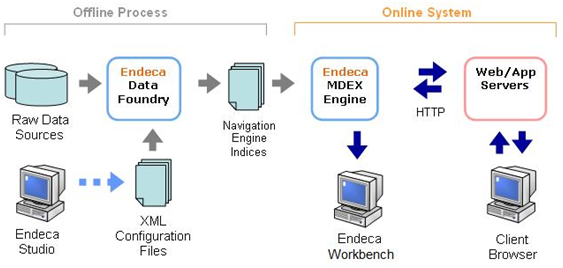
# Overview

**Project Overview**Motorcycle Superstore has deployed the Endeca IAP solution that provides search, Guided Navigation, and dynamic merchandising functionality on the Motorcycle Superstore site. Motorcycle Superstore engaged with Thanx Media Professional Services for guidance Finish(of the Data Foundry and pipeline processes for the creation of the indices that is utilized by the sites). The user interface portion and control scripts for the application were developed by Motorcycle Superstore to interact with the MDEX Engine through the Endeca Assembler API. **About the Technical Specification Document**The purpose of this document is to outline the implementation details for the Endeca IAP instance reflecting the requirements defined through a Requirements Workshop and subsequently captured in the Motorcycle Superstore Business Requirements document. The Technical Specification document contains information about Endeca, index creation, custom scripts, and operational processes through control scripts and the Endeca Workbench.

# Overview of Endeca Technology

**Endeca Architecture Overview**The basic Endeca Architecture consists of four (4) main parts as illustrated in the diagram: the Endeca Data Foundry (EDF), the Endeca MDEX Engine (MDEX), the Endeca Application Controller (EAC), and the Endeca API.



**Endeca Data Foundry (EDF)**The EDF is an offline process that first reads in the source data and application specific configuration files. Using the instructions outlined in the configuration files; the EDF transforms the raw data into structured data and indexes it. The output of the EDF is a set of indices (binary and xml files) which are provided to the Endeca MDEX Engines. This is typically a scheduled offline process that runs, and therefore happens independently of the MDEX Engines handling the runtime queries. **Endeca MDEX Engine (MDEX)**The MDEX is an online server that listens on a specific port for HTTP requests, and responds by returning binary results. Once a new set of indices is created from the EDF, the MDEX will load the indices into memory at process restart. The MDEX is also stateless, with no concept of a session object for each user. This allows multiple MDEX engines with the same set of indices to return the same results, allowing for easy scalability and redundancy behind a load balancer. **Endeca Application Controller**The Endeca Application Controller consists of the Central Server and Agents. The EAC is installed and runs on the primary Endeca server, creating a single interface for coordinating jobs across all machines. Agents are installed on each Endeca server and are responsible for the actual work of an Endeca implementation. Communication to these agents is performed through the EAC Central Server via eaccmd, a command line interface, or Workbench. **Endeca API**The Endeca Presentation Layer Java API can be installed on any application server that supports Java. The API is used to create and send the query to the MDEX Engine over HTTP. It receives the response as a binary encoded result object and uses the appropriate methods and objects to display the information in the UI. The query must be submitted to a pre-defined host and port. If only one MDEX engine exists, the host and port of the MDEX engine should be configured at the UI level. If multiple MDEX engines exist behind a load balancer, the host and port of the load balancer should be configured at the UI level.  **Implementation Architecture**Architecture of the Endeca cluster will follow the Oracle Endeca best practice of Authoring/Live clusters. Authoring environments will be run with Live data, and will serve as the Endeca Workbench test platform. An overview of the environment can be found in section 16.

# Data Definition

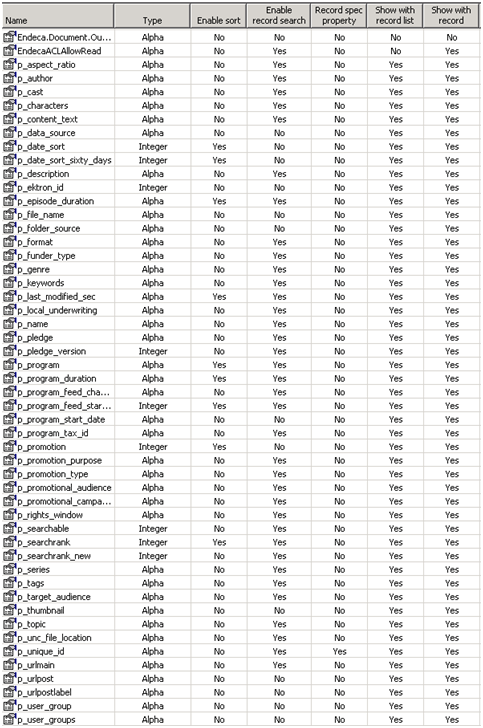
**Data Sources**There are (2) main data types that make up the Motorcycle Superstore site indices:

* Records Feed: Provides information that will be used for search, search results, and navigational purposes throughout the site.
* PS Record Data Feed: Provides program information that will also be used for search, search results, and navigational purposes throughout the site.
* Forum Data Feed: Provides the category information that will be used to display category information and setup the Product Category hierarchy for navigation.
* Search Rank Data Feed - Provides the category that will be used to display category information and setup the Product Category hierarchy for navigation.

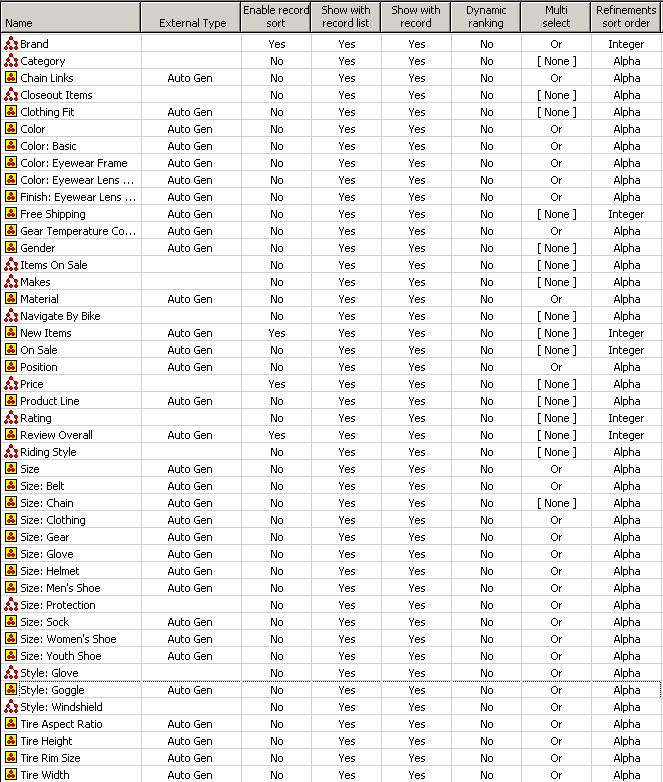
**Data Pre-processing**All data pre-processing is performed before the data is loaded into the C:/Daniel/apps/pbsconnect/data/ektron/full/ directory. Before the data is fed into the pipeline, Thanx has implemented several python scripts to clean and check the data. C:/Daniel/apps/mag/

|  |  |
| --- | --- |
| Custom Scripts | Operation |
|  |  |
|  |  |

EXAMPLE: Thanx has implemented a custom thesaurus script createThesaurus.bat which converts the thesaurus entries to the Endeca required format and feeds it directly into the IFCR. **Endeca IAP Data Structures  
  
Properties**The screenshot defines the properties to be created for this implementation.



**Dimensions**The table below defines the dimensions to be created for this implementation. The dimensions are dynamically generated from the Motorcycle Superstore database and the Attribute dimensions list will change over time. Currently there are 27 dimensions defined and these match the list of attribute values defined in this section. Dimensions configuration is defined within the api\_input configuration files, defined below.



**Dimension Groups**No dimension groups have been designated for configuration at this time. Here is where the list of the groups and the dimensions can be defined:

|  |  |
| --- | --- |
| Group | Dimensions |
|  |  |

**Record Filtering**Record filters allow an Endeca application to define subsets of the total record set and dynamically limit search and navigation results to these subsets. Record filters are applied before any search processing. The result is that the search query is performed as if the data set only contained records allowed by the record filter.  
  
Record filters are applied when query is sent into an Endeca MDEX engine. Record filters are used for the site variable and repository, but are not required since this is a single site implementation.  **Filter List**All dimension and properties can be filtered on. Dimensions are always filterable; the properties listed below are currently configured for filtering.  
  
**Current Filterable Properties**

|  |
| --- |
| Properties/Dimensions |
|  |

**Language Configuration**

**Dimension Translations**No dimension translation is necessary at this time; all dimensions will appear in English.

**Guided Navigation**Guided Navigation is enabled when dimensions are configured in Developer Studio.  
  
**Attribute Navigation**Multiple attributes exist for every product item. For this implementation, most of these attributes are auto-generated and auto-created to be available for refinements.  
 **Precedence Rules**

|  |  |
| --- | --- |
| Group | Dimensions |
|  |  |
|  |  |

**Category Matching**Dimension values are returned when a keyword search is performed. These allow shortcut links to the selected navigation states.  
  
Enabling Dimension Search via Developer Studio for the dimensions above will automatically activate this feature. The default match mode to use is  
  
MatchAll  
  
Currently all of the dimensions except 'Programs' have been enabled for Dimension Search.   
 **Keyword Search  
Search Interfaces**A search interface is a named collection of properties and dimensions, each of which is enabled for record search in Developer Studio. A search interface may also contain an ordered collection of one or more ordering strategies. When a user passes a search term (or terms) to a search interface, the Endeca IAP will search the term(s) against all the properties and dimensions listed in the search interface.  
A search interface allows you to control record search behavior for groups of one or more properties or dimensions. Some of the features that can be specified for a search interface include relevance ranking, matching across multiple properties and dimensions, keyword in context results, and partial match.  
**All Search Interface**The 'all' Search Interface is responsible for the initial search of records in the product catalog via a keyword search. The records will be searched for matches against the properties defined below.

|  |
| --- |
| Searchable Properties and Dimensions |
| primary |
| p\_sku |
| Category |
| p\_name |
| Brand |
| Collection |
| Color |
| Sports |
| p\_upc |
| p\_keyword |
| category\_top |
| p\_unique\_id |
| p\_sku |
| p\_upc |
| Category |
| p\_name |
| Brand |
| p\_keyword |
| p\_description\_short |
|  |

**Relevance Ranking**Relevance ranking modules provide a level of control over the order in which search results are displayed to the end user of an Endeca IAP application. Changes to relevance ranking configurations are an important part of on-going search and performance tuning and will continue post-launch. This section describes the strategy for a retail catalog data set. It assumes the following:The table below defines the relevance ranking modules to be configured for the search interface. A static module is normally recommended as the final module, in this instance, Motorcycle Superstore has added the date\_sort\_sixty\_days as the static module in the 'All' search interface, and searchrank as the static module in the 'search\_rank\_topic' search interface.

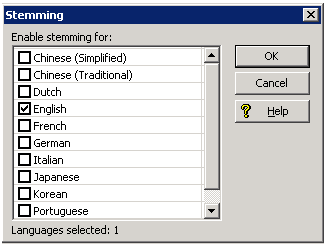
* The search mode is MatchAllPartial. By using this mode, you ensure a users search would return a two-words-out-of-five match as well as a four-words-out-of-five match, just at a lower priority. (For details, see the section 12.3.1: Match Mode).
* The strategy is based on a search interface with members such as Name, Category, and Description, in that order. The order is significant because a match on the first member ranks more highly than a cross-field match or match on the second or third member.

|  |  |
| --- | --- |
| Module Name | Module Description |
| NumFields | NumFields has been added to match the current search strategy currently in place on Motorcycle Superstore. Numfields ranks records based on the number of fields that match a query completely. |
| NTerms | Ensures in a multi-word search the more words that match the better. For example, in a three-word query, results that match all three words will be ranked above results that match only two, which will be ranked above results that match only one. |
| MaxField | Puts cross-field matches as high in priority as possible. In other words, the more searchable fields that match the customer's search term, the better, as these results will have a higher priority in the results list. |
| Glom | Decomposes cross-field matches, effectively breaking any ties resulting from MaxField. Together, MaxField and Glom provide the proper ordering, depending upon what matched. |
| Phrase | Considers results containing the user's query as an exact phrase, or a subset of the exact phrase more relevant than matches simply containing the user's search terms scattered throughout the text. |

**Record Matching**It is important to give careful consideration to the possible search terms a user might enter in the context of the data being searched. The features described below impact the way records are retrieved upon execution of a search query.  
  
This section outlines the matching configurations to be used in the Motorcycle Superstore implementation.  
  
**Match Mode**Match modes control how many of the user's search terms within a multiple-term query are considered by the Endeca Navigation Engine.  
  
The default search mode is MatchAllPartial. This mode attempts to match all terms in a users search, and returns only those results if they are available. If there are no records that match all the terms entered, those records that match at least a certain number of terms will be returned instead. This allows the accuracy of MatchAll mode when applicable, but falls back to a broader MatchPartial mode when appropriate.  
  
The MatchAllPartial search mode can be adjusted by setting the 'Match at least' and 'Omit at most' values for the search interface. 'Match at least' can be adjusted to broaden the search. By default, this is set to two to help avoid false positive matches. When set to one, meaning that at least one of the terms the user entered must match in a result, which would broaden the results.  
  
'Omit at most' can also be adjusted. By default, this is set to two, also, to help avoid false positive matches. This can be set to zero, meaning that any number of terms can be ignored from a given query.  
  
 Motorcycle Superstore has the following configured:

|  |
| --- |
| Configuration |
| Match at least 2 words |
| Omit at most 8 words |

**Auto-correct/Did You Mean**Auto-correct spelling correction operates by computing alternate spellings for user query terms, evaluating the likelihood that these alternate spellings are the best interpretation, and then using the best alternate spell-corrected query forms to return search results.  
  
For example, a user might search for records containing the text 'frme'. With spelling correction enabled, the Endeca MDEX Engine will return the expected results: those records containing the text 'frame'.  
  
The auto-correct capability is enabled for this implementation.  
  
Auto-correct is enabled for this application. If the original search criteria returns zero results, an auto-correction will be attempted. If the auto-corrected term returns results, those results will be displayed. A message 'No results were found for '[original term]' Results for '[corrected term]' are shown.' indicating the auto-corrected search terms will also be displayed.  
  
Auto-correct is enabled with a dgraph flag (below), which is set in the control scripts and/or the Manager Settings in Developer's Studio.  
  
--spl  
  
'Did You Mean' is related to auto-correct. While auto-correct is engaged when no results are found for the current search, 'Did You Mean' is utilized when less than 5 results were found for the current search yet more results would be produced had the user searched for a term of similar spelling. In this case, the results of the original search are returned and a link displaying 'Did you mean [DidYouMean term]?' which allows the user to view the results of the 'Did You Mean' term.  
  
'Did You Mean' is enabled with a dgraph flag (below), which is set in the control scripts and/or the Manager Settings in Developer's Studio.  
  
--dym  
 **Thesaurus**The thesaurus feature provides the ability to configure rules for matching queries to text containing equivalent words or concepts. The thesaurus is intended for specifying concept-level mappings between words and phrases. Thesaurus configurations are an important part of on-going search tuning and will occur post-launch.  
  
The thesaurus entries implemented are defined in the Requirements Definition documentation.  
  
 **Stemming**The stemming feature broadens search results to include word roots and word derivations. Stemming is intended to allow words with a common root form (such as the singular and plural forms of nouns) to be considered interchangeable in search operations. For example, search results for the word "lure" will include derivations such as "lur" and "lures,"while a search for "lures" will also include its word root "lure".  
  
Stemming functionality utilizes a predefined set of word forms and is not configurable. For this implementation, stemming will be enabled.   
  
Stemming will need to be enabled for each Endeca application depending on the language. This is configured though Endeca's Developer Studio by selecting Edit > Stemming from the menu.



**Stop Words**  
Stop words are words set to be ignored by the Endeca MDEX Engine. Typically, common words like "the" are included in the stop word list. In addition, you might want to add terms that are prevalent in your data set. For example, if your data consists of lists of books, you might want to add the word "book" itself to the stop word list, since a search on that word would return an impracticably large set of records.  
For the Motorcycle Superstore implementation, no words are enabled as stop words. Additional stops words can be configured with Developer Studio in the "Search Configuration" section.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

**Search Characters**Upper- and lower-case letters and the digits 0 to 9 are automatically included as valid search characters in your Endeca-enabled application. However, in the case of other characters, such as certain punctuation characters, you can specify whether the character should be indexed along with alphanumeric characters in a token or instead treated as white space.  
  
Currently there are no additional search characters set up in the pipeline.  
  
**Wildcard Search**Wildcard search allows substring matches to be considered when a query is executed, where the user may enter an asterisk ("\*") operator as wildcard character. For example, a search on 123\* will return all items that contain 1234, 12367, 123A, etc.

|  |
| --- |
| p\_description\_short |
| p\_sku |
| p\_upc |
|  |

**Phrasing**Phrasing functionality allows words that should be considered as a group instead of as individual search terms to be configured accordingly. If a user's search terms include one of these phrases, the phrase is identified and matches are only considered for those results containing the phrase, as opposed to containing the terms in any order and proximity. For example, a search for "[SAMPLE PHRASE]" will return all records that contain the terms "[SAMPLE PHRASE]" in a single search field.  
  
**Aggregate Records**By configuring aggregated records, the application is to display a group of multiple records as though it were a single record in the results list, based on the value of the rollup key.  
  
Motorcycle Superstore does not currently have a property that can be used to aggregate records.  
  
**Search Results and Display**In addition to the actual records returned by a search query, other elements that can impact the user's experience are also made available to the application rendering the results. These include the ability to sort the results, highlighting blocks of text in the resulting records that contain the search term, as well as "Did You Mean" functionality.  
  
This section outlines the display configurations used in the Motorcycle Superstore Implementation.  
  
**Did You Mean**"Did You Mean" functionality provides an application with the capability to provide the user with explicit alternative suggestions for a keyword search. For example, if a user searches for "clump" in the Motorcycle Superstore data, there will be one result. The term "clamp" however, is much more prevalent (97 results). When this feature is enabled, the MDEX Engine will respond with the one result for "clump" but will also suggest that "clamp" may be what the end-user actually intended. If multiple suggestions are returned, they will be sorted and presented according to the closeness of the match.  
  
A threshold can be set where the system will look for alternate results close to what a customer is looking for. The default threshold is 5 results, and this setting will not be changed for this implementation. If the system finds an appropriate alternative that has more results than your initial search term, it will suggest it. Once the user clicks on the "Did You Mean" term, product results are returned.  
  
**Snippeting**The snippeting feature (also referred to as keyword in context) provides the ability to return an excerpt from a record--a snippet--to an application user who performs a record search query. A snippet contains the search terms that the user provides along with a portion of the term's surrounding content to provide context.  
  
Motorcycle Superstore has indicated that snippeting feature will be enabled for the body field.**Sorting**Search results will be returned to the User Interface based upon relevance ranking. The user will then be allowed to sort those results by the following:

|  |
| --- |
| p\_price\_sort |
| p\_is\_new |
| p\_metric\_total\_units\_sold |
| p\_metric\_profit\_margin |
| p\_metric\_sales\_velocity |
| p\_metric\_days\_added |
| p\_metric\_inventory |
|  |

# Merchandising Workbench Configuration

Two merchandising opportunities will be provided within the Endeca Workbench.  
  
Banners - this is a merchandising zone that allows the business user to add links to visual images, destination urls and/or html (for rotating banners).  
  
Product spotlighting - this is a merchandising zone that allows the business user to select Programs, Tags, and Author to be highlighted within a special section on the page.  
  
**Keyword Redirects**A keyword redirect is a search term-URL pair that defines where a user will be redirected to upon entering a search term. For example, if the user searches for "shipping," the number of records returned may be small (or the search will produce no results), so it may be more appropriate to automatically redirect the user to a page about shipping.  
  
Currently there are no keyword redirects configured within the Developer Studio. Future additions should be added to the table below.

|  |  |
| --- | --- |
| Search Term | Redirect URL |
|  |  |
|  |  |

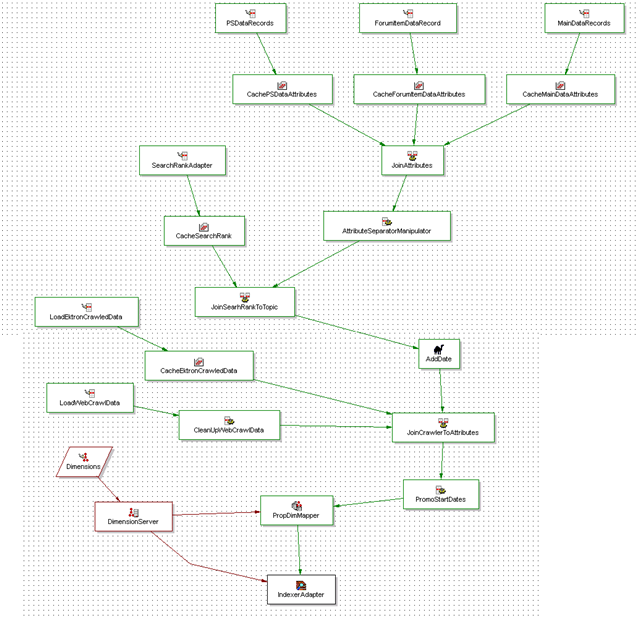
**Rule Groups**Rule Groups organize the large number of merchandising rules into smaller logical categories in order to allow multiple users to access the Endeca Workbench simultaneously. Rule Groups can be aligned to a specific department, team, template or area of the page.  
  
The default allowable rule groups for Guides search is listed below:

|  |  |
| --- | --- |
| Rule Groups |  |
|  |  |
|  |  |

**Cartridge Type**A cartridge is a defined area on a page that will contain Merchandising Rules. The merchandising cartridges:

|  |  |
| --- | --- |
| Name | Max Records |
|  |  |
|  |  |

The cartridges and the expected behavior need to be defined within the User Interface to properly render the supplemental content from a triggered rule. Please see the section 10: "User Interface" for a reference that provides more details on this.  
  
**Rules**Once cartridges have been configured, the merchandising rules themselves can be configured. These rules can be created in Developer Studio, or by business users in the Workbench.   
  
**Pipeline Configuration**The pipeline defines the import, conversion, and mapping of source data to Endeca IAP Properties and Dimensions. Motorcycle Superstore will have one pipeline indexing the product catalog. The pipeline is generated using the Endeca 2.1.2 Product Catalog Integration deployment template. No changes to default pipeline layout were made.  
  
**Pipeline Diagram**Details of the pipeline diagram are outlined below.



The pipeline diagrams shown above will be leveraged to index content and product data. The content data will be read into a separate Endeca application.  
  
**Pipeline Details  
Record Adapters**The table below describes the record adapters to be used for loading of the product data. A record adapter is defined for each input source.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Adapter Name | Type | Class | Class Path/File Name | Pass Throughs |
| LoadMainData | Delimited( ) | N/A | items.txt | N/A |
| LoadMainAttributes | Delimited( ) | N/A | attributes.txt | N/A |
|  |  |  |  |  |

**Record Manipulators**No record manipulations of the source data are required in order to properly load the documents and extract the content. At this time no record manipulators will be used.

|  |  |  |
| --- | --- | --- |
| Manipulator Name | Record Source | Manipulations |
| AddPrimaryDimManipulator | JoinAttributesToMain | No Comment Provided - Add Manually |
| FlattenMainAttributes | LoadMainAttributes | No Comment Provided - Add Manually |
|  |  |  |

**Record Assemblers**The record assembler component will define the join between the various data extracts. No joins are necessary.

|  |  |
| --- | --- |
| CacheMainAttributes | [Sources] |
| LoadMainData | [Sources] |
| JoinAttributesToMain | [Assembler Name] |
| LEFT\_JOIN | [Join Type] |
|  |  |

**Record Caches**The table below describes the record caches to be used for preparing data to join. A record cache is defined for each data set that is joined to the main record set. Since no joins are needed, no record caches exist for this pipeline.

|  |  |  |  |
| --- | --- | --- | --- |
| Cache Name | General | Sources | Record Index |
| CacheMainAttributes | -1(Combine Recs) | FlattenMainAttributes | unique\_id |
|  |  |  |  |

**Property Mapper**The property mapper is responsible for mapping the source properties (from the data extracts themselves or created through a record manipulator) to Endeca IAP dimensions and properties. The table below identifies the mappings that will be required.  
  
Full source data is needed in order to fill in this information.

|  |  |  |  |
| --- | --- | --- | --- |
| Source | Target | Target Type | Match Mode |
| P\_Endeca | Endeca | Dimension | NORMAL |
| brand | p\_brand | Property | N/A |
| description\_long | p\_description\_long | Property | N/A |
| description\_short | p\_description\_short | Property | N/A |
| image | p\_image | Property | N/A |
| is\_free\_shipping | p\_is\_free\_shipping | Property | N/A |
| is\_new | p\_is\_new | Property | N/A |
| is\_on\_sale | p\_is\_on\_sale | Property | N/A |
| keyword | p\_keyword | Property | N/A |
| name | p\_name | Property | N/A |
| price\_retail | p\_price\_retail | Property | N/A |
| price\_sale | p\_price\_sale | Property | N/A |
| price\_sort | p\_price\_sort | Property | N/A |
| sku | p\_sku | Property | N/A |
| sort\_default | p\_sort\_default | Property | N/A |
| sort\_rating | p\_sort\_rating | Property | N/A |
| unique\_id | p\_unique\_id | Property | N/A |
| url\_detail | p\_url\_detail | Property | N/A |
| brand | Brand | Dimension | AUTO\_GEN |
| category\_id | Category | Dimension | NORMAL |
| category\_id | p\_category\_id | Property | N/A |
| is\_free\_shipping | Free Shipping | Dimension | NORMAL |
| is\_new | New | Dimension | NORMAL |
| is\_on\_sale | On Sale | Dimension | NORMAL |
| price\_sort | Price Range | Dimension | NORMAL |
| group\_id | p\_group\_id | Property | N/A |
| is\_custom\_0 | p\_is\_custom\_0 | Property | N/A |
| is\_custom\_1 | p\_is\_custom\_1 | Property | N/A |
| is\_custom\_2 | p\_is\_custom\_2 | Property | N/A |
| is\_custom\_3 | p\_is\_custom\_3 | Property | N/A |
| is\_custom\_4 | p\_is\_custom\_4 | Property | N/A |
| is\_custom\_5 | p\_is\_custom\_5 | Property | N/A |
| metric\_days\_added | p\_metric\_days\_added | Property | N/A |
| metric\_inventory | p\_metric\_inventory | Property | N/A |
| metric\_pct\_bounceback | p\_metric\_pct\_bounceback | Property | N/A |
| metric\_pct\_conversion | p\_metric\_pct\_conversion | Property | N/A |
| metric\_pct\_on\_sale | p\_metric\_pct\_on\_sale | Property | N/A |
| metric\_profit\_dollars | p\_metric\_profit\_dollars | Property | N/A |
| metric\_profit\_margin | p\_metric\_profit\_margin | Property | N/A |
| metric\_sales\_velocity | p\_metric\_sales\_velocity | Property | N/A |
| metric\_total\_detail\_views | p\_metric\_total\_detail\_views | Property | N/A |
| metric\_total\_units\_sold | p\_metric\_total\_units\_sold | Property | N/A |
| search\_alpha\_0 | p\_search\_alpha\_0 | Property | N/A |
| search\_alpha\_1 | p\_search\_alpha\_1 | Property | N/A |
| search\_alpha\_2 | p\_search\_alpha\_2 | Property | N/A |
| search\_alpha\_3 | p\_search\_alpha\_3 | Property | N/A |
| search\_alpha\_4 | p\_search\_alpha\_4 | Property | N/A |
| search\_alpha\_5 | p\_search\_alpha\_5 | Property | N/A |
| search\_alpha\_6 | p\_search\_alpha\_6 | Property | N/A |
| search\_alpha\_7 | p\_search\_alpha\_7 | Property | N/A |
| search\_alpha\_8 | p\_search\_alpha\_8 | Property | N/A |
| search\_alpha\_9 | p\_search\_alpha\_9 | Property | N/A |
| sort\_alpha\_0 | p\_sort\_alpha\_0 | Property | N/A |
| sort\_alpha\_1 | p\_sort\_alpha\_1 | Property | N/A |
| sort\_alpha\_2 | p\_sort\_alpha\_2 | Property | N/A |
| sort\_alpha\_3 | p\_sort\_alpha\_3 | Property | N/A |
| sort\_alpha\_4 | p\_sort\_alpha\_4 | Property | N/A |
| sort\_alpha\_5 | p\_sort\_alpha\_5 | Property | N/A |
| sort\_float\_0 | p\_sort\_float\_0 | Property | N/A |
| sort\_float\_1 | p\_sort\_float\_1 | Property | N/A |
| sort\_float\_2 | p\_sort\_float\_2 | Property | N/A |
| sort\_float\_3 | p\_sort\_float\_3 | Property | N/A |
| sort\_float\_4 | p\_sort\_float\_4 | Property | N/A |
| sort\_float\_5 | p\_sort\_float\_5 | Property | N/A |
| sort\_integer\_0 | p\_sort\_integer\_0 | Property | N/A |
| sort\_integer\_1 | p\_sort\_integer\_1 | Property | N/A |
| sort\_integer\_2 | p\_sort\_integer\_2 | Property | N/A |
| sort\_integer\_3 | p\_sort\_integer\_3 | Property | N/A |
| sort\_integer\_4 | p\_sort\_integer\_4 | Property | N/A |
| sort\_integer\_5 | p\_sort\_integer\_5 | Property | N/A |
| accessorySeason | Season | Dimension | AUTO\_GEN |
| accessorySports | Sports | Dimension | AUTO\_GEN |
| accessoryType | Accessory Type | Dimension | AUTO\_GEN |
| animalHeight | Height | Dimension | AUTO\_GEN |
| animalSkinType | Skin Type | Dimension | AUTO\_GEN |
| animalWeight | Weight | Dimension | AUTO\_GEN |
| collection | Collection | Dimension | AUTO\_GEN |
| color | Color | Dimension | AUTO\_GEN |
| gender | Gender | Dimension | AUTO\_GEN |
| occasion | Occasion | Dimension | AUTO\_GEN |
| type | Type | Dimension | AUTO\_GEN |
| virtualBearSubtype | Virtual Bear | Dimension | AUTO\_GEN |
| exclusive | Exclusive | Dimension | AUTO\_GEN |
| large\_image | p\_large\_image | Property | N/A |
| small\_image | p\_small\_image | Property | N/A |
|  |  |  |  |

**Dimension Server**A standard dimension server will be used in this pipeline.

|  |  |  |
| --- | --- | --- |
| Name | Format | URL |
| DimensionServer | XML | ../data/state/autogen\_dimensions.xml.gz |

**Indexer Adapter**A standard indexer adapter will be used in this pipeline.

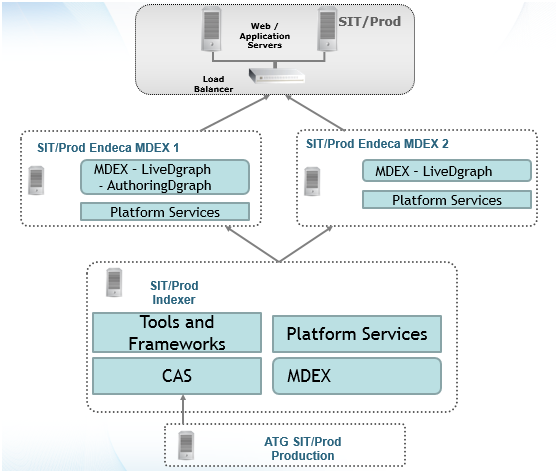
|  |  |  |
| --- | --- | --- |
| Name | URL | Output Prefix |
| IndexerAdapter | ../forge\_output/ | Motorcycle Superstore |

**User Interface**The user interface of the Motorcycle Superstore implementation will make use of Endeca Search, Guided Navigation, and Merchandising functionality. The application look-and-feel will be based on Motorcycle Superstore design. Motorcycle Superstore will use the Endeca Assembler API Java objects for querying information from the Endeca engine.  
  
**Logging**Logging should be performed every time a meaningful query is made to the MDEX Engine, which includes queries performed for the results list and the record details page. The logging API is separate from the navigation API. The Log Server, which accepts logging requests, is independent of the MDEX Engine(s).  
  
The Endeca logging queries should be made from the same application that will make the Endeca api calls. This should probably take place in the Motorcycle Superstore user interface.  
  
The "logserver" and "logport" values will be the same for all requests per Endeca Application. Each Endeca Application will have its own logging server.  
  
The various parameters for logging can be designated within the User Interface. The entries can include the following parameters:

|  |  |
| --- | --- |
| Name | Values |
| SESSION\_ID | Logging a session id allows the reporting engine to tie log entries to a session and report on per session statistics. |
| CONVERTED | Set to "TRUE" when a record has been converted. Conversion means different things for different applications. For e-commerce applications, it usually means that the product has been ordered or added to a shopping cart. Only SESSION\_ID and CONVERTED are needed for these log entries. |
| NUM\_RECORDS | The number of records returned. |
| TYPE | S - Search only N - Navigation only SN - Search then Navigation R - Record request T - Root request (N=0) |
| SEARCH\_KEY | The property or search interface searched against. |
| SEARCH\_TERMS | The search terms the user entered. |
| SEARCH\_MODE | MatchAllPartial or otherwise. |
| SORT\_KEY | Sort key used if sorting was applied. |
| DVALS | A list of dimension values used for refinement. These values are usually logged as human readable strings, rather than DVAL\_IDs. |
| DIMS | A list of dimension names used for refinement. |
| NUMREFINEMENTS | The number of refinements performed. |
| RECORD\_NAMES | The record names returned by the query. |
| DYM\_TO | Did You Mean terms triggered. |
| AUTOCORRECT\_TO | Auto-correction term triggered. |
| MERCH\_RULES | Merchandising rules that fired. |
| IN\_MERCH | Set to "1" if the record selected came from a merchandising result. |
| IN\_DYM | Set to "1" if the user clicked on a Did You Mean. |
| IN\_DIM\_SEARCH | Set to "1" if the user clicked on a dimension search result. |
| PAGE\_TIME | Record the page render time in milliseconds. Requires that the start time be recorded when the page begins loading. |
| ENE\_TIME | Record the ENE query time in milliseconds. Requires that the start time be recorded before the Endeca query is called. |
|  |  |

# Software Setup and Initial Configuration

This section provides the details on the physical infrastructure of the implementation, including the initial software setup and configuration of the application.  
  
**Physical Environments**  
The Motorcycle Superstore implementation encompasses the server environments below. Please refer to section 17.7: "Migration of Configuration Files between Environments" for details on migration of the Endeca configuration between environments.  
  
Overall design of the cluster will be the same for the Dev, Test and Production environments. The diagram below illustrates the overall design, and the installed Endeca software on each server.



Data is exported from ATG through CAS to the Indexer. The index is built during the baseline process and is deployed out to the MDEX servers via EAC. Each MDEX server hosts one instance of the Live dgraph and only one server hosts the Authoring dgraph. All workbench changes are immediately deployed to Authoring, and are promoted to Live via the promote\_content.bat script in the control directory.   
  
**Development Environment**  
The dev environment is a single server environment providing the Data Foundry services, and MDEX Engine to serve query requests to the web/application servers in the Motorcycle Superstore dev environment, the Endeca Workbench, and Logging / Report Generator server. The server hostname is Endeca-DEV-01.  
  
**Test Environment**The test environment is a single server environment providing the Data Foundry services, and MDEX Engine to serve query requests to the web/application servers in the Motorcycle Superstore test environment, the Endeca Workbench, and Logging / Report Generator server. The server hostname is Endeca-TEST-01.  
 **Production Environment**The production environment is a single (3) server environment providing the Data Foundry services, and MDEX Engine to serve query requests to the web/application servers in the Motorcycle Superstore test environment, the Endeca Workbench, and Logging / Report Generator server. The indexing server hostname is ENDECA-PROD-01.HQ.CORP.PBS.ORG.

* Endeca-DEV-01
* Endeca-TEST-01
* ENDECA-PROD-01.HQ.CORP.PBS.ORG
* ENDECA-PROD-02.HQ.CORP.PBS.ORG
* ENDECA-PROD-03.HQ.CORP.PBS.ORG

**Software Installation**Thanx Media performed the installation of the Endeca software on all servers under the E: partition. The default installation directory for all environments will be:

|  |
| --- |
| Directory |
| C:/Daniel/ |

Please refer to the Endeca Installation Guide for details on installation of the Endeca software.  
  
**Users and Permissions**Most of the Endeca software should be installed and all Endeca processes run as the "Administrator" user or equivalent. There may be permission issues during baseline updates if files are owned by different users.  
  
Each of the Motorcycle Superstore servers running an Endeca process have user accounts created for the Thanx Media implementation team to provide access to all machines in all the Endeca environments. Upon logging into that server, the Thanx Media users should have the same rights as the "Administrator" user to perform implementation related tasks.  
  
The Thanx Media team has been given the "sys-endeca" user account with Administrator rights.  
  
**Application Installation and Directory Structure**Each Motorcycle Superstore application consists of all configuration XML and supporting documents, custom Control Scripts (defined below), shell scripts and, following data updates, MDEX index files and necessary directory structure to perform all implementation-specific tasks including data forge/indexing, MDEX Engine query responses, logging and report generation.   
  
The Motorcycle Superstore application will be deployed by Thanx to the following directory:

|  |
| --- |
| Directory |
| C:/Daniel/apps/pbs/ |

The Endeca Deployment Template is a collection of operational components that provides a starting point for development and application deployment. This template includes the complete directory structure required for deployment, including EAC scripts, configuration files, and batch files or shell scripts that wrap common script functionality.  
  
The current Endeca Deployment Template is version PCI 2.1.2 and will be used for the Motorcycle Superstore applications.

|  |
| --- |
| Deployment Script Location |
| C:/Daniel/ToolsAndFrameworks/2.1.2/deployment\_template/bin/ |

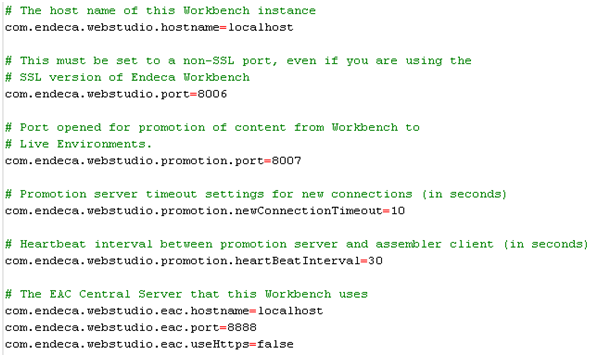
From the execution of the Endeca Deployment template's deploy.bat script, the full directory structure used by the application on the primary server is:   
  
**Workbench servers:**The default installation of the Endeca IAP software provides the necessary directory structure to power Workbench. No additional directory structure required.  
  
**Required Ports**The following TCP ports are required for ALL Endeca servers in ALL environments. The ports are non-standard and will not follow the Endeca default ports. Ports must be entered during installation of Platform Services, ToolsAndFrameworks (noted below) and during the application deployment.

|  |  |  |
| --- | --- | --- |
| Port | Process | Purpose |
| 8888 | EAC Http Service | Platform Services and EAC Port |
| 62911 | EAC Shutdown | Port for EAC Shutdown |
| 62912 | EAC JCD | EAC JCD Port |
| 8006 | Workbench | Access Workbench Application |
| 8007 | Workbench Promotion | Port used for promoting authoring to live |
| 62915 | CAS | CAS Server Port |
| 62916 | CAS Shutdown | Port for CAS Shutdown |
| 8446 | Workbench Redirect | Port for Workbench Redirect |
| 16010 | Logserver | Set port for logserver, logserver currently disabled, will need to be set below 32000 for operation. |
| 16000 | Live | Live MDEX Port |
| 16002 | Authoring | Authoring MDEX Port |
| 8084 | Workbench Shutdown | Port for Workbench Shutdown |

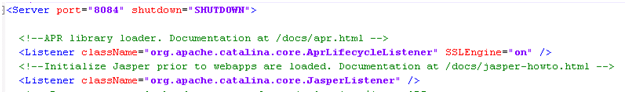
**Workbench Port Configuration**Port configuration for Platform Services and the application are set through the install and deployment template, respectively. Port setting for the Workbench must be manually configured in the ToolsAndFrameworks conf directory.

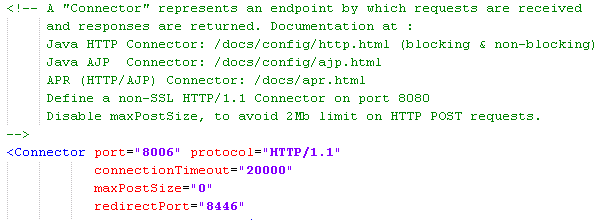
|  |
| --- |
| Directory |
| C:/Daniel/ToolsAndFrameworks/2.1.2/server/workspace/conf/ |

Within the above directory, modify the webstudio and webstudio promotion ports in the webstudio.properties file as seen below:

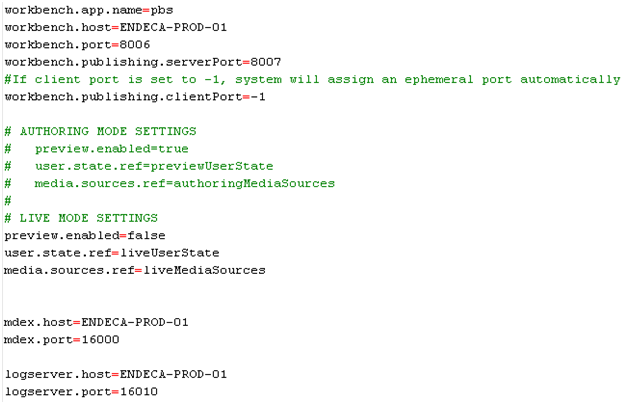


Additionally, within the server.xml file modify the SHUTDOWN port, and Connector ports. Connector port is the Webstudio port.





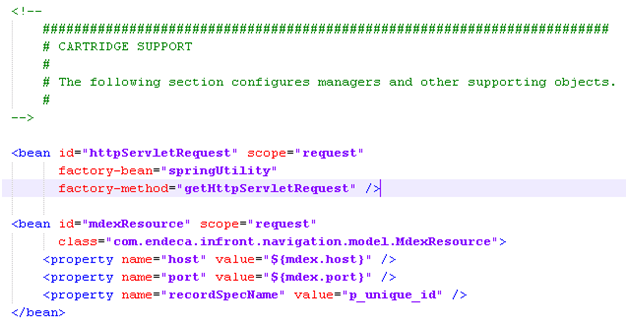
A restart of the workbench is required after the following changes are made.   
  
**Between-Environment Communication**Apart from the Motorcycle Superstore developed process for migrating configurations between environments and transfer of source data extracts, there should be no necessity for communication between servers in separate environments.   
  
The sole exception is if Motorcycle Superstore wishes that the thesaurus and merchandising rule modifications made by business users on the Production Workbench server to be propagated to the TST or SIT environments. If this is desired, an export of the site configuration can be created by executing the following script:  
  
C:/Daniel/apps/mag/control/export\_site.bat  
  
This will create an XML file with all Workbench settings. This file can then be moved to the new environment and imported using the following script:  
  
C:/Daniel/apps/mag/control/import\_site.bat $filename --force  
  
**Application Infrastructure Setup**This section details the configuration necessary for the web application. The production web application will be referred to as Motorcycle Superstore.  
  
At this point, the deployment template process should have been executed to create the application directory structure. The components and processes defined in this section are described further in section 17: "System Components and Operations."  
  
The AppConfig.xml, LiveDgraphCluster.xml , and environment.properties file was modified in order to properly provision the authoring and live web applications. Please refer to page 9 of the Deployment Template Usage Guide and the EAC Development Toolkit Usage Guide for additional details.  
  
In order to properly partition and manage multiple components, it is important to configure each component to use specific ports and directories.  
  
**Global variables**This section should require a few changes. Ensure that the eac\_hostname, eac\_port variables has the correct value for the EAC Central Server, and add Authoring and Live MDEX host and ports. Configuration for TST environment listed below  
  
Environment.properties:  
  
EAC\_PORT=8888  
EAC\_HOSTNAME=ENDECA-PROD-01  
  
#Dgraph settings  
LiveMDEXHost PORT=16000  
LiveMDEXHostA HOSTNAME= ENDECA-PROD-02  
LiveMDEXHostB HOSTNAME= 10.168.44.28  
LiveMDEXHostC HOSTNAME= 10.168.44.26  
  
AuthoringMDEXHost PORT=16002  
AuthoringMDEXHost HOSTNAME= 10.168.44.26  
  
AppConfig.xml:  
  
**Servers/Hosts**In order to manage the application across different environments, we will define host references to represent each server's role.  
  
"hostName" should represent a host or reachable IP address of the EAC agent.  
  
ITLHost will contain the Forge and Dgidx components for the nightly process.  
  
DgraphA1[LiveMDEXHostA] will contain the primary Dgraph component.  
  
DgraphB1[LiveMDEXHostB] will contain the secondary Dgraph component.  
  
DgraphC1[LiveMDEXHostC] will contain the last Dgraph component.  
  
AuthoringDgraph will contain the authoring Dgraph component  
  
webstudio will contain the Endeca Workbench component.  
  
**Forge and Dgidx component**The Forge and Dgidx components for the Staging and Production applications were not altered. Configuration for these components are found in the DataIngest.xml  
  
**Authroing Dgraph cluster component**Define a Dgraph cluster component that will be used in the authoring application. Note that the id is unique in the file, and the reference is to the staging Dgraph component.  
  
AuthoringDgraphCluster.xml:  
  
**Production Dgraph cluster component**Define a Dgraph cluster component that will be used in the production application. Note that the id is unique in the file, and the references are to the production Dgraph components.  
  
AuthoringDgraphCluster.xml:  
  
**Authoring Dgraph component**Define the Dgraph component that will be used in staging. Note that the ids are unique in the file, and the host-id value is to the staging MDEXHost reference.  
  
**Production Dgraph components**Define a Dgraph cluster component that will be used in the production application. Note that the id is unique in the file, and the references are to the production Dgraph components.  
  
LiveDgraphCluster.xml:   
  
**Authoring Dgraph component**Define the Dgraph component that will be used in staging. Note that the ids are unique in the file, and the host-id value is to the staging MDEXHost reference.  
  
**Production Dgraph components**Define the Dgraph components that will be used in production. Note that the ids are unique in the file, and the host-id values are to the production MDEXHost, MDEXHost2 references. In this setup it is acceptable to use the same port values since they will be on different physical machines.  
  
**Baseline Update script for Production**Define the Baseline Update script that will be used for the nightly process. This script is found in the DataIngest.xml.   
  
**ADD CUSTOM SCRIPTS HERE**CUSTOM SCRIPTS  
  
**Application Provisioning**In order to provision the application you may use the ./control/initialize\_services.bat script. The AppConfig.xml file must be available in the ./config/script directory.  
  
**Assembler API Configuration**First, edit the assembler.properties file located in at:  
C:/Daniel/ToolsAndFrameworks/2.1.2/reference/pbs-service/WEB-INF/assembler.properties. All variables will need to be updated to match the configuration as noted in the above sections. For mdex host and port, please reference the load balancer port and host if configured.



Next, to enable Web-service calls, if needed, we will update the assembler-context.xml file located in the same directory. First comment out the rollupKey property located in the navigationStateBuilder block:



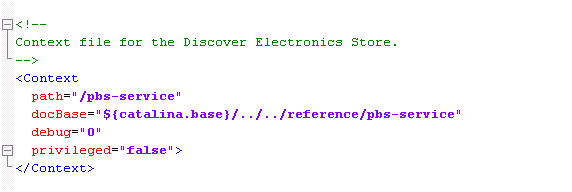
Then, note the record spec (common.id) in the mdexResource block:



Next, if needed, you can limit the properties returned in the ResultsList cartridge by adding or modifying the property names here:



When all of the values are commented out, all properties will return.  
  
Finally, add the service context file by creating a file called docen.xml in the C:/Daniel/ToolsAndFrameworks/2.1.2/erver/workspace/conf/Standalone/localhost directory. The file will contain the path from the url and the service path location as seen below:



Once setup, restart the Tools service, and verify you are receiving back xml/json from the following URLS:  
http://10.168.44.26:8006/pbs-service/json/services/guidedsearch  
http://10.168.44.26:8006/pbs-service/xml/services/guidedsearch

# System Components and Operations

This section provides the details on the vital components and processes of the implementation.   
  
**MDEX Engine Servers**The heart of the Endeca implementation is the MDEX Engine. The MDEX Engine is the online process that responds to requests from Motorcycle Superstore (via the Endeca API) by delivering guided navigation objects, keyword search results, refinement options, merchandising results, and other supporting information. This response is interpreted by the Motorcycle Superstore site, again through the Endeca API, for display of search results, refinement navigation elements and merchandising results.  
  
The MDEX Engine can be expected to provide 25 operations per second. Motorcycle Superstore has two instances of the MDEX Engine spread out across two servers dedicated to serving search and navigation requests from the Motorcycle Superstore servers. The use of an HTTP load balancer to manage traffic between the web application servers and the MDEX Engine farm is recommended.  
  
Note: Since Endeca is installed on a shared server with other Endeca resources, performance testing should also take into account other application server load.   
  
**Load Balancing MDEX Requests**An Endeca-based application relies upon the availability of an MDEX Engine to service user requests. If that MDEX Engine should be unavailable, then the Endeca portion of the application will be unable to respond to queries. The MDEX Engine might be unavailable or appear to be unavailable for any number of reasons, including hardware failure, an in-process update of the MDEX Engine's indices, or, in extreme cases, very high load on a given MDEX Engine. In addition, this deployment entails two instances of the MDEX Engine spread out over two servers. For these reasons, an HTTP load balancer is strongly recommended to manage traffic to the multiple Navigation Engines and to manage instances when an MDEX Engine is unavailable to serve requests.  
  
The MDEX Engine functions very similarly to a web server in terms of network traffic. It simply accepts HTTP requests on a specified port, and returns results to the caller. This behavior allows for standard web load balancing techniques to be applied.  
  
Ports should be dedicated to communication between the load balancer and each MDEX Engine instance on each MDEX server.  
  
Motorcycle Superstore has advised that there will be two instances for each product and content engine.   
  
**Health Checks**Load balancers are also typically able to perform "health" status checks on the various MDEX Engine processes, by either making an HTTP request to the MDEX Engine's TCP port or by opening a connection to that port to ensure that the MDEX Engine is listening and alive. The MDEX Engine's admin request is often used for this query (i.e., http://dgraphhost:16000/admin?op=ping).  
  
Upon failure of a health status check, the load balancer should be able to mark the port (process) as unavailable and remove it from the set of ports that it is directing load to (and preferably be able to alert on error - via a mechanism such as an SNMP trap). Most load balancers will also support the ability to periodically attempt to recheck a port marked as unavailable to see if it is once again available.  
  
**Backup Load Balancer**Endeca typically recommends implementing two hardware switches configured redundantly (either by network design or some other method). This will ensure that the load balancer is not a single point of failure as well.  
  
**Endeca Application Controller**The Endeca Application Controller (EAC) is the mechanism for performing data forging and indexing, index file distribution, data archiving, report generation, and other Endeca-related tasks in the production environment. The EAC is comprised of two components: the Central Server and Agents. The EAC provides your Endeca implementation with a reliable process execution and lightweight job management.  
  
**Control Scripts**Control scripts are used to perform various tasks with an Endeca implementation.  
  
These scripts are included with the Endeca Deployment Template after the bin/deploy.bat is run.  
  
**Script Execution**Scheduled jobs are created on the primary Data Foundry server to initiate particular tasks according to the schedule below. All the necessary tasks are captured in one script:

|  |  |  |
| --- | --- | --- |
| Script Name | Function | Schedule |
| Motorcycle Superstore Baseline Update | Performs a full data update. | 10:00PM on Saturday |
| Motorcycle Superstore Partial Updates | Runs the partial update process. | 9:30AM,11:30AM,1:30PM,3:30PM Daily |
| Motorcycle Superstore Thesaurus Update | Updates the thesaurus.xml file | 11:00AM on Saturday |

All scripts (including those referenced within the main script) reside in the following "control" directories on the Data Foundry server:

|  |
| --- |
| Directory |
| C:/Daniel/apps/mag/control |

**Data Extracts**It is Motorcycle Superstore's responsibility to develop the process for producing the data extract used to populate the CAS record stores.   
  
Record store data will reside in the following directories:

|  |
| --- |
| Directory |
| /lcl/$ENV/apps/edoc/endeca/CAS/workspace/state/DOCen\_en\_data |
| /lcl/$ENV/apps/edoc/endeca/CAS/workspace/state/DOCen\_en\_dimvals |
| /lcl/$ENV/apps/edoc/endeca/CAS/workspace/state/DOCen\_en\_prules |
| /lcl/$ENV/apps/edoc/endeca/CAS/workspace/state/DOCen\_en\_schema |

**Data Updates**The Motorcycle Superstore implementation will entail one type of update, the Baseline Update. It is important to note all Data Update processes are fully automated and are initiated by the execution of the script described above.   
  
**Baseline Update**A Baseline Update, or full data update, transforms all data in the provided extract files into Endeca index files for use by the MDEX Engine.  
  
Index files from the previously run are not incorporated into the current update. Therefore, the data extract files MUST include all Motorcycle Superstore product and SKU data that is to be available for search queries.  
  
The raw source data is transformed through an offline process called Forge. The Forge process readies the source data for use by the Index process and produces a set of binary and supporting XML files.  
  
The Index process then takes over and transforms the Forge output into binary index files for use by each instance of the MDEX Engine. The integrity of the index files are tested by starting an instance of the MDEX Engine on the Data Foundry server. Upon successful startup, the index files are then migrated to each instance of the MDEX Engine on all MDEX servers, as well as the MDEX Engine that serves Workbench requests on the Workbench server. The migration process is described below.  
  
**Index File Migration (Baseline Update)**The MDEX Engine derives its data from a collection of binary index files and supporting XML and other files. These files are produced during the Data Update processes described above and copied from the Data Foundry server to each instance of the MDEX Engine.  
  
It is assumed that firewalls or other restrictions will not prevent MDEX and Data Foundry servers in production from communicating with each other, nor prevent file transfer between production Endeca servers.  
  
The following describes, step-by-step, the Index File Migration process for a MDEX Update:  
 **Workbench and Configuration Migration**The Endeca Workbench is a business tool allowing Motorcycle Superstore personnel to make updates to merchandising business rules, thesaurus entries and other limited aspects of the MDEX configuration without the necessity for technical or development skills.  
  
The Workbench operates within its own environment with its own instance of the MDEX Engine. To ensure that business users are performing tasks on the current index and that their modifications to MDEX configurations are incorporated into production indices, the Baseline Update process must touch the Workbench environment.  
  
**Incorporating Workbench Configuration Changes into Full Updates**After the Baseline Forge process, created dimensions and values are uploaded to the Workbench repository by performing the following functions:  
 **Logging & Report Generation**The Logging server performs the task of receiving HTTP requests from the web application via the Endeca API and producing log entries for use by the Report Generator process to produce daily and weekly Endeca usage reports. These reports are viewed via the Endeca Workbench.  
  
Log files are rolled on a nightly basis to maintain a reasonable log file size. This process will occur at the completion of each nightly Baseline Update.  
  
**Development Environment**Application Development will occur in the development environment and deployed directly to that environment.  
  
Development occurs with the Developer Studio tool which should be installed on each developer's computer. To facilitate ease of development, we recommended using Motorcycle Superstore's current version control software (e.g. Visual SourceSafe, Subversion).  
  
**Migration from Authoring Dgraph to Production Dgraphs**Within each environment, the Authoring environment serves as the staging area for changes to the Workbench. When workbench changes (such as redirects, thesaurus, and rule manager) are ready for promotion to the live environment, the following script can be executed from command line:

1. The Index files are transferred from the Data Foundry server into temporary directories created for each instance of the MDEX Engine on all MDEX Engine servers, as well as a backup directory on each server. The backup directory is created so that during the subsequent Baseline Update, this set of Index files is available should difficulties during the migration process necessitate a rollback to the last valid Index.
2. One by one, a command is issued to the MDEX Engine process to shutdown. It is important to note that all requests currently in queue for that Engine after the command has been issued are served before the actual shutdown of the process occurs.
3. The existing MDEX Engine data directory is deleted and the temporary directory data directory is renamed to serve as the current data directory.
4. The MDEX request logs are then archived.
5. The MDEX Engine is then started and is ready to serve requests.
6. Steps 2-5 are repeated for each MDEX Engine. The end result is minimal downtime for the MDEX Engine being updated while keeping a significant number of MDEX Engines online to continue to serve requests.
7. After baseline is completed, cleans Workbench input directories.
8. Copies the XML configuration files from the Workbench server to the DataFoundry server.
9. Runs updateWsConfig to integrate changes into the repository.

|  |
| --- |
| Directory |
| C:/Daniel/apps/mag/control/promote\_content.bat |

Or from within the Workbench admin console:

