required" />
  <attribute name="MultiSelect" type="tns:MultiSelect" use="required" />
  <attribute name="GroupName" type="string" use="optional" />
</complexType>
```

Attributes:

`Name`	The name of the dimension value.
`Id`	The dimension value ID.
`MultiSelect`	Tagging a dimension as multi-select.
`GroupName`	The name of the dimension group that this dimension is part of, if any.

DimensionList data type

The `DimensionList` data type represents a list of dimensions.

```
<complexType name="DimensionList">
  <sequence>
    <element name="Dimension" type="tns:Dimension" minOccurs="0" maxOc¬
curs="unbounded" />
  </sequence>
</complexType>
```

DimensionSearch data type

The `DimensionSearch` data type represents the search terms that can be used to perform a keyword search and retrieve dimension values with matching names.

```
<complexType name="DimensionSearch">
  <simpleContent>
    <extension base="tns:NonEmptyString">
      <attribute name="RelevanceRankingStrategy" type="tns:NonEmptyString"
 />
      <attribute name="Mode" type="tns:SearchMode" />
    </extension>
  </simpleContent>
</complexType>
```

Attributes:

`RelevanceRankingStrategy`	The relevance ranking strategy, composed of one or more pre-defined relevance ranking modules.
`Mode`	The search mode, one of: All, AllAny, AllPartial, Any, Boolean, Partial, PartialMax, or Unknown.

DimensionSearchQuery data type

The `DimensionSearchQuery` data type can be used to perform a keyword search and retrieve dimension values with matching names.

At a minimum, in order to execute a dimension search, the query has to be provided with search terms. Additionally, `DimensionSearchQuery` has a number of properties that provide an application greater control over the dimension values that are returned. For example, results may be limited to dimension values from a single dimension by providing the ID of the dimension in the `SearchWithinDimen¬sionValueId` element or limited to a specified number of dimensions by providing a list of dimension IDs in the `SearchWithinDimensionValueIds` element.

In the list of elements below:

- `AggregationKey` is the property or dimension to use as the rollup key for aggregated records in a dimension search query.
- `RefinementConfigs` represents a dynamic refinement configuration for a dimension value. This will determine how refinements are computed under this dimension value.
- `DimensionValuesPerDimension` is the number of dimension value results to return per dimension. This is useful when the number of matching dimension values is high and you want to limit it.
- `EqlExpression` is a string written in the Endeca Query Language (EQL). It cannot be URL-escaped as it is in the Navigation API. For more information about EQL, see the *Advanced Development Guide*.
- `RecordFilter` is an expression that restricts the results of the query.
- `SearchWithinDimensionValueId` is the ID of the dimension within which you want to limit your search. The ID of the root dimension may be used. This is a single value.
- `SearchWithinDimensionValueIds` is a list of dimension IDs that limit your search to the dimensions you specify.
- `LanguageId` is a string that allows per-query specification of the language to parse search terms in.

For details on the other elements, see the corresponding data type entry.

```
<complexType name="DimensionSearchQuery">
  <all>
    <element name="AggregationKey" type="tns:NonEmptyString" minOccurs="0"
 />
    <element name="RefinementConfigs" type="tns:DimensionSearchRefinement¬
ConfigList" minOccurs="0" />
    <element name="DimensionValuesPerDimension" type="unsignedLong" minOc¬
curs="0" />
    <element name="EqlExpression" type="tns:NonEmptyString" minOccurs="0"
/>
    <element name="RangeFilters" type="tns:RangeFilterList" minOccurs="0"
/>
    <element name="RecordFilter" type="tns:NonEmptyString" minOccurs="0"
/>
    <element name="DimensionSearch" type="tns:DimensionSearch" />
   <element name="SearchWithinDimensionValueId" type="tns:DimensionValueId"
 minOccurs="0" />
   <element name="SearchWithinDimensionValueIds" type="tns:DimensionValueI¬
dList" minOccurs="0" />
   <element name="SelectedDimensionValueIds" type="tns:DimensionValueIdList"
 minOccurs="0" />
    <element name="LanguageId" type="tns:NonEmptyString" minOccurs="0" />
  </all>
</complexType>
```

DimensionSearchAppliedFilters data type

The `DimensionSearchAppliedFilters` data type represents filters that were applied to the navigation state defined by the `DimensionSearchQuery`.

```
<complexType name="DimensionSearchAppliedFilters">
  <sequence>
    <element name="EqlExpression" type="tns:NonEmptyString" minOccurs="0"
/>
    <element name="RangeFilters" type="tns:RangeFilterList" minOccurs="0"
/>
    <element name="RecordFilter" type="tns:NonEmptyString" minOccurs="0"
/>
    <element name="SearchReport" type="tns:SearchReport" />
   <element name="SearchWithinDimensionValueId" type="tns:DimensionValueId"
 minOccurs="0" />
   <element name="SelectedDimensionValueIds" type="tns:DimensionValueIdList"
 minOccurs="0" />
    <element name="LanguageId" type="tns:NonEmptyString" minOccurs="0" />
  </sequence>
</complexType>
```

DimensionSearchRefinementConfig data type

The `DimensionSearchRefinementConfig` data type can be used to set dimension search options. In particular, this data type sets whether to return refinement counts for a specified dimension Id.

In the list of elements below:

- `DimensionValueId` represents the unique identifier for a dimension value.
- `ShowCounts` indicates whether to return refinement counts for a given `DimensionValueId`.

For details on the other elements, see the corresponding data type entry.

```
<complexType name="DimensionSearchRefinementConfig">
  <attribute name="DimensionValueId" type="tns:DimensionValueId" use="re¬
quired" />
  <attribute name="ShowCounts" type="boolean" use="optional" />
</complexType>
```

DimensionSearchRefinementConfigList data type

The `DimensionSearchRefinementConfigList` data type can be used to create a list of any number of `DimensionSearchRefinementConfig` types in a dimension search.

For details on the other elements, see the corresponding data type entry.

```
<complexType name="DimensionSearchRefinementConfigList">
  <sequence>
   <element name="RefinementConfig" type="tns:DimensionSearchRefinementCon¬
fig" minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>
```

DimensionSearchResult data type

The `DimensionSearchResult` data type represents results for a dimension search query.

```
<complexType name="DimensionSearchResult">
  <sequence>
    <element name="Dimensions" type="tns:DimensionList" />
   <element name="MatchingDimensionsResult" type="tns:MatchingDimensionsRe¬
sult" />
   <element name="AppliedFilters" type="tns:DimensionSearchAppliedFilters"
 />
  </sequence>
</complexType>
```

DimensionState data type

The `DimensionState` data type represents the state of the dimension and is related to navigation.

```
<complexType name="DimensionState">
  <sequence>
    <element name="Refinements" type="tns:RefinementList" minOccurs="1"
maxOccurs="unbounded" />
    <element name="SelectedDimensionValues" type="tns:DimensionValueS¬
tateList" minOccurs="0" maxOccurs="1" />
    <element name="ImplicitDimensionValues" type="tns:DimensionValueS¬
tateList" minOccurs="0" maxOccurs="1" />
  </sequence>
  <attribute name="DimensionName" type="string" use="required" />
  <attribute name="DimensionId" type="tns:DimensionValueId" use="required"
 />
</complexType>
```

Attributes:

`DimensionName`	The name of a dimension.
`DimensionId`	The ID for a dimension.

DimensionStateList data type

The `DimensionStateList` data type contains a list of dimension states.

```
<complexType name="DimensionStateList">
  <sequence>
    <element name="DimensionState" type="tns:DimensionState" minOccurs="1"
 maxOccurs="unbounded" />
  </sequence>
</complexType>
```

DimensionValue data type

The `DimensionValue` data type represents the lowest-level building block that composes dimensions, navigations, and classifications of records.

`DimensionValue` is a node in the dimension tree. The `DimensionValues` child element is a list of the child dimension values, and the `PropertyList` child element is a list of properties of this dimension value.

```
<complexType name="DimensionValue">
  <sequence>
    <element name="DimensionValues" type="tns:DimensionValueList" minOc¬
```

```
curs="0" maxOccurs="1" />
    <element name="Properties" type="tns:PropertyList" minOccurs="0" maxOc¬
curs="1" />
  </sequence>
  <attribute name="Name" type="string" use="required" />
  <attribute name="Id" type="tns:DimensionValueId" use="required" />
  <attribute name="IsLeaf" type="boolean" use="required" />
  <attribute name="IsNavigable" type="boolean" use="required" />
  <attribute name="StaticRecordCount" type="unsignedLong" use="optional"
/>
</complexType>
```

Attributes:

`Name`	The name of the dimension value.
`Id`	The ID of the dimension value.
`IsLeaf`	Whether this is a leaf dimension value.
`IsNavigable`	Whether this is a navigable dimension value.
`StaticRecordCount`	The total number of records that this dimension value is assigned to. This number is not always returned, and does not change based on the navigation state.

DimensionValueId data type

`DimensionValueId` is a simple type representing the unique identifier for the dimension value.

```
<simpleType name="DimensionValueId">
  <restriction base="string">
    <pattern value="[0-9]+" />
  </restriction>
</simpleType>
```

DimensionValueIdList data type

The `DimensionValueIdList` data type contains a list of dimension value IDs.

Each element this data type contains is the ID of a dimension value.

```
<complexType name="DimensionValueIdList">
  <sequence>
    <element name="DimensionValueId" type="tns:DimensionValueId" minOc¬
curs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>
```

DimensionValueList data type

The `DimensionValueList` data type represents a list of dimension values.

```
<complexType name="DimensionValueList">
  <sequence>
    <element name="DimensionValue" type="tns:DimensionValue" minOccurs="1"
 maxOccurs="unbounded" />
```

```
  </sequence>
</complexType>
```

DimensionValueState data type

The `DimensionValueState` data type represents the actual dimension value in the dimension tree, and can be used to display the dimension value or look it up. This type is used on results.

```
<complexType name="DimensionValueState">
  <attribute name="Name" type="string" use="required" />
  <attribute name="Id" type="tns:DimensionValueId" use="required" />
</complexType>
```

Attributes:

`Name`	The name of the dimension value.
`Id`	The ID of the dimension value.

DimensionValueStateList data type

The `DimensionValueStateList` data type contains a list of dimension value states.

```
<complexType name="DimensionValueStateList">
  <sequence>
    <element name="DimensionValue" type="tns:DimensionValueState" minOc¬
curs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>
```

DimensionValueStratum data type

The `DimensionValueStratum` data type represents the assignment of a dimension value to a specific stratum for sorting for the dimension boost and bury feature.

Dimension values are sorted by their assigned strata and then by whatever the refinement sorting method for the dimension is.

```
<complexType name="DimensionValueStratum">
  <attribute name="DimensionValueId" type="tns:DimensionValueId" use="re¬
quired" />
  <attribute name="Stratum" type="int" use="required" />
</complexType>
```

Attributes:

`DimensionValueId`	Specifies the ID of the dimension value to be assigned to a stratum.
`Stratum`	An integer that represents the stratum to which the dimension value is assigned. A positive integer indicates that the dimension value will be boosted, while a negative integer indicates that the dimension value will be buried.

DimensionValueStratumList data type

The `DimensionValueStratumList` data type contains a list of stratified dimension values.

```
<complexType name="DimensionValueStratumList">
  <sequence>
    <element name="DimensionValueStratum" type="tns:DimensionValueStratum"
 minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>
```

DisabledRefinementsConfig data type

The `DisabledRefinementsConfig` data type represents a configuration for disabled refinements. It specifies which top-level filters (such as selected dimensions, text searches, range filters, EQL filters or range filters) should be included in the base navigation state. The MDEX Engine uses base and default navigation states to compute disabled refinements.

```
<complexType name="DisabledRefinementsConfig">
    <sequence>
      <element name="BaseDimensionIds" type="tns:DimensionValueIdList"
minOccurs="0" maxOccurs="1" />
    </sequence>
    <attribute name="EqlFilterInBase" type="boolean" use="optional" de¬
fault="false"/>
    <attribute name="TextSearchInBase" type="boolean" use="optional" de¬
fault="false"/>
    <attribute name="RangeFiltersInBase" type="boolean" use="optional" de¬
fault="false"/>
  </complexType>
```

Attributes:

`BaseDimensionIds`	A list of dimension IDs to be included into the base navigation state.
`EqlFilterInBase`	Whether to include EQL filters into the base navigation state.
`TextSearchInBase`	Whether to include text searches into the base navigation state.
`RangeFiltersinBase`	Whether to include range filters into the base navigation state.

Geocode data type

The `Geocode` data type contains the longitude and latitude values that represent a geocode property.

```
<complexType name="Geocode">
  <attribute name="Latitude" type="double" use="required" />
  <attribute name="Longitude" type="double" use="required" />
</complexType>
```

Attribute:

`Latitude`	The latitude of the location in whole and fractional degrees. Positive values indicate north latitude and negative values indicate south latitude.
`Longitude`	The longitude of the location in whole and fractional degrees. Positive values indicate east longitude, and negative values indicate west longitude.

GreaterThanFilter data type

The `GreaterThanFilter` data type represents one of the conditions that can be set on a range filter.

This data type contains the non-inclusive lower bound of the filter, plus an optional `GeocodeReference` element for geocode properties.

```
<complexType name="GreaterThanFilter">
  <complexContent>
    <extension base="tns:RangeFilter">
      <sequence>
        <element name="GeocodeReference" type="tns:Geocode" minOccurs="0" 
maxOccurs="1" />
      </sequence>
      <attribute name="LowerBound" type="double" />
    </extension>
  </complexContent>
</complexType>
```

Attributes:

`LowerBound`	The lower bound of the range filter.

GreaterThanOrEqualFilter data type

The `GreaterThanOrEqualFilter` data type represents one of the conditions that can be set on a range filter.

This data type contains the inclusive lower bound of the filter. Unlike `GreaterThanFilter`, it cannot be used with geocode properties.

```
<complexType name="GreaterThanOrEqualFilter">
  <complexContent>
    <extension base="tns:RangeFilter">
      <attribute name="LowerBound" type="double" />
    </extension>
  </complexContent>
</complexType>
```

Attributes:

`LowerBound`	The lower bound of the range filter.

IncludedRecordAttributeList data type

The `IncludedRecordAttributeList` data type represents a list of included record attributes.

```
<complexType name="IncludedRecordAttributeList">
  <sequence>
    <element name="IncludedRecordAttribute" type="tns:NonEmptyString" 
minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>
```

KeywordRedirectList data type

The `KeywordRedirectList` data type contains a list of search terms used to trigger a keyword redirect to a defined URL.

```
<complexType name="KeywordRedirectList">
  <sequence>
   <element name="KeywordRedirect" type="string" minOccurs="0" maxOccurs="un¬
bounded" />
  </sequence>
</complexType>
```

LessThanFilter data type

The `LessThanFilter` data type represents one of the conditions that can be set on a range filter.

This data type contains the non-inclusive upper bound of the filter plus an optional `GeocodeReference` element for geocode properties.

```
<complexType name="LessThanFilter">
  <complexContent>
    <extension base="tns:RangeFilter">
      <sequence>
        <element name="GeocodeReference" type="tns:Geocode" minOccurs="0"
maxOccurs="1" />
      </sequence>
      <attribute name="UpperBound" type="double" />
    </extension>
  </complexContent>
</complexType>
```

Attributes:

`UpperBound`	The upper bound of the range filter.

LessThanOrEqualFilter data type

The `LessThanOrEqualFilter` data type represents one of the conditions that can be set on a range filter.

This data type contains the inclusive upper bound of the filter. Unlike `LessThanFilter`, it cannot be used with geocode properties.

```
<complexType name="LessThanOrEqualFilter">
  <complexContent>
    <extension base="tns:RangeFilter">
      <attribute name="UpperBound" type="double" />
    </extension>
  </complexContent>
</complexType>
```

Attributes:

`UpperBound`	The upper bound of the range filter.

MatchingCompoundDimensionsResult data type

The `MatchingCompoundDimensionsResult` data type represents the results of a query for a compound dimension result retrieved from the MDEX Engine.

```
<complexType name="MatchingCompoundDimensionsResult">
  <sequence>
    <element name="DimensionValuesPerDimension" type="unsignedLong" minOc¬
curs="0" />
   <element name="MatchingCompoundDimensionValues" type="tns:CompoundDimen¬
sionValueList" />
  </sequence>
  <attribute name="HasMore" type="boolean" use="required" />
</complexType>
```

Attributes:

`HasMore`	Indicates whether there are more results that are not shown.

MatchingDimensionsResult data type

The `MatchingDimensionsResult` data type represents the results of a query for a dimension result retrieved from the MDEX Engine.

```
<complexType name="MatchingDimensionsResult">
  <sequence>
    <element name="DimensionValuesPerDimension" type="unsignedLong" minOc¬
curs="0" />
    <element name="MatchingDimensions" type="tns:ResultDimensionList"
minOccurs="0" maxOccurs="1" />
  </sequence>
  <attribute name="HasMore" type="boolean" use="required" />
</complexType>
```

Attributes:

`HasMore`	Indicates whether there are more results that are not shown.

MatchingDimensionValueList data type

The `MatchingDimensionValueList` data type represents a list of matching dimension values.

```
<complexType name="MatchingDimensionValueList">
  <sequence>
   <element name="MatchingDimensionValue" type="tns:AttributeDimensionValue"
 minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>
```

MetadataResult data type

The `MetadataResult` data type represents static MDEX Engine metadata such as property and dimension data, sort keys, search keys, and aggregation keys.

```
<complexType name="MetadataResult">
  <sequence>
```

```
    <element name="SearchKeys" type="tns:SearchKeyList" />
    <element name="SortKeys" type="tns:SortKeyList" />
    <element name="AggregationKeys" type="tns:AggregationKeyList" />
    <element name="AttributeMetadataList" type="tns:AttributeMetadataList"
 />
  </sequence>
</complexType>
```

MultiSelect data type

The `MultiSelect` data type is an enumeration of possible multi-select values.

If a dimension is multi-select enabled, end users can select more than one dimension value for that dimension.

```
<simpleType name="MultiSelect">
  <restriction base="string">
    <enumeration value="And" />
    <enumeration value="Or" />
    <enumeration value="None" />
  </restriction>
</simpleType>
```

NavigationAppliedFilters data type

The `NavigationAppliedFilters` data type represents the set of filters applied to a navigation state.

```
<complexType name="NavigationAppliedFilters">
  <sequence>
    <element name="EqlExpression" type="tns:NonEmptyString" minOccurs="0"
maxOccurs="1" />
   <element name="RangeFilterList" type="tns:RangeFilterList" minOccurs="0"
 maxOccurs="1" />
    <element name="RecordFilter" type="tns:NonEmptyString" minOccurs="0"
maxOccurs="1" />
   <element name="SearchReports" type="tns:SearchReportList" minOccurs="0"
 maxOccurs="1" />
   <element name="SelectedDimensionValueIds" type="tns:DimensionValueIdList"
 minOccurs="0" maxOccurs="1" />
    <element name="LanguageId" type="tns:NonEmptyString" minOccurs="0" />
  </sequence>
</complexType>
```

NavigationQuery data type

The `NavigationQuery` data type contains the elements, such as `RecordFilter` and `RangeFilter`, used by navigation queries.

In the list of elements below:

- `RecordOffset` is the offset in your record list.
- `RecordsPerPage` is the number of records to return.
- `BusinessRulesFilter` is a filter for your business rules. Dynamic business rule filters allow an Endeca application to define arbitrary subsets of dynamic business rules and restrict merchandising results to only the records that can be promoted by these subsets.

- `EqlExpression` is a string written in the Endeca Query Language (EQL). It cannot be URL-escaped as it is in the Navigation API. For more information about EQL, see the *Advanced Development Guide*.
- `RecordFilter` is an expression that restricts the results of the query.
- `UserProfile` is a string that identifies a condition that makes a dynamic business rule fire.
- `AnalyticsExpression` is a string in the Analytics language.
- `LanguageId` is a string that allows per-query specification of the language to parse search terms in.
- `DimensionValueStrata` is a list of stratified dimension values

For details on the other elements, see the corresponding data type entry.

```
<complexType name="NavigationQuery">
  <all minOccurs="0">
    <element name="SelectedDimensionValueIds" type="tns:DimensionValueIdList"
 minOccurs="0" />
    <element name="RefinementConfigs" type="tns:RefinementConfigList"
minOccurs="0" />
    <element name="AggregationKey" type="tns:AggregationKey" minOccurs="0"
 />
    <element name="RecordOffset" type="unsignedLong" minOccurs="0" />
    <element name="RecordsPerPage" type="unsignedLong" minOccurs="0" />
    <element name="IncludedRecordAttributes" type="tns:IncludedRecordAt¬
tributeList" minOccurs="0" />
    <element name="Sorts" type="tns:SortList" minOccurs="0" />
    <element name="BusinessRulesFilter" type="tns:NonEmptyString" minOc¬
curs="0" />
    <element name="BusinessRulesPreviewTime" type="tns:BusinessRulePreview¬
Time" minOccurs="0" />
    <element name="EqlExpression" type="tns:NonEmptyString" minOccurs="0"
/>
    <element name="RecordFilter" type="tns:NonEmptyString" minOccurs="0"
/>
    <element name="RangeFilters" type="tns:RangeFilterList" minOccurs="0"
/>
    <element name="Searches" type="tns:SearchList" minOccurs="0" />
    <element name="AnalyticsExpression" type="tns:NonEmptyString" minOc¬
curs="0" />
    <element name="UserProfiles" type="tns:UserProfileList" minOccurs="0"
/>
    <element name="LanguageId" type="tns:NonEmptyString" minOccurs="0" />
    <element name="DimensionValueStrata" type="tns:DimensionValueStratumList"
 minOccurs="0"/>
  </all>
  <attribute name="AlternativePhrasingMode" type="tns:AlternativePhrasing¬
Mode" use="optional" />
</complexType>
```

Attributes:

`AlternativePhrasingMode`	The `AlternativePhrasingMode` data type enumerates the possible values for alternative phrasing mode. It indicates whether the MDEX Engine uses one of the alternative phrasings it has computed instead of the end user's original query when computing the set of documents to return.

NavigationResult data type

The NavigationResult data type represents the results of a NavigationQuery.

```
<complexType name="NavigationResult">
  <sequence>
    <element name="Dimensions" type="tns:DimensionList" />
   <element name="NavigationStatesResult" type="tns:NavigationStatesResult"
 />
    <element name="RecordsResult" type="tns:RecordsResult" />
    <element name="AppliedFilters" type="tns:NavigationAppliedFilters" />
    <element name="BusinessRulesResult" type="tns:BusinessRulesResult" />
    <element name="KeywordRedirects" type="tns:KeywordRedirectList" />
    <element name="AnalyticsResult" type="tns:AnalyticsResult"/>
  </sequence>
</complexType>
```

NavigationStatesResult data type

The `NavigationStatesResult` data type represents the results of a query for a navigation state result retrieved from the MDEX Engine.

```
<complexType name="NavigationStatesResult">
  <sequence>
    <element name="RefinementConfigs" type="tns:RefinementConfigList"
minOccurs="0" maxOccurs="1" />
    <element name="DimensionStates" type="tns:DimensionStateList" minOc¬
curs="0" maxOccurs="1" />
  </sequence>
</complexType>
```

NonEmptyString data type

The `NonEmptyString` data type is a simple type.

```
<simpleType name="NonEmptyString">
  <restriction base="string">
    <minLength value="1"/>
  </restriction>
</simpleType>
```

PropertyList data type

The `PropertyList` data type contains a list of property values.

```
<complexType name="PropertyList">
  <sequence>
   <element name="Property" type="tns:Property" minOccurs="1" maxOccurs="un¬
bounded" />
  </sequence>
</complexType>
```

Property data type

The `Property` data type represents an Endeca property.

```
<complexType name="Property">
  <simpleContent>
    <extension base="string">
      <attribute name="Key" type="tns:NonEmptyString" use="required" />
    </extension>
  </simpleContent>
</complexType>
```

Attributes:

`Key`	The name of the property.

RangeFilter data type

The `RangeFilter` data type represents a range filter on a navigation record set. A filter is composed of a record property name and a set of conditions that have to be true in order for a record to pass through the filter.

All range filters extend from the `RangeFilter` data type, so they all include the `AttributeName` attribute. `AttributeName` is the name of the property or dimension value you want to filter on.

```
<complexType name="RangeFilter">
  <attribute name="AttributeName" type="string" use="required" />
</complexType>
```

Attributes:

`AttributeName`	The record property or dimension value name.

RangeFilterList data type

The `RangeFilterList` data type represents a collection of range filters.

This data type contains some number of typed range filters, but cannot contain the base `RangeFilter`.

```
<complexType name="RangeFilterList">
  <sequence>
    <choice minOccurs="1" maxOccurs="unbounded">
      <element name="LessThanFilter" type="tns:LessThanFilter" />
     <element name="LessThanOrEqualFilter" type="tns:LessThanOrEqualFilter"
 />
      <element name="GreaterThanFilter" type="tns:GreaterThanFilter" />
     <element name="GreaterThanOrEqualFilter" type="tns:GreaterThanOrEqual¬
Filter" />
      <element name="BetweenFilter" type="tns:BetweenFilter" />
    </choice>
  </sequence>
</complexType>
```

Record data type

The `Record` data type represents an individual Endeca record from the data set stored in an MDEX Engine.

```
<complexType name="Record">
  <sequence>
```

```
    <element name="Attributes" type="tns:AttributeList" minOccurs="0" max¬
Occurs="1" />
    <element name="Snippets" type="tns:SnippetList" minOccurs="0" maxOc¬
curs="1" />
  </sequence>
  <attribute name="Id" type="string" use="required" />
</complexType>
```

Attributes:

`Id`	The record ID.

RecordDetailsAppliedFilters data type

The `RecordDetailsAppliedFilters` data type represents a record filter on a record details query.

```
<complexType name="RecordDetailsAppliedFilters">
  <sequence>
    <element name="RecordFilter" type="tns:NonEmptyString" minOccurs="0"
/>
  </sequence>
</complexType>
```

RecordDetailsQuery data type

The `RecordDetailsQuery` data type represents the record details for a single record from the MDEX Engine.

This data type contains the record spec of the record in question. The optional `RecordFilter` element is an expression that restricts the results of the query.

```
<complexType name="RecordDetailsQuery">
  <all>
    <element name="RecordFilter" type="tns:NonEmptyString" minOccurs="0"
/>
  </all>
  <attribute name="Id" type="tns:NonEmptyString" use="required" />
</complexType>
```

Attributes:

`Id`	The record ID.

RecordDetailsResult data type

The `RecordDetailsResult` data type represents the results of a query for a record details result retrieved from the MDEX Engine.

```
<complexType name="RecordDetailsResult">
  <sequence>
    <element name="Dimensions" type="tns:DimensionList" />
    <element name="Record" type="tns:Record" />
    <element name="AppliedFilters" type="tns:RecordDetailsAppliedFilters"
/>
  </sequence>
</complexType>
```

RecordList data type

The `RecordList` data type contains a list of Endeca records.

```
<complexType name="RecordList">
  <sequence>
    <choice>
     <element name="Record" type="tns:Record" minOccurs="1" maxOccurs="un¬
bounded" />
      <element name="AggregateRecord" type="tns:AggregateRecord" minOc¬
curs="1" maxOccurs="unbounded" />
    </choice>
  </sequence>
</complexType>
```

RecordsPerAggregateRecord data type

The `RecordsPerAggregateRecord` data type enumerates the available choices for specifying how may constituent records to include with each aggregate record.

```
<simpleType name="RecordsPerAggregateRecord">
  <restriction base="string">
    <enumeration value="All" />
    <enumeration value="None" />
    <enumeration value="One" />
  </restriction>
</simpleType>
```

RecordsResult data type

The `RecordsResult` data type provides the information relating to records that is returned by the MDEX Engine in response to a navigation query.

```
<complexType name="RecordsResult">
  <sequence>
    <element name="Sorts" type="tns:SortList" minOccurs="0" maxOccurs="1"
/>
    <element name="IncludedRecordAttributes" type="tns:IncludedRecordAt¬
tributeList" minOccurs="0" maxOccurs="1" />
    <element name="AggregationKey" type="tns:AggregationKey" minOccurs="0"
 maxOccurs="1" />
   <element name="Records" type="tns:RecordList" minOccurs="0" maxOccurs="1"
 />
  </sequence>
  <attribute name="Offset" type="unsignedLong" use="required" />
  <attribute name="RecordsPerPage" type="unsignedLong" use="required" />
  <attribute name="TotalRecordCount" type="unsignedLong" use="required" />

  <attribute name="TotalAggregateRecordCount" type="unsignedLong" use="op¬
tional"/>
</complexType>
```

Attributes:

`Offset`	The record offset.
`RecordsPerPage`	The number of records per page.

`TotalRecordCount`	The total record count.
`TotalAggregateRecordCount`	The total aggregate record count.

RefinementConfig data type

The `RefinementConfig` data type represents a dynamic refinement configuration for a dimension value. This will determine how refinements are computed under this dimension value.

```
<complexType name="RefinementConfig">
  <attribute name="DimensionValueId" type="tns:DimensionValueId" use="re¬
quired" />
  <attribute name="Expose" type="boolean" default="true" use="optional" />

  <attribute name="LimitDimensionValues" type="boolean" use="optional" />
  <attribute name="OrderByRecordCount" type="boolean" use="optional" />
  <attribute name="MaximumDimensionValueCount" type="unsignedLong"
use="optional" />
  <attribute name="ShowCounts" type="boolean" use="optional" />
</complexType>
```

Attributes:

`DimensionValueId`	The dimension value ID.
`Expose`	Whether to expose the refinements for the dimension value.
`LimitDimensionValues`	Whether to limit the number of refinements.
`OrderByRecordCount`	Whether to order the dimension values by record count. **Note:** If you are using the `OrderByRecordCount` attribute to enable dynamic refinement ranking, keep in mind that the counts used by this feature are returned in the dimension tree, while the ordering is returned in the navigation state refinement tree. (This is true regardless of whether this feature is enabled globally in the `Refinement_config.xml` file or on a per-query basis.)
`MaximumDimensionValueCount`	The maximum number of dimension values to return, if `LimitDimensionValues` is true.
`ShowCounts`	Indicates whether to return refinement counts for a given `DimensionValueId`.

RefinementList data type

The `RefinementList` data type represents a list of refinements.

```
<complexType name="RefinementList">
  <complexContent>
    <extension base="tns:DimensionValueStateList">
      <attribute name="ParentName" type="string" use="required" />
     <attribute name="ParentId" type="tns:DimensionValueId" use="required"
 />
```

```
      <attribute name="HasMore" type="boolean" use="required" />
      <attribute name="IsRefinable" type="boolean" use="required" />
    </extension>
  </complexContent>
</complexType>
```

Attributes:

`ParentName`	The name of the parent dimension value.
`ParentId`	The parent's dimension value ID.
`HasMore`	Indicates whether there are more results that are not shown.
`IsRefinable`	Whether the dimension value can be refined.

RefinementConfigList data type

The `RefinementConfigList` data type represents a collection of dynamic refinement configs.

```
<complexType name="RefinementConfigList">
  <sequence>
    <element name="DisabledRefinementsConfig" type="tns:DisabledRefine¬
mentsConfig" minOccurs="0" maxOccurs="1" />
    <element name="RefinementConfig" type="tns:RefinementConfig" minOc¬
curs="0" maxOccurs="unbounded" />
  </sequence>
  <attribute name="ExposeAllRefinements" type="boolean" use="optional" />
</complexType>
```

Attributes:

`ExposeAllRefinements`	Globally sets whether to expose all of the dynamic refinement configs. Individual refinement settings override this setting.

RelevanceRanking data type

The `RelevanceRanking` data type sets the options for standalone relevance ranking.

The content of this data type is the collection of terms you want to perform relevance ranking upon.

```
<complexType name="RelevanceRanking">
    <simpleContent>
      <extension base="tns:NonEmptyString">
        <attribute name="Key" type="tns:NonEmptyString" use="required" />
       <attribute name="Strategy" type="tns:NonEmptyString" use="required"
 />
        <attribute name="Mode" type="tns:SearchMode" />
      </extension>
    </simpleContent>
  </complexType>
```

Attributes:

`Key`	The search key.
`Strategy`	The relevance ranking strategy to use.

`Mode`	The search mode, one of: All, AllAny, AllPartial, Any, Boolean, Partial, PartialMax, or Unknown.

ResultDimension data type

The `ResultDimension` data type represents the dimension search result, of which there is one per dimension.

```
<complexType name="ResultDimension">
  <sequence>
    <element name="MatchingDimensionValue" type="tns:AttributeDimensionValue"
 minOccurs="1" maxOccurs="unbounded" />
  </sequence>
  <attribute name="Name" type="tns:NonEmptyString" use="required" />
  <attribute name="Id" type="tns:DimensionValueId" use="required" />
</complexType>
```

Attributes:

`Name`	The dimension value name.
`Id`	The dimension value ID.

ResultDimensionList data type

The `ResultDimensionList` data type represents a list of dimension search results.

```
<complexType name="ResultDimensionList">
  <sequence>
    <element name="MatchingDimension" type="tns:ResultDimension" minOc¬
curs="1" maxOccurs="unbounded" />
  </sequence>
</complexType>
```

Search data type

The `Search` data type sets the options for record search in the navigation query.

The content of the Search element is your search terms.

```
<complexType name="Search">
  <attribute name="Key" type="string" use="required" />
  <attribute name="RelevanceRankingStrategy" type="string" />
  <attribute name="Mode" type="tns:SearchMode" />
  <attribute name="SnippetLength" type="unsignedLong" use="optional" />
  <attribute name="EnableSnippeting" type="boolean" use="optional" />
</complexType>
```

Attributes:

`Key`	The search key.
`RelevanceRankingStrategy`	The relevance ranking strategy, composed of one or more pre-defined relevance ranking modules.

`Mode`	The search mode, one of: All, AllAny, AllPartial, Any, Boolean, Partial, PartialMax, or Unknown.
`SnippetLength`	Maximum number of words in a snippet. This only takes effect if snippeting is enabled either in your configuration or via the `En¬ableSnippeting` attribute.
`EnableSnippeting`	Whether to enable the snippeting feature, which returns excerpts from records.

SearchAdjustment data type

The `SearchAdjustment` data type represents a single alternate search adjustment made by the MDEX Engine when a record search is performed.

The suggestion may or may not have been applied to the result set.

```
<complexType name="SearchAdjustment">
  <sequence>
    <element name="AdjustmentType" type="tns:AdjustmentType" minOccurs="1"
 maxOccurs="unbounded" />
  </sequence>
  <attribute name="Terms" type="tns:NonEmptyString" use="required" />
  <attribute name="RecordCountIfApplied" type="unsignedLong" use="required"
 />
</complexType>
```

Attributes:

`Terms`	Search terms.
`RecordCountIfApplied`	The number of records that would match if the terms of the search adjustment were used in the search.

SearchAdjustmentList data type

The `SearchAdjustmentList` data type represents a list of alternate search adjustments made by the MDEX Engine when a record search is performed.

```
<complexType name="SearchAdjustmentList">
  <sequence>
    <element name="SearchAdjustment" type="tns:SearchAdjustment" minOc¬
curs="1" maxOccurs="unbounded" />
  </sequence>
</complexType>
```

SearchKey data type

The `SearchKey` data type represents a search on a navigation record set.

The search key can be the name of an Endeca property or dimension (if enabled for record search) or an Endeca search interface. The search key indicates which field of the records to search the terms against.

```
<complexType name="SearchKey">
  <simpleContent>
```

```
    <extension base="tns:NonEmptyString">
      <attribute name="IsSearchInterface" type="boolean" use="required" />

    </extension>
  </simpleContent>
</complexType>
```

Attributes:

`IsSearchInterface`	Whether this search key is a search interface.

SearchKeyList data type

The `SearchKeyList` data type represents a collection of record search keys.

```
<complexType name="SearchKeyList">
  <sequence>
    <element name="SearchKey" type="tns:SearchKey" minOccurs="1" maxOc¬
curs="unbounded" />
  </sequence>
</complexType>
```

SearchList data type

The `SearchList` data type represents a list of searches.

This data type contains your search elements.

```
<complexType name="SearchList">
  <sequence>
    <element name="Search" type="tns:Search" minOccurs="1" maxOccurs="un¬
bounded" />
  </sequence>
  <attribute name="EnableDidYouMean" type="boolean" use="optional" />
</complexType>
```

Attributes:

`EnableDidYouMean`	Whether "did you mean" is enabled for all searches in this query, allowing an application to provide the user with explicit alternative suggestions for a keyword search.

SearchMode data type

The `SearchMode` data type enumerates the search modes available when performing a text search.

```
<simpleType name="SearchMode">
  <restriction base="string">
    <enumeration value="All"/>
    <enumeration value="AllAny"/>
    <enumeration value="AllPartial"/>
    <enumeration value="Any"/>
    <enumeration value="Boolean"/>
    <enumeration value="Partial"/>
    <enumeration value="PartialMax"/>
    <enumeration value="Unknown"/>
  </restriction>
</simpleType>
```

SearchReport data type

The `SearchReport` data type represents a description of a text search executed by the MDEX Engine.

```
<complexType name="SearchReport">
  <sequence>
    <element name="ErrorMessage" type="string" minOccurs="0" maxOccurs="1"
 />
    <choice>
      <sequence>
        <element name="Search" type="tns:Search" />
       <element name="MatchedRecordCount" type="unsignedLong" />ERec class
 is composed of properties and values it is tagged by.

      </sequence>
      <sequence>
        <element name="DimensionSearch" type="tns:DimensionSearch" />
        <element name="MatchedDimensionValueCount" type="unsignedLong" />
      </sequence>
      <sequence>
       <element name="CompoundDimensionSearch" type="tns:CompoundDimension¬
Search" />
        <element name="MatchedCompoundDimensionValueCount" type="unsigned¬
Long" />
      </sequence>
    </choice>
    <element name="MatchedMode" type="tns:SearchMode" />
    <element name="MatchedTermsCount" type="unsignedLong" />
   <element name="AppliedSearchAdjustments" type="tns:SearchAdjustmentList"
 minOccurs="0" maxOccurs="1" />
    <element name="SuggestedSearchAdjustments" type="tns:SearchAdjust¬
mentList" minOccurs="0" maxOccurs="1" />
   <element name="TruncatedTerms" type="string" minOccurs="0" maxOccurs="1"
 />
    <element name="WordInterpretations" type="tns:WordInterpretationList"
minOccurs="0" maxOccurs="1" />
  </sequence>
</complexType>
```

SearchReportList data type

The `SearchReportList` data type contains a list of search report elements.

```
<complexType name="SearchReportList">
  <sequence>
    <element name="SearchReport" type="tns:SearchReport" minOccurs="1"
maxOccurs="unbounded" />
  </sequence>
</complexType>
```

SnippetList data type

The `SnippetList` data type represents a list of snippets.

A snippet consists of search terms, surrounding context words, and ellipses.

```
<complexType name="SnippetList">
  <sequence>
   <element name="Snippet" type="tns:Property" minOccurs="1" maxOccurs="un¬
bounded" />
  </sequence>
</complexType>
```

Sort data type

The `Sort` data type contains the key on which records are sorted and describes how records should be sorted in relation to a particular property.

```
<complexType name="Sort">
  <sequence>
    <element name="ReferenceGeocode" type="tns:Geocode" minOccurs="0" max¬
Occurs="1" />
  </sequence>
  <attribute name="Key" type="tns:NonEmptyString" use="required" />
  <attribute name="Direction" type="tns:SortDirection" use="optional" />
</complexType>
```

Attributes:

`Key`	The sort key.
`Direction`	Ascending or Descending.

SortDirection data type

The SortDirection data type enumerates the directions in which records may be sorted.

```
<simpleType name="SortDirection">
  <restriction base="string">
    <enumeration value="Ascending"/>
    <enumeration value="Descending"/>
  </restriction>
</simpleType>
```

SortKeyList data type

The `SortKeyList` data type represents a collection of record sort keys.

```
<complexType name="SortKeyList">
  <sequence>
    <element name="SortKey" type="tns:NonEmptyString" minOccurs="1" maxOc¬
curs="unbounded" />
  </sequence>
</complexType>
```

SortList data type

The `SortList` data type represents a list of record sort keys.

This data type contains either a single standalone `RelevanceRanking` element or some number of `Sort` elements.

```
<complexType name="SortList">
  <sequence>
    <choice>
     <element name="Sort" type="tns:Sort" minOccurs="1" maxOccurs="unbound¬
ed" />
      <element name="RelevanceRanking" type="tns:RelevanceRanking" />
    </choice>
  </sequence>
</complexType>
```

UserProfileList data type

The `UserProfileList` data type represents a collection of user profiles.

```
<complexType name="UserProfileList">
  <sequence>
   <element name="UserProfile" type="tns:string" minOccurs=1 maxOccurs="un¬
bounded" />
  </sequence>
</complexType>
```

WordInterpretationList data type

The `WordInterpretationList` data type represents word or phrase substitutions made during text search processing.

```
<complexType name="WordInterpretationList">
  <sequence>
    <element name="WordInterpretation" type="tns:Property" minOccurs="1"
maxOccurs="unbounded" />
  </sequence>
</complexType>
```

MDEX API through XQuery examples

This section contains some examples of the MDEX API through XQuery in action.

Dimension search query example

This example uses the dimension tree structure to build a sequence of strings that represent the path from the root to the specified dimension value.

The following query:

```
import module namespace mdex = "http://www.endeca.com/XQuery/mdex/2008" at
 "mdex.xq";

let $query :=
   <Query xmlns="http://www.endeca.com/MDEX/data/IR600">
      <DimensionSearch>best</DimensionSearch>
      <DimensionValuesPerDimension>3</DimensionValuesPerDimension>
      <SelectedDimensionValueIds>
         <DimensionValueId>4294967161</DimensionValueId>
      </SelectedDimensionValueIds>
   </Query>
return
   mdex:dimension-search-query($query)
```

Would return this result:

```
<Results xmlns="http://www.endeca.com/MDEX/data/IR600">
   <Dimensions>
      <Dimension Name="Drinkability" Id="3" MultiSelect="None" Group¬
Name="Characteristics">
         <DimensionValue Name="Drinkability" Id="3" IsLeaf="false" IsNavi¬
gable="false">
            <DimensionValues>
               <DimensionValue Name="Best after 1998" Id="4294967041"
IsLeaf="true" IsNavigable="true"/>
               <DimensionValue Name="Best after 1997" Id="4294967052"
IsLeaf="true" IsNavigable="true"/>
               <DimensionValue Name="Best after 1996" Id="4294967093"
IsLeaf="true" IsNavigable="true"/>
            </DimensionValues>
         </DimensionValue>
      </Dimension>
     <Dimension Name="Designation" Id="7" MultiSelect="None" GroupName="Rat¬
ings">
        <DimensionValue Name="Designation" Id="7" IsLeaf="false" IsNaviga¬
ble="false">
            <DimensionValues>
               <DimensionValue Name="Best Buy" Id="8031" IsLeaf="true" Is¬
Navigable="true"/>
            </DimensionValues>
         </DimensionValue>
      </Dimension>
   </Dimensions>
   <MatchingDimensionsResult HasMore="true">
      <DimensionValuesPerDimension>3</DimensionValuesPerDimension>
      <MatchingDimensions>
         <MatchingDimension Name="Designation" Id="7">
            <MatchingDimensionValue Key="Designation" DimensionId="7"
Id="8031">Best Buy</MatchingDimensionValue>
         </MatchingDimension>
         <MatchingDimension Name="Drinkability" Id="3">
            <MatchingDimensionValue Key="Drinkability" DimensionId="3"
Id="4294967093">Best after 1996</MatchingDimensionValue>
            <MatchingDimensionValue Key="Drinkability" DimensionId="3"
Id="4294967052">Best after 1997</MatchingDimensionValue>
            <MatchingDimensionValue Key="Drinkability" DimensionId="3"
Id="4294967041">Best after 1998</MatchingDimensionValue>
         </MatchingDimension>
      </MatchingDimensions>
   </MatchingDimensionsResult>
   <AppliedFilters>
      <SearchReport>
         <DimensionSearch>best</DimensionSearch>
         <MatchedDimensionValueCount>80</MatchedDimensionValueCount>
         <MatchedMode>All</MatchedMode>
         <MatchedTermsCount>1</MatchedTermsCount>
      </SearchReport>
      <SelectedDimensionValueIds>
         <DimensionValueId>4294967161</DimensionValueId>
      </SelectedDimensionValueIds>
   </AppliedFilters>
</Results>
```

Build display record output example

This example converts a record from the default MDEX API through XQuery format into a customized format for an application to consume.

The following query:

```
import module namespace mdex = "http://www.endeca.com/XQuery/mdex/2008" at
 "mdex.xq";

declare namespace mdata = "http://www.endeca.com/MDEX/data/IR600";

(:~
 : Converts a record from the default MDEX API through XQuery format
 : into a customized format for an application to consume
 : @param $record An mdata:Record element in MDEX API through XQuery format

 : @param $titleKey The key of an attribute that should be used as the title

 : element. Title will default to the record ID if an attribute with this
 : key does not exist. If multiple attributes with this key exists, it will

 : use the first one as dictated by document order.
 : @return An element of the following format
 : <Record>
 :    <Title>title</Title>
 :    <Row>
 :       <Key>key</Key>
 :       <Value>value</Value>
 :    </Row>
 : </Record>
 :)
declare function local:build-display-record($record as element(mdata:Record,
 xs:untyped), $titleKey as xs:string)
as element(Record, xs:untyped)
{
   element Record {
     let $titleAttribute := $record/mdata:Attributes/*[@Key = $titleKey][1]

      return
         element Title {
            if (fn:exists($titleAttribute)) then
               fn:data($titleAttribute)
            else
               fn:data($record/@Id)
         },
      for $attribute in $record/mdata:Attributes/*[@Key != $titleKey]
      return
         element Row {
            element Key {fn:data($attribute/@Key)},
            element Value {fn:data($attribute)}
         }
   }
};
```

```
let $query :=
   <Query xmlns="http://www.endeca.com/MDEX/data/IR600">
      <SelectedDimensionValueIds>
         <DimensionValueId>8026</DimensionValueId>
         <DimensionValueId>29</DimensionValueId>
      </SelectedDimensionValueIds>
      <RefinementConfigs>
         <RefinementConfig DimensionValueId="8"/>
      </RefinementConfigs>
      <Searches>
         <Search Key="P_Description">fruity</Search>
      </Searches>
      <RecordsPerPage>1</RecordsPerPage>
   </Query>
let $navigationResults := mdex:navigation-query($query)
for $record in $navigationResults/mdata:RecordsResult/mdata:Records/mda¬
ta:Record
return
   local:build-display-record($record, "P_Name")
```

Would return this result:

```
<Record>
   <Title>Cabernet Franc Yakima Valley Red Willow Vineyard Signature Se¬
ries</Title>
   <Row>
      <Key>Review Score</Key>
      <Value>80 to 90</Value>
   </Row>
   <Row>
      <Key>P_DateReviewed</Key>
      <Value>09/30/94</Value>
   </Row>
   <Row>
      <Key>P_Description</Key>
      <Value>Bright and fruity, soft-textured, sharply focused, with mar¬
velous berry and cranberry flavors and none of the stalkiness that afflicts
 other Cabernet Francs. Drinkable now. (400 cases produced)</Value>
   </Row>
   <Row>
      <Key>P_Price</Key>
      <Value>16.000000</Value>
   </Row>
   <Row>
      <Key>P_Region</Key>
      <Value>Washington</Value>
   </Row>
   <Row>
      <Key>P_Score</Key>
      <Value>87</Value>
   </Row>
   <Row>
      <Key>P_WineID</Key>
      <Value>35078</Value>
   </Row>
   <Row>
      <Key>P_Winery</Key>
      <Value>Columbia</Value>
   </Row>
   <Row>
      <Key>P_WineType</Key>
      <Value>Cabernet Franc</Value>
   </Row>
   <Row>
      <Key>P_WineType</Key>
      <Value>Red</Value>
   </Row>
   <Row>
      <Key>P_Year</Key>
      <Value>1992</Value>
   </Row>
</Record>
```

Build breadcrumbs output example

This example creates breadcrumbs for selected dimension values and text searches based on results returned from an invocation of the `mdex:navigation-query()` function.

The following query:

```
import module namespace mdex = "http://www.endeca.com/XQuery/mdex/2008" at
 "mdex.xq";

declare namespace mdata = "http://www.endeca.com/MDEX/data/IR600";

(:~
 : Creates bread crumbs for selected dimension values and text
 : searches based on results returned from an mdex:navigation-query()
 : invocation
 : @param $navigationResults mdata:NavigationResults element as returned
from an
 : mdex:navigation-query() invocation
 : @return Breadcrumbs element of the following format
 : <Breadcrumbs>
 :    <Breadcrumb>Wine Type -> Red -> Merlot</Breadcrumb>
 :    <Breadcrumb>Text Search: Sparkling</Breadcrumb>
 : </Breadcrumbs>
 :)
declare function local:build-breadcrumbs($navigationResults as element(mda¬
ta:NavigationResults, xs:untyped))
as element(Breadcrumbs, xs:untyped)
{
   let $appliedFilters := $navigationResults/mdata:AppliedFilters
   return
      element Breadcrumbs {
         for $selectedDimValId in $appliedFilters/mdata:SelectedDimension¬
ValueIds/mdata:DimensionValueId
         return
            element Breadcrumb {
               fn:string-join(local:path-from-dimval-id(fn:data($selected¬
DimValId), fn:exactly-one($navigationResults/mdata:Dimensions)), " -> ")
            },
         for $searchReport in $appliedFilters/mdata:SearchReports/mda¬
ta:SearchReport
         return
            element Breadcrumb {
              fn:concat("Text Search: ", fn:zero-or-one(fn:data($searchRe¬
port/mdata:Search)))
            }
      }
};

(:~
 : Uses the Dimension tree(s) structure to build a sequence of strings
 : that represent the path from the root to the specified dimension value
 : (Note: This function will run fairly slowly when used on a large
 : collection of dimensions)
 : @param $dimValId Dimension value ID of the dimension value to search for

 : @param $dimensions mdata:Dimensions element from a result returned from
 a
 : MDEX API through XQuery invocation
 : @return Sequence of strings that represent the path from the root
 : dimension value up to the specified dimension value if the dimension
 : value is found. Returns the empty sequence otherwise.
 :)
declare function local:path-from-dimval-id($dimValId as xs:string, $dimen¬
sions as element(mdata:Dimensions, xs:untyped))
as xs:string*
{
   for $dimVal in $dimensions//mdata:DimensionValue[descendant-or-
self::mdata:DimensionValue/@Id = $dimValId]
   return
      fn:string(fn:exactly-one(fn:data($dimVal/@Name)))
};

let $query :=
   <Query xmlns="http://www.endeca.com/MDEX/data/IR600">
      <SelectedDimensionValueIds>
         <DimensionValueId>8026</DimensionValueId>
         <DimensionValueId>4294967281</DimensionValueId>
      </SelectedDimensionValueIds>
      <RefinementConfigs>
         <RefinementConfig DimensionValueId="8"/>
      </RefinementConfigs>
      <Searches>
         <Search Key="P_Description">fruity</Search>
      </Searches>
      <RecordsPerPage>1</RecordsPerPage>
   </Query>
let $navigationResults := mdex:navigation-query($query)
return
   local:build-breadcrumbs($navigationResults)
```

Would return this result:

```
<Breadcrumbs>
   <Breadcrumb>Wine Type -> Red -> Cabernet Franc</Breadcrumb>
   <Breadcrumb>Flavors -> Berry</Breadcrumb>
   <Breadcrumb>Text Search: fruity</Breadcrumb>
</Breadcrumbs>
```

Navigation query example

The following example shows a navigation query and its result.

The following query:

```
import module namespace mdex = "http://www.endeca.com/XQuery/mdex/2008" at
 "mdex.xq";

let $query :=
   <Query xmlns="http://www.endeca.com/MDEX/data/IR600">
      <SelectedDimensionValueIds>
         <DimensionValueId>8026</DimensionValueId>
         <DimensionValueId>4294967281</DimensionValueId>
      </SelectedDimensionValueIds>
      <RefinementConfigs>
         <RefinementConfig DimensionValueId="8"/>
      </RefinementConfigs>
      <Searches>
         <Search Key="P_Description">fruity</Search>
      </Searches>
      <RecordsPerPage>1</RecordsPerPage>
   </Query>
return
   mdex:navigation-query($query)
```

Would return this result:

```
<Results xmlns="http://www.endeca.com/MDEX/data/IR600">
   <Dimensions>
      <Dimension Name="Vintage" Id="2" MultiSelect="None">
         <DimensionValue Name="Vintage" Id="2" IsLeaf="false" IsNaviga¬
ble="false">
            <DimensionValues>
               <DimensionValue Name="1992" Id="4294967284" IsLeaf="true"
IsNavigable="true"/>
            </DimensionValues>
         </DimensionValue>
      </Dimension>
      <Dimension Name="Drinkability" Id="3" MultiSelect="None" Group¬
Name="Characteristics">
         <DimensionValue Name="Drinkability" Id="3" IsLeaf="false" IsNavi¬
gable="false"/>
      </Dimension>
     <Dimension Name="Body" Id="5" MultiSelect="None" GroupName="Character¬
istics">
         <DimensionValue Name="Body" Id="5" IsLeaf="false" IsNaviga¬
ble="false">
            <DimensionValues>
              <DimensionValue Name="Complex" Id="4294967064" IsLeaf="true"
 IsNavigable="true"/>
             <DimensionValue Name="Delicious" Id="4294967140" IsLeaf="true"
 IsNavigable="true"/>
               <DimensionValue Name="Lively" Id="4294967199" IsLeaf="true"
 IsNavigable="true"/>
               <DimensionValue Name="Rich" Id="4294967236" IsLeaf="true"
IsNavigable="true"/>
            </DimensionValues>
         </DimensionValue>
      </Dimension>
     <Dimension Name="Designation" Id="7" MultiSelect="None" GroupName="Rat¬
ings">
        <DimensionValue Name="Designation" Id="7" IsLeaf="false" IsNaviga¬
ble="false">
            <DimensionValues>
               <DimensionValue Name="Highly Recommended" Id="8029"
```

```
IsLeaf="true" IsNavigable="true"/>
            </DimensionValues>
         </DimensionValue>
      </Dimension>
      <Dimension Name="Region" Id="8" MultiSelect="None">
         <DimensionValue Name="Region" Id="8" IsLeaf="false" IsNaviga¬
ble="false">
            <DimensionValues>
             <DimensionValue Name="Virginia" Id="4294965997" IsLeaf="true"
 IsNavigable="true"/>
             <DimensionValue Name="Northeast" Id="4294966758" IsLeaf="true"
 IsNavigable="true"/>
               <DimensionValue Name="Washington" Id="4294966974"
IsLeaf="true" IsNavigable="true"/>
               <DimensionValue Name="Napa" Id="4294967161" IsLeaf="true"
IsNavigable="true"/>
            </DimensionValues>
            <Properties>
               <Property Key="DGraph.More">0</Property>
            </Properties>
         </DimensionValue>
      </Dimension>
     <Dimension Name="Review Score" Id="9" MultiSelect="None" GroupName="Rat¬
ings">
         <DimensionValue Name="Review Score" Id="9" IsLeaf="false" IsNavi¬
gable="false">
            <DimensionValues>
              <DimensionValue Name="80 to 90" Id="29" IsLeaf="true" IsNav¬
igable="true"/>
            </DimensionValues>
         </DimensionValue>
      </Dimension>
     <Dimension Name="Price Range" Id="10" MultiSelect="None" GroupName="Rat¬
ings">
         <DimensionValue Name="Price Range" Id="10" IsLeaf="false" IsNavi¬
gable="false">
            <DimensionValues>
               <DimensionValue Name="$10 to $20" Id="8033" IsLeaf="true"
IsNavigable="true"/>
            </DimensionValues>
         </DimensionValue>
      </Dimension>
      <Dimension Name="Winery" Id="11" MultiSelect="None">
         <DimensionValue Name="Winery" Id="11" IsLeaf="false" IsNaviga¬
ble="false">
            <DimensionValues>
             <DimensionValue Name="Dickerson" Id="4294965608" IsLeaf="true"
 IsNavigable="true"/>
            </DimensionValues>
         </DimensionValue>
      </Dimension>
     <Dimension Name="Flavors" Id="12" MultiSelect="None" GroupName="Char¬
acteristics">
         <DimensionValue Name="Flavors" Id="12" IsLeaf="false" IsNaviga¬
ble="false">
            <DimensionValues>
              <DimensionValue Name="Currant" Id="4294967155" IsLeaf="true"
 IsNavigable="true"/>
               <DimensionValue Name="Oak" Id="4294967194" IsLeaf="true"
IsNavigable="true"/>
               <DimensionValue Name="Fruity" Id="4294967218" IsLeaf="true"
 IsNavigable="true"/>
               <DimensionValue Name="Fruit" Id="4294967269" IsLeaf="true"
IsNavigable="true"/>
               <DimensionValue Name="Cherry" Id="4294967279" IsLeaf="true"
 IsNavigable="true"/>
               <DimensionValue Name="Berry" Id="4294967281" IsLeaf="true"
IsNavigable="true"/>
            </DimensionValues>
         </DimensionValue>
      </Dimension>
      <Dimension Name="Wine Type" Id="6200" MultiSelect="None">
         <DimensionValue Name="Wine Type" Id="6200" IsLeaf="false" IsNavi¬
gable="false">
            <DimensionValues>
               <DimensionValue Name="Red" Id="8021" IsLeaf="false" IsNavi¬
gable="true">
                  <DimensionValues>
                     <DimensionValue Name="Merlot" Id="8025" IsLeaf="true"
 IsNavigable="true"/>
                     <DimensionValue Name="Cabernet Franc" Id="8026"
IsLeaf="true" IsNavigable="true"/>
                  </DimensionValues>
               </DimensionValue>
            </DimensionValues>
         </DimensionValue>
      </Dimension>
      <Dimension Name="Endeca" Id="4294967294" MultiSelect="None">
        <DimensionValue Name="Endeca" Id="4294967294" IsLeaf="false" IsNav¬
igable="false">
            <DimensionValues>
               <DimensionValue Name="Endeca" Id="4294967293" IsLeaf="true"
 IsNavigable="true"/>
            </DimensionValues>
         </DimensionValue>
      </Dimension>
   </Dimensions>
   <NavigationStatesResult>
      <RefinementConfigs>
         <RefinementConfig DimensionValueId="8"/>
      </RefinementConfigs>
      <DimensionStates>
         <DimensionState DimensionName="Wine Type" DimensionId="6200">
            <Refinements ParentName="Cabernet Franc" ParentId="8026" Has¬
More="false" IsRefinable="false"/>
            <SelectedDimensionValues>
               <DimensionValueReference Name="Cabernet Franc" Id="8026"/>
            </SelectedDimensionValues>
         </DimensionState>
         <DimensionState DimensionName="Region" DimensionId="8">
            <Refinements ParentName="Region" ParentId="8" HasMore="false"
IsRefinable="true">
               <DimensionValueReference Name="Napa" Id="4294967161"/>
               <DimensionValueReference Name="Northeast" Id="4294966758"/>

               <DimensionValueReference Name="Virginia" Id="4294965997"/>
              <DimensionValueReference Name="Washington" Id="4294966974"/>

            </Refinements>
         </DimensionState>
         <DimensionState DimensionName="Vintage" DimensionId="2">
            <Refinements ParentName="Vintage" ParentId="2" HasMore="false"
 IsRefinable="true"/>
         </DimensionState>
         <DimensionState DimensionName="Price Range" DimensionId="10">
            <Refinements ParentName="Price Range" ParentId="10" Has¬
More="false" IsRefinable="true"/>
         </DimensionState>
         <DimensionState DimensionName="Review Score" DimensionId="9">
            <Refinements ParentName="Review Score" ParentId="9" Has¬
More="false" IsRefinable="true"/>
         </DimensionState>
         <DimensionState DimensionName="Body" DimensionId="5">
            <Refinements ParentName="Body" ParentId="5" HasMore="false"
IsRefinable="true"/>
         </DimensionState>
         <DimensionState DimensionName="Flavors" DimensionId="12">
            <Refinements ParentName="Berry" ParentId="4294967281" Has¬
More="false" IsRefinable="false"/>
            <SelectedDimensionValues>
               <DimensionValueReference Name="Berry" Id="4294967281"/>
            </SelectedDimensionValues>
            <ImplicitDimensionValues>
               <DimensionValueReference Name="Fruit" Id="4294967269"/>
               <DimensionValueReference Name="Fruity" Id="4294967218"/>
            </ImplicitDimensionValues>
         </DimensionState>
         <DimensionState DimensionName="Drinkability" DimensionId="3">
            <Refinements ParentName="Drinkability" ParentId="3" Has¬
More="false" IsRefinable="true"/>
         </DimensionState>
         <DimensionState DimensionName="Endeca" DimensionId="4294967294">
            <Refinements ParentName="Endeca" ParentId="4294967294" Has¬
More="false" IsRefinable="false"/>
            <ImplicitDimensionValues>
               <DimensionValueReference Name="Endeca" Id="4294967293"/>
            </ImplicitDimensionValues>
         </DimensionState>
      </DimensionStates>
   </NavigationStatesResult>
   <RecordsResult Offset="0" RecordsPerPage="1" TotalRecordCount="6">
      <Records>
         <Record Id="35078">
            <Attributes>
               <AssignedDimensionValue Key="Review Score" DimensionId="9"
Id="29">80 to 90</AssignedDimensionValue>
               <Property Key="P_DateReviewed">09/30/94</Property>
               <Property Key="P_Description">Bright and fruity, soft-tex¬
tured, sharply focused, with marvelous berry and cranberry flavors and none
 of the stalkiness that afflicts other Cabernet Francs. Drinkable now. (400
 cases produced)</Property>
               <Property Key="P_Name">Cabernet Franc Yakima Valley Red
Willow Vineyard Signature Series</Property>
               <Property Key="P_Price">16.000000</Property>
               <Property Key="P_Region">Washington</Property>
               <Property Key="P_Score">87</Property>
               <Property Key="P_WineID">35078</Property>
               <Property Key="P_Winery">Columbia</Property>
               <Property Key="P_WineType">Cabernet Franc</Property>
               <Property Key="P_WineType">Red</Property>
               <Property Key="P_Year">1992</Property>
            </Attributes>
         </Record>
      </Records>
   </RecordsResult>
   <AppliedFilters>
      <SearchReports>
         <SearchReport>
            <Search Key="P_Description">fruity</Search>
            <MatchedRecordCount>3802</MatchedRecordCount>
            <MatchedMode>All</MatchedMode>
            <MatchedTermsCount>1</MatchedTermsCount>
         </SearchReport>
      </SearchReports>
      <SelectedDimensionValueIds>
         <DimensionValueId>8026</DimensionValueId>
         <DimensionValueId>4294967281</DimensionValueId>
      </SelectedDimensionValueIds>
   </AppliedFilters>
   <BusinessRulesResult>
      <BusinessRules>
         <BusinessRule Id="1" NavigationStateRecordCount="18" Style="Style
 1" Title="Recommended Merlots" Zone="Zone One">
            <SelectedDimensionValueIds>
               <DimensionValueId>8025</DimensionValueId>
               <DimensionValueId>8029</DimensionValueId>
            </SelectedDimensionValueIds>
            <Properties>
               <Property Key="StyleTitle">Style One Title</Property>
            </Properties>
            <Records>
               <Record Id="37370">
                  <Attributes>
                     <AssignedDimensionValue Key="Wine Type" Dimension¬
Id="6200" Id="8025">Merlot</AssignedDimensionValue>
                     <AssignedDimensionValue Key="Region" DimensionId="8"
Id="4294967161">Napa</AssignedDimensionValue>
                     <AssignedDimensionValue Key="Winery" DimensionId="11"
 Id="4294965608">Dickerson</AssignedDimensionValue>
                     <AssignedDimensionValue Key="Vintage" DimensionId="2"
 Id="4294967284">1992</AssignedDimensionValue>
                     <AssignedDimensionValue Key="Price Range" Dimension¬
Id="10" Id="8033">$10 to $20</AssignedDimensionValue>
                     <AssignedDimensionValue Key="Review Score" Dimension¬
Id="9" Id="29">80 to 90</AssignedDimensionValue>
                     <AssignedDimensionValue Key="Designation" Dimension¬
Id="7" Id="8029">Highly Recommended</AssignedDimensionValue>
                     <AssignedDimensionValue Key="Body" DimensionId="5"
Id="4294967064">Complex</AssignedDimensionValue>
                     <AssignedDimensionValue Key="Body" DimensionId="5"
Id="4294967140">Delicious</AssignedDimensionValue>
                     <AssignedDimensionValue Key="Body" DimensionId="5"
Id="4294967199">Lively</AssignedDimensionValue>
                     <AssignedDimensionValue Key="Body" DimensionId="5"
Id="4294967236">Rich</AssignedDimensionValue>
                    <AssignedDimensionValue Key="Flavors" DimensionId="12"
 Id="4294967279">Cherry</AssignedDimensionValue>
                    <AssignedDimensionValue Key="Flavors" DimensionId="12"
 Id="4294967155">Currant</AssignedDimensionValue>
                    <AssignedDimensionValue Key="Flavors" DimensionId="12"
 Id="4294967194">Oak</AssignedDimensionValue>
                     <Property Key="P_Body">Complex</Property>
                     <Property Key="P_Body">Delicious</Property>
                     <Property Key="P_Body">Lively</Property>
                     <Property Key="P_Body">Rich</Property>
                     <Property Key="P_DateReviewed">02/28/95</Property>
                    <Property Key="P_Description">Intense and lively, with
 a solid core of currant, cherry and light oak shadings that give it rich¬
ness, depth and complexity. Delicious now, but should drink well through
1998. (200 cases produced) Highly Recommended</Property>
                   <Property Key="P_Designation">Highly Recommended</Prop¬
erty>
                  <Property Key="P_Designation">Limited Reserve</Property>

                     <Property Key="P_Designation">Reserve</Property>
                     <Property Key="P_Flavor">Cherry</Property>
                     <Property Key="P_Flavor">Currant</Property>
                     <Property Key="P_Flavor">Oak</Property>
                     <Property Key="P_Name">Merlot Napa Valley Limited Re¬
serve</Property>
                     <Property Key="P_Price">17.000000</Property>
                     <Property Key="P_Region">Napa</Property>
                     <Property Key="P_Score">90</Property>
                     <Property Key="P_WineID">37370</Property>
                     <Property Key="P_Winery">Dickerson</Property>
                     <Property Key="P_WineType">Merlot</Property>
                     <Property Key="P_WineType">Red</Property>
                     <Property Key="P_Year">1992</Property>
                  </Attributes>
               </Record>
            </Records>
         </BusinessRule>
      </BusinessRules>
   </BusinessRulesResult>
   <KeywordRedirects/>
</Results>
```

Analytics example

In this example, an Analytics query selects the top three highest-rated wineries based on wines priced less than $20.

The Analytics statement is submitted as a string, and a list of records is returned.

Note: Analytics is a separately licensed product. For details, contact your Endeca representative.

The following query:

```
mdex:navigation-query(
 <Query xmlns="http://www.endeca.com/MDEX/data/IR600">
  <AnalyticsExpression>RETURN "Best Wineries" AS SELECT AVG("P_Score") AS
"Average Score" WHERE P_Price &lt; 20 GROUP BY "Winery" ORDER BY "Average
Score" DESC PAGE (0, 3)</AnalyticsExpression>
 </Query>
)
```

Would return the following result. (This snippet displays only the Analytics portion of the query. This element would be under a `NavigationResults` element.)

```
<AnalyticsResult xmlns="http://www.endeca.com/MDEX/data/IR600">
  <AnalyticsStatementResult Name="Best Wineries" TotalRecordCount="4902">
    <Record Id="528">
      <Attributes>
        <AssignedDimensionValue DimensionId="11" Id="4294964603" Key="Win¬
```

```
ery">Jean-Paul Droin</AssignedDimensionValue>
        <Property Key="Average Score">94.000000</Property>
      </Attributes>
    </Record>
    <Record Id="4664">
      <Attributes>
        <AssignedDimensionValue DimensionId="11" Id="4294960751" Key="Win¬
ery">3 Bridges</AssignedDimensionValue>
        <Property Key="Average Score">94.000000</Property>
      </Attributes>
    </Record>
    <Record Id="1266">
      <Attributes>
        <AssignedDimensionValue DimensionId="11" Id="4294961269" Key="Win¬
ery">Roberto Ferraris</AssignedDimensionValue>
        <Property Key="Average Score">93.000000</Property>
      </Attributes>
    </Record>
  </AnalyticsStatementResult>
</AnalyticsResult>
```

Chapter 5

Early Access features

Early Access content in this release

This release of Web services and XQuery for Endeca includes some features that are in an Early Access state.

The interfaces and behavior of these Early Access features may change, based on information gathered during this Early Access program, and they are not supported for use in production. Endeca gives no guarantees about backwards compatibility of these features in future releases.

The Early Access release of these features gives you a chance to start working with the new functionality as soon as possible in the development cycle. At the same time, your feedback can uncover problems and help shape ongoing development.

Dimension value specs

A dimension value spec, or dimension value specifier, is a special dimension value property used to identify a dimension value. The Data Update API uses dimension value specs to represent assignments.

Important: This feature is available as Early Access software. Its interface is likely to change in a future release, and its use is not supported in production.

A dimension value spec, together with the name of the dimension, provides a simple way to refer to dimension values that is stable from one baseline update to the next and across Dgraph replicas in a cluster. The dimension value spec will gradually replace dimension value IDs as a way to refer to dimension values in the various Endeca XQuery APIs. During this transitional period, over the course of the next several releases, a set of conversion functions (documented below) make it possible for you to link the two.

Although a dimension value spec is not required, when one is present it serves as the primary key for the dimension values within a dimension. One dimension value property, called `DGraph.Spec`, is used to assign and read the dimension value spec. Unlike other dimension value properties, dimension value specs are indexed, making it possible to use them for efficient lookup.

You can create dimension value properties, including the dimension value spec, in Developer Studio, except on the root dimension value. If you want to add the dimension value spec to a root dimension value, you must edit or transform the `dimensions.xml` file directly. (For more information about adding dimension value properties, see the section "Working with Dimensions and Dimension Values" in the *Developer Studio Help*.)

Dimension value specs have the following requirements:

- They must be unique within the dimension. That is to say, two dimension values in the same dimension cannot have the same value of `DGraph.Spec`.
- They must be single assign. That is, one dimension value cannot have two properties named `DGraph.Spec`. (Take care when creating properties in Developer Studio, because that tool does not enforce single assign.)

If either of these requirements is violated, you will receive an error during Dgidx processing or when using the Data Update API.

Note: Neither the explicit setting of dimension value specs nor their auto-generation is supported with file-based partial updates in this release.

Dimension value spec limitations

Keep in mind the following limitations to using dimension value specs.

- You cannot use dimension value specs to navigate to or look up details of dimension values in the Presentation APIs. You can only do so through XQuery, using the conversion functions provided for this purpose.

Note: Dimension value IDs continue to work in the Presentation API as in previous releases.

Conversion functions for dimension value specs

Three conversion functions facilitate the use of dimension value specs at query time by allowing convenient translation between dimension value specs and dimension value IDs.

Important: This feature is available as Early Access software. Its interface is likely to change in a future release, and its use is not supported in production.

Many Endeca features, including the MDEX API through XQuery (or MAX), use dimension value IDs rather than dimension value specs as identifiers. However, you may write a custom Web service that uses dimension value specs. The conversion functions described below, when used within a custom Web service, allow you convert request and response values between dimension value IDs and dimension value specs.

For example, you might submit a query with a dimension name and dimension value spec pair. You could use the `mdex-data:dimension-value-id-from-spec` function to rewrite the dimension name and its dimension value spec to its ID. You could run the query with the ID through MAX and get the result, and then use the `mdex-data:dimension-value-spec-from-id` function to rewrite ID to dimension name plus dimension value spec. (Keep in mind that while dimension value IDs are unique across the MDEX Engine, dimension value specs are only unique by dimension. That is why both the dimension name and the dimension value spec are required.)

- `mdex-data:dimension-value-spec-from-id` returns the dimension value spec of the dimension value with the given ID, if any.
- `mdex-data:dimension-name-from-id` returns the name of the dimension containing the dimension value with the given ID, if any.

- `mdex-data:dimension-value-id-from-spec` returns the ID of the dimension value within the given dimension, with the given spec, if any.

All of these functions raise an error if the specified dimension value does not exist or has no spec. For more information, see the topic "XQuery try/catch expressions."

For full details and examples of these functions, see the specific topics that follow.

Declaring and using conversion functions

The conversion functions use the namespaces and schema described below.

The namespace used by conversion functions is:

```
http://www.endeca.com/MDEX/update/2009/EarlyAccess
```

The namespace used by conversion functions elements is:

```
http://www.endeca.com/MDEX/data/2009/EarlyAccess
```

To use any of the `mdex-data` functions, the import statement that follows, or equivalent declarations, must be included in that module.

```
import module namespace mdex-data =
"http://www.endeca.com/MDEX/update/2009/EarlyAccess at "mdex_data.xq";
```

The schema file, `mdex_data.xsd`, is located in `$ENDECA_MDEX_ROOT/conf/schema`.

mdex-data:dimension-value-id-from-spec

The `mdex-data:dimension-value-id-from-spec` function returns the ID of the dimension value within the given dimension, with the given spec, if any. It raises an error if there is no dimension value within the dimension with the given spec value.

Important: This feature is available as Early Access software. Its interface is likely to change in a future release, and its use is not supported in production.

Function Declaration	`declare function mdex-data:dimension-value-id-from-spec($dimensionName as xs:string, $spec as xs:string) as xs:string`
Function Summary	Returns the ID for the dimension value that is associated with `$spec` within the dimension `$dimensionName`.
Parameters	`$dimensionName` is the name of the dimension that the dimension value belongs to. `$spec` is the value of the spec property on the dimension value you are looking for.
Returns	The dimension value ID of the dimension value within `$dimensionName` that is associated with `$spec`.

Example	Using the following example dimension value (in MAX format):
	`<data:DimensionValue Name="Merlot" Id="8025" IsLeaf="true" IsNavigable="true">`
	` <data:Properties>`

	`  <data:Property Key="DGraph.Spec">/Wine%20Type/Merlot</mda¬`
	`ta:Property>`
	` </data:Properties>`
	`</data:DimensionValue>`
	
	the function `mdex-data:dimension-value-id-from-spec("Wine Type", "/Wine%20Type/Merlot")`
	
	returns "8025".

mdex-data:dimension-value-spec-from-id

The `mdex-data:dimension-value-spec-from-id` function returns the dimension value spec of the dimension value with the given ID, if any. It raises an error if there is no dimension value within the dimension with the given ID.

Important: This feature is available as Early Access software. Its interface is likely to change in a future release, and its use is not supported in production.

Function Declaration	`declare function`
	`mdex-data:dimension-value-spec-from-id($id as xs:string) as`
	` xs:string`

Function Summary	Returns the spec value associated with the dimension value specified by `$id`.
Parameters	`$id` is the ID of the dimension value you want the spec for.
Returns	The dimension value spec associated with the dimension value.

Example	Using the following example dimension value (in MAX format):
	`<data:DimensionValue Name="Merlot" Id="8025" IsLeaf="true"`
	`IsNavigable="true">`
	` <data:Properties>`
	`  <data:Property Key="DGraph.Spec">/Wine%20Type/Merlot</mda¬`
	`ta:Property>`
	` </data:Properties>`
	`</data:DimensionValue>`
	
	the function `mdex-data:dimension-value-spec-from-id("8025")`
	
	returns "/Wine%20Type/Merlot".

mdex-data:dimension-name-from-id

The `dimension-name-from-id` function returns the name of the dimension containing the dimension value with the given ID, if any. It raises an error if no dimension contains a dimension value with the given ID.

Important: This feature is available as Early Access software. Its interface is likely to change in a future release, and its use is not supported in production.

Function Declaration	`declare function mdex-data:dimension-name-from-id($id as xs:string) as xs:string`
Function Summary	Returns the name of the dimension that the dimension value with ID `$id` belongs to.
Parameters	`$id` is the ID of the dimension value you want to find the dimension for.
Returns	The name of the dimension that the dimension value belongs to.

Example	Using the following example dimension value (in MAX format):
	`<data:DimensionValue Name="Merlot" Id="8025" IsLeaf="true" IsNavigable="true">`
	` <data:Properties>`
	`  <data:Property Key="DGraph.Spec">/Wine%20Type/Merlot</mda¬ta:Property>`
	` </data:Properties>`
	`</data:DimensionValue>`
	the function `mdex-data:dimension-name-from-id("8025")` returns "Wine Type".

fn:put() and persistent document storage

The built-in functions `fn:put()` and `fn:doc()` can be used to store and retrieve XML documents in the MDEX Engine.

Important: This feature is available as Early Access software. Its interface is likely to change in a future release, and its use is not supported in production.

`fn:put()` is an updating function specified by the XQuery Update Facility 1.0 Candidate Recommendation 1 that stores a specified document or element at a specified URI. Documents stored via `fn:put()` can be retrieved in subsequent queries via `fn:doc()`.

The syntax of `fn:put()` is as follows:

```
fn:put($node as node(), $uri as xs:string) as empty-sequence()
```

The Endeca XQuery implementation supports the use of `fn:put()` with the following constraints:

- `$node` must be a document or an element node. (Element nodes stored by `fn:put()` are implicitly converted to documents.)
- The `$uri` at which the document node is place must begin with the prefix `"mdex://documents/"`.

Note the following:

- Calling `fn:put()` with a URI for which a document already exists overwrites the previous value.
- Directly calling `fn:put()` with any URI that does not begin with the `mdex://documents/` prefix results in the error `err:FODC0005: Invalid argument to fn:doc or fn:doc-available` being called. (The prefix `err` here maps to the namespace *http://www.w3.org/2005/xqt-errors*.)
- The use of `fn:doc()` on a URI that does not begin with `mdex://` (such as `file://` or `http://`) is unaffected by this feature.

- The use of `fn:doc()` on URIs that begin with `mdex://` is unaffected by the `--xquery_fndoc` command-line flag.
- It is not possible to completely remove documents stored with the `fn:put()` function. However, documents can be overwritten with very small documents of the same name.

Persistent document storage example

The code example below stores the element `<prices/>` at the URI `mdex://documents/prices`.

```
fn:put(<prices/>, "mdex://documents/prices")
```

Subsequent queries can retrieve the `<prices/>` document using `fn:doc()` and specifying the same URI, as in the example code below:

```
fn:doc("mdex://documents/prices")
```

Chapter 6

Features Enabled with XQuery

This section contains topics describing how to implement various MDEX Engine features using the MDEX Engine API for XQuery (MAX) and the MDEX Engine Web service.

Using the MAX API for disabled refinements requests

You can use the MDEX Engine API for XQuery (MAX API) to issue navigation requests that let you display disabled refinements.

Disabled refinements represent those refinements that end users could reach if they were to remove top-level filters that have been already selected from their current navigation state. To compute disabled refinements, the MDEX Engine calculates the base navigation state and the default navigation state.

Note: For more information about disabled refinements, as well as about the base and default navigation states used to calculate disabled refinements, see the section "Working with Dimensions" in the *Basic Developer's Guide*.

You can display disabled refinements in your front-end application using the `DisabledRefine¬mentsConfig` data type that is described in one of the MAX API schema files, `mdex.xsd`, located in `ENDECA_MDEX_ROOT/conf/schema`.

Here is a snippet from the `mdex.xsd` file that describes the `DisabledRefinementsConfig` data type:

```
<complexType name="DisabledRefinementsConfig">
    <sequence>
      <element name="BaseDimensionIds" type="tns:DimensionValueIdList"
minOccurs="0" maxOccurs="1" />
    </sequence>
    <attribute name="EqlFilterInBase" type="boolean" use="optional" de¬
fault="false"/>
    <attribute name="TextSearchInBase" type="boolean" use="optional" de¬
fault="false"/>
    <attribute name="RangeFiltersInBase" type="boolean" use="optional" de¬
fault="false"/>
  </complexType>
```

You can see that for the `DisabledRefinementsConfig` type, you can specify the following aspects of the base navigation state:

- `BaseDimensionIDs` — this is the list of dimension IDs that should be included in the base navigation state

- `EqlFilterInBase` — this is the EQL filter that should be included in the base navigation state
- `TextSearchInBase` — this is the text search that should be included in the base navigation state
- `RangeFiltersInBase` — this is the range filter that should be included in the base navigation state

If the `DisabledRefinementsConfig` element is empty, or not included in the navigation query, this means that no configuration for disabled refinements is sent to the MDEX Engine.

For example, the following navigation query contains the `DisabledRefinementsConfig` element:

```
mdex:navigation-query(
  <Query xmlns="http://www.endeca.com/MDEX/data/IR600">
    <SelectedDimensionValueIds>
   <DimensionValueId>110001</DimensionValueId>
   <DimensionValueId>210001</DimensionValueId>
 </SelectedDimensionValueIds>
 <RefinementConfigs>
    <DisabledRefinementsConfig TextSearchInBase="true">
       <BaseDimensionIds>
         <DimensionValueId>100000</DimensionValueId>
       </BaseDimensionIds>
    </DisabledRefinementsConfig>
   <RefinementConfig id="400000"/>
 </RefinementConfigs>
 <Searches>
   <Search Key="All">television</Search>
 </Searches>
  </Query>
)
```

In this query, a text search and a dimension value ID are specified. This means that the MDEX Engine will include these top-level filters into its base navigation state.

Sending this query to the MDEX Engine through the MDEX Web service (with the SOAP UI or another Web services testing tool), results in the MDEX Engine response which contains the `DGraph.DisabledRefinement` property. You can locate this property in the MDEX Engine response and render the disabled refinements as grayed out in the front-end application.

Example: identifying disabled refinements from query output

Disabled refinements are returned in the same way as regular refinements. You can identify disabled refinements with the `DGraph.DisabledRefinement` property of the MDEX Engine.

In the MAX API output, you can locate this property in the dimension tree returned with the query results. For example, the following function determines whether results contain dimension values that are disabled refinements:

```
(:~
  This function determines whether $results indicate that
  the dimension with id $dimValId is a disabled refinement
:)
declare function isDisabledRefinement(
  $results as element(mdata:NavigationResults, xs:untyped),
  $dimValId as xs:string
) as xs:bool {
  let $dimVal := $results/mdata:Dimensions//mdata:DimensionValue[@Id=$dim¬
ValId]
  let $disabledRefinementProp :=
   $dimVal/mdata:Properties/mdata:Property[@Key='DGraph.DisabledRefinement']
  return
    if(not exists $disabledRefinementProp)
    then false()
    else
      data($disabledRefinementProp) eq '1'
}
```

Retrieving refinement counts for records that match descriptors

For each dimension that has been enabled to return refinement counts, the MDEX Engine returns refinement counts for records that match descriptors. Descriptors are selected dimension values in this navigation state.

The counts for descriptors are returned as part of the results in XML format that the MDEX Engine returns for a navigation query.

This capability of retrieving refinement counts for descriptors is the default behavior of the MDEX Engine. No additional configuration (for example, Dgraph command line options) is needed to enable this capability.

The count represents the number of records that match this dimension value in the current navigation state.

- For a multi-AND or a single-select dimension, this number is the same as the number of matching records.
- For a multi-OR dimension, this number is smaller than the total number of matching records if there are multiple selections from that dimension.

The following procedure shows how you can retrieve counts for descriptors using a custom module in XQuery that is in turn being used by the main XQuery module for making a navigation query. The examples illustrate the logic for retrieving this information from the MDEX Engine; However, you can organize your code in a different way to achieve similar results.

To access the refinement counts for descriptors:

1. Create a custom XQuery module, similar to the following example. The `example.xq` is a library module containing a custom function for retrieving descriptor counts from the output of a navigation query. This example uses the MDEX Engine XQuery API (MAX).

   This example does the following:

   - Identifies the dimension states
   - For each dimension state, finds descriptors for records and aggregated records
   - Finds `DGraph.Bins` and `Dgraph.AggrBins` dimension value properties for these specific dimension values. (`DGraph.Bins` and `Dgraph.AggrBins` are dimension value properties that are computed at query time).

   ```
   module namespace example = 'http://www.endeca.com/example';

   declare namespace mdata = 'http://www.endeca.com/MDEX/data/IR600';

   declare function example:property-value-for-dval(
       $dimName as xs:string,
       $id as xs:string,
   ```

```
    $dimensions as element(mdata:Dimensions, xs:untyped),
    $propName as xs:string
) as xs:string?
{
  let $dval := $dimensions/mdata:Dimension[@Name = $dimName]//mdata:Di¬
mensionValue[@Id = $id]
  let $prop := $dval/mdata:Properties/mdata:Property[@Key = $propName]
  return string(zero-or-one($prop))
};

declare function example:descriptors-with-counts(
    $results as element(mdata:NavigationResults, xs:untyped)
) as element(example:descriptors, xs:untyped)
{
  <example:descriptors>{
    for $dimState in $results/mdata:NavigationStatesResult/mdata:Dimen¬
sionStates/mdata:DimensionState
    let $dimName := data($dimState/@DimensionName)
    for $descriptor in $dimState/mdata:SelectedDimensionValues/mdata:Di¬
mensionValueReference
    let $dvalId := data($descriptor/@Id)
    return
    <example:descriptor>{
      attribute DimensionName { $dimName },
      attribute Id { $dvalId },
      attribute Name { data($descriptor/@Name) },
      attribute RecordCount {
        example:property-value-for-dval(
          exactly-one($dimName),
          exactly-one($dvalId),
          exactly-one($results/mdata:Dimensions),
          'DGraph.Bins'
        )
      },
      attribute AggrRecordCount {
        example:property-value-for-dval(
          exactly-one($dimName),
          exactly-one($dvalId),
          exactly-one($results/mdata:Dimensions),
          'DGraph.AggrBins'
        )
      }
    }</example:descriptor>
  }</example:descriptors>
};
```

Once you have the `example.xq` library module, you can use it as part of the main module in XQuery which you use for making a navigation query.

2. Declare and use the `example.xq` module as part of the main module. This can be done as shown in the following example of the `example-main.xq` main module:

```
import module namespace mdex = "http://www.endeca.com/XQuery/mdex/2008"
 at "mdex.xq";
import module namespace example = "http://www.endeca.com/example" at
"example.xq";

declare namespace mdata = "http://www.endeca.com/MDEX/data/IR600";

let $result := mdex:navigation-query(
<Query xmlns="http://www.endeca.com/MDEX/data/IR600">
```

```
  <RefinementConfigs ExposeAllRefinements="true">
    <RefinementConfig DimensionValueId="5" Expose="true" LimitDimension¬
Values="false" />
  </RefinementConfigs>
  <SelectedDimensionValueIds>
    <DimensionValueId>4294967187</DimensionValueId>
    <DimensionValueId>4294967250</DimensionValueId>
  </SelectedDimensionValueIds>
</Query>
)
return example:descriptors-with-counts($result)
```

3. Use the `example-main.xq` module to make a navigation query to the MDEX Engine using the packaged MDEX Query Web service.

   Depending on your development environment, you can use any of the Web services testing tools to run this query with the Endeca MDEX Query Web service.

4. Examine the results of the navigation query returned by the `example-main.xq` module. The results are returned by the MDEX Engine in XML and may look as follows:

```
<descriptors xmlns="http://www.endeca.com/example">
  <descriptor DimensionName="Body" Id="4294967250" Name="Ripe" Record¬
Count="12493" AggrRecordCount=""/>
  <descriptor DimensionName="Body" Id="4294967187" Name="Smooth" Record¬
Count="4461" AggrRecordCount=""/>
</descriptors>
```

You can now use the returned counts for records that match descriptors in your front-end application.

Issuing dimension value boost requests

The MAX API supports making requests that boost or bury returned refinements.

Dimension value boost and bury is an MDEX Engine feature that lets you specify, in a request, that certain dimension values should be sorted higher (boosted) or lower (buried) than other dimension values. The feature is documented in the *Basic Development Guide*.

The `DimensionValueStratum` data type represents the assignment of a dimension value to a specific stratum:

```
<complexType name="DimensionValueStratum">
  <attribute name="DimensionValueId" type="tns:DimensionValueId" use="re¬
quired" />
  <attribute name="Stratum" type="int" use="required" />
</complexType>
```

The `DimensionValueId` attribute specifies the ID of the dimension value to be assigned to a stratum, while the `Stratum` attribute is an integer that represents the stratum to which the dimension value is assigned.

The `DimensionValueStratumList` data type contains a list of stratified dimension values (`Dimen¬ sionValueStratum` elements):

```
<complexType name="DimensionValueStratumList">
  <sequence>
    <element name="DimensionValueStratum" type="tns:DimensionValueStratum"

       minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>
```

In turn, the `NavigationQuery` data type takes a `DimensionValueStrata` element that specifies the list of stratified dimension values:

```
<complexType name="NavigationQuery">
  <all minOccurs="0">
  ...
     <element name="DimensionValueStrata" type="tns:DimensionValueStratum¬
List" minOccurs="0"/>
  </all>
  ...
</complexType>
```

Thus, a navigation query for dimension value boost would contain a `DimensionValueStrata` element similar to this example:

```
<mdata:NavigationQuery>
  ...
  <mdata:DimensionValueStrata>
    <mdata:DimensionValueStratum DimensionValueId="405" Stratum="2"/>
    <mdata:DimensionValueStratum DimensionValueId="406" Stratum="2"/>
    <mdata:DimensionValueStratum DimensionValueId="409" Stratum="1"/>
    <mdata:DimensionValueStratum DimensionValueId="410" Stratum="1"/>
  </mdata:DimensionValueStrata>
</mdata:NavigationQuery>
```

The request creates two strata (stratum 2 and stratum 1) and assigns two dimension values to each of them.

Index