pitney bowes

Spectrum Technology Platform

Version 12.0

Information Extraction Guide

Table of Contents

1 - Introduction

In this section

Information Extraction Module

The Information Extraction Module provides advanced text processing capabilities and information extraction from natural language input text.

Features Provided

Text Categorization	Allows you to assign custom categories to unstructured text. This is possible after you have trained a *text categorization model* using the Administration Utility. This feature can be used to index patient health-care reports, classify documents by domains and subdomains, and categorize email into SPAM and non-SPAM, among other applications.
Entity Extraction	Allows you to train models to extract entities from unstructured data. The Module ships with some *preexisting entities*. If required, define custom entities using the *CustomEntity* model type. After you create and train domain-specific models, you can extract entities based on the model trained by you.
Relationship Extraction	Allows you to identify the relationship type binding a pair of entities in any natural language input text.

Languages Supported

For all the stages of Information Extraction Module, the current release supports information extraction capabilities for *English* language input text only.

Note: In case of the **Entity Extractor** stage, in addition to English, support for these languages is in the *beta* phase:

es	Spanish (Mexico)
fr	French
de	German
pt	Portuguese (Brazil)

Important: These *beta* languages are available only in case of *Custom Entity*, and not for preexisting entities.

Model Security

Security permissions must be assigned in **Management Console** to perform various functions using Information Extraction:

- View permissions are necessary to categorize or list the model.
- Modify permissions are necessary to retrain or import the model (if model already exists).
- Create permissions are necessary to import or train the model.
- Delete permissions are necessary to delete the model.

2 - Text Categorization

In this section

Introduction to Text Categorization

Text categorization, also known as text classification, is the process of assigning custom categories to the unstructured content or plain text, such as email, news articles, and comments on the basis of how much of its content matches the category. Categorization can be done based on subject, author, date, or virtually any classification system you devise.

You can create your own categorizer by training a categorizer model with your data and categories. The trainer analyzes the data and stores the information it gains in the training process. It then analyzes the content and determines the category to which the content belongs.

The text categorization feature uses statistical text categorization process. It applies machine learning methods to learn automatic classification rules that are based on human-labeled training documents.

Because you are able to apply the categorization of your choice, you first need to "train" your model to "learn" the categories. After this, you can use that model in the **Text Categorizer** stage to categorize your unstructured data.

Spectrum™ Technology Platform uses administration utility commands to manage text categorization models. For a description of these commands, see **Administration Utility Commands** on page 30.

Preparing Data

The first step in using text categorization is preparing your input file and your test file. For this, you need to structure the data as tab separated values in both the files. The files need to have details in this format:

- UFT-8 encoding
- Tab-separated data in two columns, where the first column contains the category name (for example: "Patient" or "Provider") and the second column has the data for each category (as displayed in the example below)

Your data should look as:

```
Patient     John Smith dob04181963 224 Main St. Atl GA 30311
Provider    Mark Johnson M.D. NPI5489512047 412 Washington Atl GA 30301
```

Configuring Options

This involves creation of a `Training Options` file that contains information about your model and the options to be applied for training the model. This file must be in XML format with UFT-8 encoding and must include these header and the required training features:

Header in the Training Options File

The header mentions details of the model, its type, and the path of the test and input files.

- `modelName`: Name of the model
- `modelType`: The type of model (which is `TC`, meaning text categorization in this case)
- `modelDescription`: Description of the model
- `inputFilePath`: Location of the input file used for training the model
- `testFilePath`: Location of the test file

> **Note:**
>
> The test file measures the effectiveness of a model. It determines the behavior of the custom model with various training parameters. As a best practice you should use different input and test files in training or evaluating your custom models.

`algorithm`: The machine learning algorithm used for training the model (default is `MaxEnt`)

Training Features

These are the training features you can use to create a new category.

> **Note:** If you use multiple features, those can be placed in any order within the file.

- **Linguistic feature**: To specify the language properties
  - `Stemming`: Reduces words to their stem, or root. For example, "insurer", "insured", and "insures" can all be reduced to the root "insure".

```
<trainingFeature>
  <featureName>Stemming</featureName>
</trainingFeature>
```

- **Keyword features**: To define the list of keywords
  - `IgnoreWords`: Also known as stop words, this feature filters out common words that have no effect on categorization, such as "the", "and", and "but". These words should be separated only

by a comma, not by spaces. You can also use the `Append` key with this feature, which when set to "True", will be added to the existing list of stopwords.

```
<trainingFeature>
  <featureName>IgnoreWords</featureName>
  <featureParams>
   <entry>
    <key>WordList</key>
    <value>
     and,the,for,with,still,tri,rep,cust,keep,get,req,call
    </value>
   </entry>
   <entry>
    <key>Append</key>
    <value>True</value>
   </entry>
  </featureParams>
 </trainingFeature>
```

- `CategoryKeywords`: Identifies a category for a list of keywords belonging to multiple custom lists. For example, Weekdays in `CategoryKeywords` list contains `Keywords` as Monday, Tuesday, Wednesday, Thursday, and Friday.

  This feature can optionally specify if the match should be case sensitive. When used, the default is `true`.

```
<trainingFeature>
 <featureName>CategoryKeywords</featureName>
 <featureParams>
  <entry>
   <key>Weekdays</key>
                    <!-- List of weekdays -->
   <value>Monday,Tuesday,Wednesday,Thursday,Friday</value>
  </entry>
  <entry>
   <key>WeekendDays</key>
                    <!-- List of weekend days -->
   <value>Saturday,Sunday</value>
  </entry>
  <entry>
   <key>CaseSensitive</key>
                    <value>True</value>
  </entry>
 </featureParams>
</trainingFeature>
```

- `KeyWords`: Searches for words that you have specified as belonging to a custom list, such as *DaysOfWeek* or *Month*. Also optionally specifies whether the match should be case sensitive, which, when used, has "`true`" as default.

```
<trainingFeature>
 <featureName>KeyWords</featureName>
 <featureParams>
  <entry>
   <key>KeyWordList</key>
   <value>Monday,Tuesday</value>
  </entry>
  <entry>
   <key>CaseSensitive</key>
   <value>False</value>
  </entry>
 </featureParams>
</trainingFeature>
```

- **Lexical feature**: To specify the lexeme properties
  - `NGram`: Searches for a portion of a longer string, with "n" representing the number of words to look for. For example, if you are looking for the phrase "to be or "not to be", you might search for a unigram of "to" or "be", or a bigram of "to be" or "or not", or a trigram of "to be or" or "not to be".

```
<trainingFeature>
  <featureName>NGram</featureName>
  <featureParams>
   <entry>
    <key>Count</key>
    <value>3</value>
   </entry>
  </featureParams>
</trainingFeature>
```

A sample training options file:

```
<trainingOptions>
 <modelName>modelone</modelName>
 <modelType>TC</modelType>
 <modelDescription>modelOne</modelDescription>

<inputFilePath>C:/SpectrumIE/textclassification/train_Input.csv</inputFilePath>


<testFilePath>C:/SpectrumIE/textclassification/train_Test.txt</testFilePath>

    <algorithm>SVM</algorithm>

 <trainingFeatures>
```

```
<!-- Keyword features -->
<trainingFeature>
 <featureName>IgnoreWords</featureName>
 <featureParams>
  <entry>
   <key>WordList</key>
   <value>
    and,the,for,with,still,tri,rep,cust,keep,get,req,call
   </value>
  </entry>
  <entry>
   <key>Append</key>
   <value>True</value>
  </entry>
 </featureParams>
</trainingFeature>

<trainingFeature>
 <featureName>CategoryKeywords</featureName>
 <featureParams>
  <entry>
   <key>Category1/key>
   <value>CategoryKeyword1,CategoryKeyword2</value>
  </entry>
  <entry>
   <key>Category2/key>
   <value>CategoryKeyword3,CategoryKeyword4</value>
  </entry>
            </featureParams>
</trainingFeature>

<trainingFeature>
           <featureName>KeyWords</featureName>
      <featureParams>
  <entry>
   <key>KeyWordList</key>
   <value>
    jam,misfeed,install,help,mechanical,failure,jam,pc,connection
   </value>
  </entry>
 </featureParams>
</trainingFeature>

<!-- Linguistic feature -->
<trainingFeature>
 <featureName>Stemming</featureName>
</trainingFeature>

<!-- Lexical feature -->
<trainingFeature>
 <featureName>NGram</featureName>
 <featureParams>
```

```
   <entry>
    <key>Count</key>
    <value>3</value>
   </entry>
  </featureParams>
</trainingFeature>

</trainingFeatures>
</trainingOptions>
```

Training the Model

After creating an options file, you need to train your model to discover potentially predictive relationships. You can do this by applying the machine learning methods. Spectrum™ Technology Platform uses **iemodel train** on page 36 CLI command to train a model. After training the model you can use in categorization.

Evaluating the Model

You may want to test your model after training it to ensure that the training options file is correct and categories are being assigned as expected.

You can test the model using the **iemodel trainAndevaluate model** on page 37 CLI command.

Categorizing Text

1. Create a data flow that includes a source stage like **Read from File** or **Read from XML**, the **Text Categorizer** stage, and a sink stage like **Write to File** or **Write to XML**.
2. In the source stage, point to your input file.
3. In the **Text Categorizer** stage, select the model in the **Categorizer name** field. This is the model you trained in the text categorization phase. For information about training a model, see **Training the Model** on page 12.
4. In the **Category count** field, select the number of matching levels of category that should be included in the output. For example, the closest match or closest plus the second close match.

   Note: The maximum value corresponds to the number of different categories specified while training the model.

5. Click **OK**.
6. Run the job.

3 - Entity Extraction

In this section

Introduction

Entity extraction is the process of identifying and retrieving entities from an unstructured data. You can use the preexisting entities shipped along with the **Entity Extractor** stage, or you can train a model to extract custom entities.

Preexisting Entities

Preexisting entities are those that come with the **Information Extraction** Module.

To find a list of the preexisting entities, open the **Entity Extractor** stage, select the **Override system default options with the following values** check box, and click **Quick Add**. The list of the entities gets displayed in the **Select entities** section.

- *Person*
- *Address*
- *ProperNouns*
- *ISBN*
- *CreditCard*
- *ZipCode*
- *WebAddress*
- *Mention*
- *HashTag*
- *SSN*
- *Phone*
- *Email*
- *Date*
- *Location*
- *Organization*

Follow the remaining steps in this section to extract these kinds of entities from your data.

Extracting Preexisting Entities

1. Create a dataflow that includes a **Read from Documents** source stage, an **Entity Extractor** stage, and a sink stage like **Write to File** or **Write to XML**.

2. In the source stage, point to your input file.
3. In the **Entity Extractor** stage, select the entities based on the data that you want to extract from the input file. For example, if you want to select names of all persons and addresses in the file, select the *Address* and *Person* entities.

   Note: *Address* and *Person* are the default entities. To extract data based on any other entity, select the **Override system default options with the following values** check box, and click **Quick Add**. The list of the entities gets displayed in the **Select entities** section.

4. To get the frequency in the input file of the data related to the specified entities, select the **Output entity count** check box.
5. Click **OK**.
6. Run the job.

Custom Entities

As with preexisting entities, you can also train models to retrieve custom entities. These entities can belong to any domain and can be of any type. For example, you can use medical text to extract a list of diagnoses or pharmaceuticals. The process of extracting custom entities includes:

1. Preparing data: Preparing the input file and test file
2. Configuring the options: Creating training options file that contains information about the model and the options to be applied while training the model
3. Training the model
4. Extracting entities

When you successfully perform all these steps, the new entity type gets added to the list in the **Entity Extractor** stage, and you can use it to extract details from an unstructured file.

Preparing Data for Custom Entities

The first step in creating custom entities is preparing your input file and your test file. The custom entities feature requires that the entities in those files be surrounded by magicWord you specify in your training options file (which is discussed in the next topic).

Let's say you are extracting diagnoses from unstructured data in your input file and you have designated the magicWord *DIAGNOSIS* in your training options file. Every time the name of a disease or condition appears in the text, the word would be enclosed with that magicWord, as follows:

```
The term diagnostic criteria designates the specific combination of
signs, symptoms, and test results that the clinician uses to attempt
```

```
to determine the correct diagnosis. Some examples of diagnostic
criteria, also known as clinical case definitions, are: Amsterdam
criteria for DIAGNOSIShereditary nonpolyposis colorectal cancerDIAGNOSIS
McDonald criteria for DIAGNOSISmultiple sclerosisDIAGNOSIS ACR criteria

for DIAGNOSISsystemic lupus erythematosusDIAGNOSIS Centor criteria for

DIAGNOSISstrep throatDIAGNOSIS.
```

For information about identifying magicWord, see the next topic.

Configuring Options for Custom Entities

This involves creation of a `Training Options` file that contains information about your model and the options to be applied for training the model. This file must be in XML format with UFT-8 encoding and must include these header and the required training features:

Header in the Training Options File

The header mentions details of the model, path of the test and input files, and the keyword for annotating the custom entities.

- `modelName`: Name of the custom model
- `modelType`: The type of the custom model (which is *CustomEntity*).
- `modelDescription`: Description of the custom model
- `inputFilePath`: Path of the tagged file used for training the model (input file)
- `testFilePath`: Path of the file used for testing the model
- `magicWord`: Keyword used to annotate the custom entities
- `language`: The language used in the text.

Note: English is supported. Dutch, French, German, and Spanish are in the beta phase.

Training Features

You can use these training features to create the custom entities.

- **Linguistic features**: To specify the language properties
  - `POSTagger`: Tagging to identify parts of speech, such as nouns, pronouns, adjectives, and verb.

```
<trainingFeature>
   <featureName>POSTagger</featureName>
</trainingFeature>
```

- **Orthographic features**: To specify the structural properties

- `CaseIdentifier`: Identifies whether the custom entities are all capital letters, lower-cased, or a mix of both.

```
<trainingFeature>
 <featureName>CaseIdentifier</featureName>
</trainingFeature>
```

- `NumericIdentifier`: Identifies whether the custom entities are numeric or alphanumeric.

```
<trainingFeature>
 <featureName>NumericIdentifier</featureName>
</trainingFeature>
```

- `1st2ndIdentifier`: Identifies whether the custom entities are ordinals, such as 1st, 2nd, and 3rd.

```
<trainingFeature>
 <featureName>1st2ndIdentifier</featureName>
</trainingFeature>
```

- `PatternMatcher`: Matches words against one or more patterns using regular expressions. When multiple expressions are provided, includes join condition `AND` for all expressions or `OR` (default) for any expression.

```
<trainingFeature>
 <featureName>PatternMatcher</featureName>
  <featureParams>
   <entry>
    <key>RegEx1</key>
    <value>b[aeiou]t</value>
   </entry>
   <entry>
    <key>RegEx2</key>
    <value>b[xyz]t</value>
   </entry>
   <entry>
    <key>JoinCondition</key>
    <value>AND</value>
   </entry>
  </featureParams>
</trainingFeature>
```

- **Keyword features**: To define the list of keywords
  - `CategoryKeywords`: Identifies a category for a list of keywords belonging to multiple custom lists. For example, Weekdays in `CategoryKeywords` list contains `Keywords` as Monday, Tuesday, Wednesday, Thursday, and Friday.

This feature can optionally specify if the match should be case sensitive. When used, the default is `true`.

```
<trainingFeature>
 <featureName>CategoryKeywords</featureName>
 <featureParams>
  <entry>
   <key>Weekdays</key>
                    <!-- List of weekdays -->
   <value>Monday,Tuesday,Wednesday,Thursday,Friday</value>
  </entry>
  <entry>
   <key>WeekendDays</key>
                    <!-- List of weekend days -->
   <value>Saturday,Sunday</value>
  </entry>
  <entry>
   <key>CaseSensitive</key>
                    <value>True</value>
  </entry>
 </featureParams>
</trainingFeature>
```

- `KeyWords`: Searches for words that you have specified as belonging to a custom list, such as *DaysOfWeek* or *Month*. Also optionally specifies whether the match should be case sensitive, which, when used, has "`true`" as default.

```
<trainingFeature>
 <featureName>KeyWords</featureName>
 <featureParams>
  <entry>
   <key>KeyWordList</key>
   <value>Monday,Tuesday</value>
  </entry>
  <entry>
   <key>CaseSensitive</key>
   <value>False</value>
  </entry>
 </featureParams>
</trainingFeature>
```

- `Substring`: Extracts part of a string as specified in the parameters. Can also be used for prefix and suffix extractions.
  - `StartLocation`: Left or right. Position where substring should be extracted. Default is `Left`.
  - `StartPosition`: Start position for the substring. The default is `0`.
  - `EndPosition`: End position for the substring. Default is `3`.

- `MinLength`: Minimum length of word to which this feature should apply. Default is `3`.

```
<trainingFeature>
 <featureName>Substring</featureName>
  <featureParams>
   <entry>
    <key>StartLocation</key>
   </entry>
   <entry>
    <key>StartPosition</key>
    <value>1</value>
   </entry>
   <entry>
    <key>EndPosition</key>
    <value>4</value>
   </entry>
   <entry>
    <key>MinLength</key>
  </featureParams>
</trainingFeature>
```

- **Lexical Features**: To specify the lexeme properties
  - `FeatureWindow`: Specifies the window for feature generation

```
<trainingFeature>
 <featureName>FeatureWindow</featureName>
 <!-- Number of preceding tokens used to create the feature set.
Default is 3 -->
  <entry>
   <key>Before</key>
   <value>1</value>
  </entry>
 <!-- Number of succeeding tokens used to create the feature set.
Default is 3 -->
  <entry>
   <key>After</key>
   <value>2</value>
  </entry>
</trainingFeature>
```

A complete sample training options file for custom entities is shown below:

```
<trainingOptions>
 <modelName>CustomModel</modelName>
 <modelType>CustomEntity</modelType>
 <modelDescription>CustomDiagnosesModel</modelDescription>

<inputFilePath>C:/SpectrumIE/custom_model/Custom_Input.csv</inputFilePath>
```

```
<testFilePath>C:/SpectrumIE/custom_model/Custom_Test.txt</testFilePath>

      <magicWord>DIAGNOSIS</magicWord>
      <language>English</language>

     <trainingFeatures>

<!-- Lexical features-->
<trainingFeature>
 <featureName>FeatureWindow</featureName>
 <featureParams>
  <entry>
   <key>Before</key>
   <value>1</value>
  </entry>
  <entry>
   <key>After</key>
   <value>2</value>
  </entry>
 </featureParams>
</trainingFeature>

<!-- Orthographic features-->
<trainingFeature>
 <featureName>CaseIdentifier</featureName>
     </trainingFeature>

<trainingFeature>
 <featureName>NumericIdentifier</featureName>
</trainingFeature>
</trainingFeatures>
</trainingOptions>
```

Training the Custom Entities Model

After creating an options file, you need to train your model to identify custom entities. Spectrum™ Technology Platform does this with the **iemodel train** on page 36 CLI command. A trained model is used to retrieve custom entities.

Evaluating the Custom Entities Model

You may want to test your model after training it to ensure that the training options file is correct and the entities are being extracted as expected. To test your model, use the **iemodel trainAndevaluate model** on page 37 CLI command.

Extracting Custom Entities

The trained custom entity, which is now available in the entity list of the **Entity Extractor** stage can be used to extract relevant information from your unstructured data.

For the steps to extract preexisting entities, see **Extracting Preexisting Entities** on page 15.

4 - Relationship Extraction

In this section

Introduction

Relationship Extraction allows you to identify the relationship types between a pair of entities identified in the source content. It analyzes the natural language content in the source document, and identifies the relationship types existing between all the entity pairs identified.

Entity Types Supported

Currently, the entity types supported for relationship extraction are:

- *Person*
- *Organization*
- *Location*

Relationship Types

RelationshipType	Entity1 Type	Entity2 Type	Relationships Covered
AffiliatedWith	*Person*	*Organization*	Indicates any professional or academic relationship between the *Person* and the *Organization* entities. The relationship can be any of these or other similar relationships: • *Person* is studying in or studied in *Organization* • *Person* is working with or worked with *Organization* • *Person* has been offered a job at *Organization* **Note:** This is an indicative list of the relationships this type covers. For example, `James has studied from the University of Toronto and works at ABC Corp.` Here, two relationships can be parsed: Entity1 = `James`, RelationshipType = `AffiliatedWith`, Entity2 = `University of Toronto` Entity1 = `James`, RelationshipType = `AffiliatedWith`, Entity2 = `ABC Corp`

RelationshipType	Entity1 Type	Entity2 Type	Relationships Covered
LivesIn	*Person*	*Location*	Indicates a relationship between the *Person* and the *Location* entities. The relationship can be any of these: • *Person* stays in or stayed in *Location* • *Person* shifted to *Location* • *Person* was born in *Location* • *Person* died at *Location* **Note:** This is an indicative list of the relationships this type covers. For example, `John Jamison, a National Weather Service meteorologist in Galveston, reported that a massive hurricane was about to hit the East Coast the next day.` Entity1 = `John Jamison`, RelationshipType = *LivesIn*, Entity2 = `Galveston`
OrgBasedIn	*Organization*	*Location*	Indicates that the *Organization* has at least one of its offices in the *Location*. The *Location* can be a branch office, development office, headquarters, and the like. For example, `HSBC Holdings Plc. is headquartered in London, United Kingdom.` Here, two relationships can be parsed: Entity1 = `HSBC Holdings Plc.`, RelationshipType = *OrgBasedIn*, Entity2 = `London` Entity1 = `HSBC Holdings Plc.`, RelationshipType = *OrgBasedIn*, Entity2 = `United States of America`

RelationshipType	Entity1 Type	Entity2 Type	Relationships Covered
LocatedIn	*Location*	*Location*	Indicates the relationship between two different locations, where one of the entities is geographically contained within the other entity. **Example 1** `Canberra is the capital of Australia.` Here, Entity1 = `Canberra`, RelationshipType = *LocatedIn*, Entity2 = `Australia` **Example 2** `India has as its capital New Delhi.` Here, Entity1 = `India`, RelationshipType = *LocatedIn*, Entity2 = `New Delhi`
Negative	*Person* *Organization* *Location*	*Organization* *Location*	Indicates that none of the above relationship types could be parsed between the two corresponding entities. For example, `New Delhi and New York are good places to live in.` On parsing this input text, none of the supported relationship types are parsed between any pair of identified entities. Hence it can be broken down into *Negative* relationship types between the identified entities: Entity1 = `New Delhi`, RelationshipType = *Negative*, Entity2 = `New York`

Note: You can connect a **Splitter** stage at the output of the **Relationship Extractor** stage to extract the identified relationship types and the corresponding pair of entities joined by the relationship. The splitter stage converts the hierarchal output of this stage to a flat output.

Example

In case of a complex input text, multiple possible relationship type combinations might be parsed for the same sentence.

For example,

```
James McCarthy has settled in New York, United States as director of ABC Technologies.
```

When the **Relationship Extractor** stage parses this input text using the relationship types selected in the stage options, the relationships found are:

Relationship 1	Entity1 = `James McCarthy`, Entity1 Type = *Person*, RelationshipType = *LivesIn*, Entity2 = `New York`, Entity2 Type = *Location*
Relationship 2	Entity1 = `James McCarthy`, Entity1 Type = *Person*, RelationshipType = *AffiliatedWith*, Entity2 = `ABC Technologies`, Entity2 Type = *Organization*
Relationship 3	Entity1 = `ABC Technologies`, Entity1 Type = *Organization*, RelationshipType = *OrgBasedIn*, Entity2 = `United States`, Entity2 Type = *Location*
Relationship 4	Entity1 = `ABC Technologies`, Entity1 Type = *Organization*, RelationshipType = *OrgBasedIn*, Entity2 = `New York`, Entity2 Type = *Location*
Relationship 5	Entity1 = `James McCarthy`, Entity1 Type = *Person*, RelationshipType = *LivesIn*, Entity2 = `United States`, Entity2 Type = *Location*
Relationship 6	Entity1 = `New York`, Entity1 Type = *Location*, RelationshipType = *LocatedIn*, Entity2 = `United States`, Entity2 Type = *Location*

5 - Administration Utility Commands

In this section

Administration Utility Commands

The Administration Utility commands are used to manage, evaluate, and train Information Extraction Module models.

iemodel delete

The `iemodel delete` command returns a list of all Information Extraction Module models.

Usage

`iemodel delete --n` *modelName*

Required	Argument	Description
Yes	`--n` *modelName*	Specifies the name of the model you want to delete. Directory paths you specify here are relative to the location where you are running the Administration Utility.

Example

This example deletes the model called "MyModel".

```
iemodel delete --n MyModel
```

iemodel evaluate model

The `iemodel evaluate` command evaluates an Information Extraction Module model that has previously been trained.

Usage

`iemodel evaluate model --n` *modelName* `--t` *testFileName* `--o` *outputFileName* `--c` *categoryCount* `--d` *trueOrfalse*

Required	Argument	Description
Yes	`--n` *modelName*	Specifies the name and location of the model you want to evaluate. Directory paths you specify here are relative to the location where you are running the Administration Utility.
Yes	`--t` *testFileName*	Specifies the name and location of the test file used to evaluate the model.
No	`--o` *outputFileName*	Specifies the name and location of the output file that will store the evaluation results.
No	`--c` *categoryCount*	Specifies the number of categories in the model; must be a numeric value. **Note:** It is applicable only for Text Classification model.
No	`--d` *trueOrfalse*	Specifies whether to display a table with entity wise detailed analysis; the value must be `true` or `false`, as below: **true** Detailed evaluation results are required. **false** Detailed evaluation results are not required. The default is `false`. The *Model Evaluation Results* table, and *Confusion Matrix* with its columns, as described below, display the counts per entity. **Note:** If the command is run without this argument or with the argument value `false`, the *Model Evaluation Results* table and *Confusion Matrix* are not displayed. Only the *Model Evaluation Statistics* are displayed.

Output

Model Evaluation Statistics

Executing this command displays these evaluation statistics in a tabular format:

- **Precision**: It is a measure of exactness. Precision defines the proportion of correctly identified tuples.
- **Recall**: It is a measure of completeness of the results. Recall can be defined as a fraction of relevant instances that are retrieved.
- **F1 Measure**: It is the measure of the accuracy of a test. The computation of F1 score takes into account both precision and recall of the test. It can be interpreted as the weighted average of the precision and recall, where F1 score reaches its best value at 1 and worst at 0.
- **Accuracy**: It measures the degree of correctness of results. It defines the closeness of the measured value to the known value.

Model Evaluation Results

If the command is run with the argument `--d true`, the match counts of all the entities are displayed in a tabular format. The columns of the table are:

Input Count	The number of occurrences of the entity in the input data.
Mismatch Count	The number of times the entity match failed.
Match Count	The number of times the entity match succeeded.

Confusion Matrix

The *Confusion Matrix* (shown below) allows visualization of how an algorithm performs. It illustrates the performance of a classification model.

```
Confusion Matrix:
+------------------+-------+-----------+-----------+
| CONFUSION MATRIX |       | PREDICTED |           |
+------------------+-------+-----------+-----------+
|                  |       |  POSITIVE | NEGATIVE  |
|           ACTUAL |  TRUE |         3 |         0 |
|                  | FALSE |         0 |         0 |
+------------------+-------+-----------+-----------+
```

The column represents the instances in a predicted class while the row represents the instances in an actual class. Some of the terms associated with the confusion matrix are:

Actual	The number of occurrences of the entity in the actual class.
Predicted	The number of occurrences of the entity in the predicted class.
TP	**True Positive:** The number of entity occurrences predicted as positive and actually true as well.
TN	**True Negative:**The number of entity occurrences predicted as negative but actually true.
FP	**False Positive:** The number of entity occurrences predicted as positive but actually false.
FN	**False Negative:** The number of entity occurrences predicted as negative and actually false as well.

Example

This example:

- Evaluates the model called "MyModel"
- Uses a test file called "ModelTestFile" in the same location
- Stores the output of the evaluation in a file called "MyModelTestOutput"
- Specifies a category count of 4
- Specifies that a detailed analysis of the evaluation is required

```
iemodel evaluate model --n MyModel --t
C:\Spectrum\IEModels\ModelTestFile --o
C:\Spectrum\IEModels\MyModelTestOutput --c 4 --d true
```

iemodel evaluate train_model

The `iemodel evaluate train_model` command evaluates and trains an existing **Information Extraction Module** model. This function cannot be performed on a new model.

> **Note:** For better results on evaluation and training of an existing **Information Extraction Module**, use this command: `iemodel trainAndevaluate model`. For details, refer **iemodel trainAndevaluate model** on page 37.

Usage

`iemodel evaluate train_model --f` *trainingOptionsFile* `--u` *trueOrFalse* `--o` *outputFileName* `--c` *categoryCount* `--d` *trueOrfalse*

Required	Argument	Description
Yes	`--f` *trainingOptionsFile*	Specifies the name and location of the training options file used to train the model. Directory paths you specify here are relative to the location where you are running the Administration Utility.
No	`--u` *overWriteIfExists*	Specifies whether to overwrite the existing trained model (if one exists). *TrueOrFalse* is one of the following: **true** Overwrites the existing model. **false** Does not overwrite the existing model.
No	`--o` *outputFileName*	Specifies the name and location of the output file that will store the evaluation results.
No	`--c` *categoryCount*	Specifies the number of categories in the model; must be a numeric value. **Note:** It is applicable only for Text Classification model.
No	`--d` *trueOrfalse*	Specifies whether to display a table with entity wise detailed analysis; the value must be `true` or `false`, as below: **true** Detailed evaluation results are required. **false** Detailed evaluation results are not required.

Required	Argument	Description
		The default is `false`. The *Model Evaluation Results* table, with its columns as described below,displays the counts per entity. **Note:** If the command is run without this argument or with the argument value `false`, the Model Evaluation Results table is not displayed. Only the Model Evaluation Statistics are displayed.

Output

Model Evaluation Statistics

Executing this command displays these evaluation statistics in a tabular format:

- Precision
- Recall
- F1 Measure

Model Evaluation Results

If the command is run with the argument `--d true`, the match counts of all the entities are displayed in a tabular format. The columns of the table are:

Input Count	The number of occurrences of the entity in the input data.
Mismatch Count	The number of times the entity match failed.
Match Count	The number of times the entity match succeeded.

Example

This example:

- Uses a training options file called "ModelTrainingFile" that is located in "C:\Spectrum\IEModels"
- Overwrites any existing output file of the same name
- Stores the output of the evaluation in a file called "MyModelTestOutput"
- Specifies a category count of 4
- Specifies that a detailed analysis of the evaluation is required

```
iemodel evaluate train_model --f
C:\Spectrum\IEModels\ModelTrainingFile --u true --o
C:\Spectrum\IEModels\MyModelTestOutput --c 4 --d true
```

iemodel export

The `iemodel export` command exports an Information Extraction Module model and its metadata.

Usage

`iemodel export --n` *modelName* `--o` *outputDirectory*

Required	Argument	Description
Yes	`--n` *modelName*	Specifies the name of the model you want to export. Directory paths you specify here are relative to the location where you are running the Administration Utility.
Yes	`--o` *outputDirectory*	Specifies the location of the folder that will store the exported model and its metadata.

Example

This example exports a model named `MyModel` that places the output in a folder called "MyModelExport", which is located in "C:\Spectrum\IEModels\MyModelExport".

```
iemodel export --n MyModel --o
C:\Spectrum\IEModels\MyModelExport
```

iemodel import

The `iemodel import` command imports an Information Extraction Module model and its metadata.

Usage

`iemodel import --n` *modelName* `--o` *inputDirectory* `--u` *trueOrFalse*

Required	Argument	Description
Yes	`--n` *modelName*	Specifies the name of the model you want to import. Directory paths you specify here are relative to the location where you are running the Administration Utility.
Yes	`--o` *inputDirectory*	Specifies the location of the folder that will store the imported model and its metadata.

Required	Argument	Description
No	`--u` *overWriteIfExists*	Specifies whether to overwrite the existing model (if one exists). *TrueOrFalse* is one of the following: **true** Overwrites the existing model. **false** Does not overwrite the existing model.

Example

This example imports a model named `MyModel` that stores the model in a folder called "MyModelExport", which is located in "C:\Spectrum\IEModels\MyModelExport". It also overwrites any existing model of the same name.

```
iemodel import --n MyModel --o
C:\Spectrum\IEModels\MyModelExport --u true
```

iemodel list

The `iemodel list` command returns a list of all Information Extraction Module models.

Usage

```
iemodel list
```

Example

This example lists all models.

```
iemodel list
```

iemodel train

The `iemodel train` command trains an Information Extraction Module model. It calls your training options file, which points to your input file and applies the options you have specified.

Usage

`iemodel train --f` *trainingOptionsFile* `--u` *trueOrFalse*

Required	Argument	Description
Yes	`--f` *trainingOptionsFile*	Specifies the name and location of the training options file used to train the model. Directory paths you specify here are relative to the location where you are running the Administration Utility.
No	`--u` *trueOrFalse*	Specifies whether to overwrite the existing model with the same name (if one exists), where *TrueOrFalse* is one of the following: **true** Overwrites the existing model. **false** Does not overwrite the existing model.

Example

This example trains a model listed in the *TrainingOptions.xml* file that is stored the `C:` drive and overwrites any existing model of the same name.

```
iemodel train --f c:/TrainingOptions.xml --u true
```

iemodel trainAndevaluate model

The `iemodel trainAndevaluate model` command evaluates and trains a new model as well as an existing model. In case of an existing model you need to overwrite it with the newly trained model by using "true" for the argument --u in the command.

This command calls your training options file and provides an optional output file with evaluation results, should you choose to produce that file.

Usage

`iemodel trainAndevaluate model --f` *trainingOptionsFile* `--u` *trueOrFalse* `--o` *outputFileName* `--c` *categoryCount* `--d` *trueOrfalse*

Required	Argument	Description
Yes	`--f` *trainingOptionsFile*	Specifies the name and location of the training options file used to train the model. Directory paths you specify here are relative to the location where you are running the Administration Utility.
No	`--u` *overWriteIfExists*	Specifies whether to overwrite the existing trained model (if one exists). **true** Overwrites the existing model. **false** Does not overwrite the existing model.

Required	Argument	Description
No	`--o` *outputFileName*	Specifies the name and location of the output file that will store the evaluation results.
No	`--c` *categoryCount*	Specifies the number of categories in the model; must be a numeric value. **Note:** It is applicable only for Text Classification model.
No	`--d` *trueOrfalse*	Specifies whether to display a table with entity wise detailed analysis; the value must be `true` or `false`, as below: **true** Detailed evaluation results are required. **false** Detailed evaluation results are not required. The default is `false`. The *Model Evaluation Results* table, and *Confusion Matrix* with its columns, as described below, display the counts per entity. **Note:** If the command is run without this argument or with the argument value `false`, the *Model Evaluation Results* table and *Confusion Matrix* are not displayed. Only the *Model Evaluation Statistics* are displayed.

Output

Model Evaluation Statistics

Executing this command displays these evaluation statistics in a tabular format:

- **Precision**: It is a measure of exactness. Precision defines the proportion of correctly identified tuples.
- **Recall**: It is a measure of completeness of the results. Recall can be defined as a fraction of relevant instances that are retrieved.
- **F1 Measure**: It is the measure of the accuracy of a test. The computation of F1 score takes into account both precision and recall of the test. It can be interpreted as the weighted average of the precision and recall, where F1 score reaches its best value at 1 and worst at 0.
- **Accuracy**: It measures the degree of correctness of results. It defines the closeness of the measured value to the known value.

Model Evaluation Results

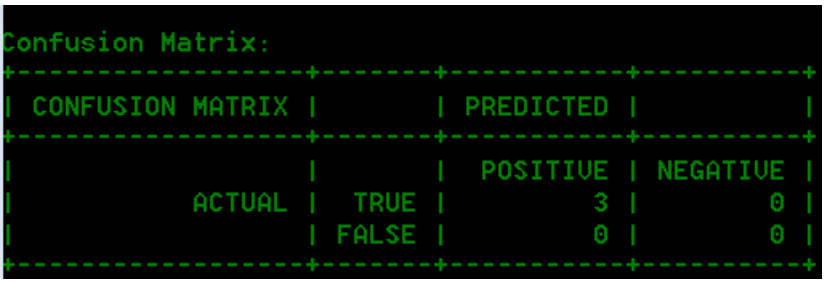
If the command is run with the argument `--d true`, the match counts of all the entities are displayed in a tabular format. The columns of the table are:

Input Count	The number of occurrences of the entity in the input data.

Mismatch Count	The number of times the entity match failed.
Match Count	The number of times the entity match succeeded.

Confusion Matrix

The *Confusion Matrix* (shown below) allows visualization of how an algorithm performs. It illustrates the performance of a classification model.



The column represents the instances in a predicted class while the row represents the instances in an actual class. Some of the terms associated with the confusion matrix are:

Actual	The number of occurrences of the entity in the actual class.
Predicted	The number of occurrences of the entity in the predicted class.
TP	**True Positive:** The number of entity occurrences predicted as positive and actually true as well.
TN	**True Negative:**The number of entity occurrences predicted as negative but actually true.
FP	**False Positive:** The number of entity occurrences predicted as positive but actually false.
FN	**False Negative:** The number of entity occurrences predicted as negative and actually false as well.

Example

This example:

- Uses a training options file called "ModelTrainingFile" that is located in "C:\Spectrum\IEModels"
- Overwrites any existing output file of the same name
- Stores the output of the evaluation in a file called "MyModelTestOutput"
- Specifies a category count of 4
- Specifies that a detailed analysis of the evaluation is required

```
iemodel trainAndevaluate model --f
C:\Spectrum\IEModels\ModelTrainingFile --u true --o
C:\Spectrum\IEModels\MyModelTestOutput --c 4 --d true
```

6 - Stages Reference

In this section

Information Extraction Components

The Information Extraction Module includes these stages.

- **Read From Documents**—Reads unstructured input data from various file formats and extracts the contents.
- **Entity Extractor**— Extracts entities such as names and addresses from unstructured data passed as strings.
- **Text Categorizer**— Assigns custom categories to unstructured content or plain text (such as email, news articles, and comments) based on how much of that content contains material for that category.
- **Relationship Extractor**— Extracts relationships between entities.

Read from Documents

Read from Documents is a source stage that reads unstructured input data from various file formats and extracts the contents. Possible sources include legal documents, customer feedback, product reviews, news articles, blogs, social networks, and so on. Read from Documents also extracts metadata fields such as author and creation date. Once the data has been extracted it can be used for various types of processing, including entity extraction and string manipulation among others. The data can also be used to build search indexes for unstructured text searches.

Note: Each document is considered one record for this stage.

Input

The input for Read from Documents is a single file or folder. This stage supports the following file types:

- Text
- PDF
- Microsoft Outlook
- Microsoft Word
- HTML

Read from Documents performs three types of extractions:

- Document—Use the entire document

- Page—Use a specific page of a document
- Selective—Use a selected part of a document
- Bookmarks—Use bookmarks from a PDF document

Read from Documents is part of the Information Extraction Module.

Options

File Properties Tab

The following table lists the options that control the type of information returned by Read from Documents.

Table 1: Read from Documents Options

Option	Description	
Server name	Specifies the name of the Spectrum Technology Platform server being used.	
File/folder name	The path and name of the source document or folder. If you want to point to a folder, use an asterisk as a wildcard character ("*") to select all files in the folder. If you want to point to multiple files of the same type within a folder, use the wildcard character plus the file extension ("*.pdf").	
File type	The source document's file type, which will automatically be selected once you select a source: • Text • PDF • Microsoft Outlook • Microsoft Word • HTML	
Extraction type	**Documentation**	Use the entire document.
	Page	Use a specific page of a document.
	Selection	Use a selected portion of a document.
	Bookmarks	Use bookmarks from a PDF document.

Option	Description
Page selection	Only with Page extraction type. Select all pages or a range of pages.
Selected extraction	Only with Selection extraction type. Specifies the type of search.
Specify text	Only with Selection extraction type. Specifies the text to look for.
Exclude start text	Only with Selection extraction type and Start text option. Omits entered string from the returned data.
Specify end text	Only with Selection extraction type. Specifies the end text to look for.
Exclude end text	Only with Selection extraction type. Omits entered string from the end of the returned data.
Selection return	Only with Selection extraction type. Specifies how many paragraphs to return for each result. For example, if you choose "2" here, the returned data for each result will include the paragraph the result is in plus the subsequent paragraph, totaling two paragraphs. Default is 1. Not valid when end text is specified.

Fields Tab

Click **Regenerate** to define input fields.

Table 2: Output Data Options

Option	Description
Attribute Name	Shows the attribute that is most like the input field. For instance, if one if your fields contains date information and you call it "Date," you will see the "Date" attribute assigned to that field. This column is not editable.
Name	The name of the field. This column is editable.

Option	Description
Type	The field's data type.
Include	Specifies which fields to be included in a search index.

Output

The Read from Documents stage has two outgoing ports. One port captures the data that was read by the stage and returned based on the criteria entered. It can include plain text or metadata (such as author, language, date created, and so on). This port can be connected to any stage that reads incoming data, such as Write to File or Write to XML, as well as primary stages such as Validate Address or Write to Search Index. It can also be connected to the Information Extractor stage if you want to return information about certain entity types that are in the document. When you select the Document extraction type the output will contain flat data; when you select the Page or Selection extraction type the output will contain hierarchical data.

The other port collects any records that the dataflow did not process correctly. This is called the Error Port, and records that pass through this port into the sink are considered malformed. Capturing malformed records can help you identify the problem with those records. When you attach a sink to the Error Port, the resulting output file will contain all the fields from the malformed records. It will also contain a Reason field that specifies why the record failed.

Table 3: Unstructured Reader Output

Field Name	Description / Valid Values
Author	Typically contains the name of the person who created or updated the document. This information is part of the document's metadata.
Bookmark	Contains all the bookmarks from the PDF input file. For Bookmarks extraction types only.
BookmarkNo	Contains all the bookmarks from the PDF input file. For Bookmarks extraction types only.

Field Name	Description / Valid Values
ContentLength	Indicates the length of the document. This value varies depending on the extraction type selected: **Document** The number of pages in the document. **Page** "1", to represent the single page of content.
Contents	Varies based on extraction type. For example, Document extraction types will output the entire document as flat data. Page, Selection, and Bookmarks extraction types will output hierarchical data.
ContentType	Indicates the type of document that was read, such as PDF, .txt, and so on.
Creator	Typically ontains the name of the person who created the document. This information is part of the document's metadata.
Date	Indicates the date the document was created or last updated.
Keywords	Contains any keywords that were provided in the document's metadata.
Language	Indicates the language in which the document was written.
NPages	Indicates the number of pages in the document.
PageContents	Contains the contents of the selected page(s). For Page extraction types only.
PageNo	Contains the page number for the bookmark. For Page extraction types only.
Parent	Contains the path of the bookmark, similar to XPath of an XML file. For Bookmarks extraction types only.
ResourceName	Indicates the file name of the document.

Field Name	Description / Valid Values
SectionContents	Contains the contents of the selected section. For Selection extraction types only.
SectionNo	Indicates the number of that section within the document. For Selection extraction types only.
Subject	Contains the subject of the document that was provided in the document's metadata.
Title	Contains the title of the document that was provided in the document's metadata.

Entity Extractor

Entity Extractor extracts entities such as names and addresses from strings of unstructured data (also known as plain text).

It is possible that not all entities for any selected type will be returned because accuracy varies depending on the type of input. Because Entity Extractor uses natural-language processing, a string containing a grammatically correct sentence from a news article or blog would likely have a more accurate return of names than a simple list of names and dates.

Input

Entity Extractor takes unstructured strings of data as the input. It can also use the **Read from Documents** stage as an input if you want to extract entities from an unstructured document. The **Read from Documents** stage reads the document and returns text based on the user-defined settings. The **Entity Extractor** extracts the required information from this text based on the selected entities.

Table 4: Input Format

Field Name	Description
PlainText	The unstructured string of data from which you want to extract information.

Options

Entity Extractor options enable you to select entities based on which you want to extract information from the input string. By default, you can extract information using *Person* and *Address* as the entity types. However, you can use the **Quick Add** function and select any or all of the 15 pre-configured entities.

Option Name	Description
Override system default options with the following values	Select the check box to override the default entity types *Address* and *Person*. When you select the check box, the **Quick Add** button gets enabled. Click this button and select the entities you need for extracting the text. The selected entities get added to the **Entity Type** list.

Option Name	Description
Entity Type	Specifies the type of data you want to extract from the unstructured string. ***Address*** ***CreditCard*** ***Date*** ***Email*** ***HashTag*** ***ISBN*** ***Location*** ***Mention*** ***Organization*** ***Person*** ***Phone*** ***ProperNouns*** ***SSN*** ***WebAddress*** ***ZipCode***
Output entities count	Specifies whether to return a count of the number of times a particular entity occurred in the output. **true** Return a count of the entities found in the unstructured string. **false** Do not return a count of the entities found in the unstructured string.

Output

The output from **Entity Extractor** is a list of the matching entities found in the input string. For example, if you selected an entity type as "Person," the output would contain a list of the person names found in the input string. Similarly, if you selected the **Entity Type** as "Date," the output will be a list of the dates found in the input string.

Each entity, whether a name, address, or date, is returned only once even if the entity appears multiple times in the input string.

To see the number of times the entity appeared in the input string you can select the **Output entity count** option in the **Entity Extractor Options** window.

Field Name	Description
Text	The text extracted from the string.
Type	The entity type of the extracted text. One of the following: ***Address*** ***CreditCard*** ***Date*** ***Email*** ***HashTag*** ***ISBN*** ***Location*** ***Mention*** ***Organization*** ***Person*** ***Phone*** ***ProperNouns*** ***SSN*** ***WebAddress*** ***ZipCode***
Count	If the option to return a count is enabled, this field contains the number of times a particular entity appeared in the input. For example, if you chose to return `Name` entities and the input text contained five instances of the name `John`, the name `John` will be included in the output just once, with `Name` as the entity type, and "5" as the output count.

Text Categorizer

This stage helps you assign custom categories to unstructured content or plain text (such as email, news articles, and comments) based on the extent of matching content it has. The stage lists the defined categories, from which you can select the one you need for your categorization. However, you need to create these categories by training a categorizer model with your data. For details, see **Introduction to Text Categorization** on page 7.

Input

The stage takes unstructured strings of data as input . It can also use the **Read from Documents** stage as an input if you want to categorize text from an unstructured document. The **Read from Documents** stage reads the document and returns text based on the user-defined settings. This is read by the **Text Categorizer** stage to give you the desired output.

Table 5: Input Format

Field Name	Description
PlainText	The unstructured string of data from which you want to extract information.

Options

The **Text Categorizer Options** gives you the choice of selecting parameters based on which you want to classify your input string of data. You can select the model for categorization and the number of matching levels to which you want the output. For example, only the closest match or closest plus the second close match.

Option Name	Description
Override system default options with the following values	To override the default option and select the categorizer from the **Categorizer name** drop down.
Categorizer name	Specifies the model to be used for text categorization. It lists all the models you trained in the text categorization phase. **Note:** For more information, see **Training the Model** on page 12.
Category count	The count of matching levels of category that you want in the output. For example, select 1 to display just the closest match and 2 to display the closest plus the second close match. **Note:** The maximum value corresponds to the number of different classes specified while training the model.

Output

The output lists the categories into which the content of the input string are classified and the rank of that category. The rank signifies how close the input content matches the category. For example, 1 means it is the closest match to the category and 2 means closest plus the next close match.

Field Name	Description
Category	The predicted category for each record in the input file.
Rank	The rank of categories from the highest score to the lowest score.

Relationship Extractor

The **Relationship Extractor** stage allows you to identify the relationship types between the identified entities in the source content.

The **Relationship Extractor** stage identifies:

1. Entity1
2. Entity1 Type
3. Relation Type
4. Entity2
5. Entity2 Type

Important: The stage tries to achieve the maximum possible accuracy while identifying the relationship types between any two entities in the input text. However, relationships other than the accurate relationship between the two entities can also be identified while parsing complicated sentences in the input text.

Input

The **Relationship Extractor** stage takes natural language strings of data as the input, and identifies the entities and the relationship types existing between each pair of entities.

Use the **Read from Documents** stage as a source stage if the input text is from an unstructured document. The **Read from Documents** stage reads the document and returns text based on the user-defined settings.

The **Relationship Extractor** stage then identifies all the entities and the relationship type existing between each pair of entities.

Table 6: Input Format

Field Name	Description
PlainText	The unstructured string of data from which you want to identify the relationship types existing between each pair of entities.

Options

The **Relationship Extractor** stage options enable you to specify which relationship types you wish to identify in the input text.

By default, the relationship types identified are:

1. *AffiliatedWith*
2. *LivesIn*
3. *OrgBasedIn*
4. *LocatedIn*

Option Name	Description
Override system default options with the following values	Select the check box to override the default relationship types identified and specify which relationship types you wish to identify and extract from the input text. On selecting the check box, the **Quick Add** button is enabled. Click **Quick Add** to select the relationship types you wish to identify in the text. The selected entities get added to the **Relationship Type** list.

Output

The output from **Relationship Extractor** is a list of the sets of relationships identified between pairs of entities found in the input string.

For example, if in the stage options, you have selected the *LivesIn* and *OrgBasedIn* relationship types to be extracted, the output contains a list of the all the sets of *Person LivesIn Location* and *Organization OrgBasedIn Location* identified in the input text.

Each entity pair with its binding relationship type is listed only once.

For each extracted set of entities and their relationship, the information extracted is:

Field Name	Description
Entity1	The first entity of a pair of entities extracted from the input text.
Entity1 Type	The entity type of the first entity of the pair of entities extracted from the input text. The entity type is any one of these: • *Person* • *Organization* • *Location*
Type	The relationship type identified between Entity1 and Entity2. For more information about relationship types, see **Relationship Types** on page 25. **Note:** Only the relationship types selected for extraction in the stage options are identified and listed.
Entity2	The second entity of a pair of entities extracted from the input text.
Entity2 Type	The entity type of the second entity of the pair of entities extracted from the input text. The entity type is any one of these: • *Person* • *Organization* • *Location*

Notices

USPS® Notices

Pitney Bowes Inc. holds a non-exclusive license to publish and sell ZIP + 4® databases on optical and magnetic media. The following trademarks are owned by the United States Postal Service: CASS, CASS Certified, DPV, eLOT, FASTforward, First-Class Mail, Intelligent Mail, LACS^{Link}, NCOA^{Link}, PAVE, PLANET Code, Postal Service, POSTNET, Post Office, RDI, Suite^{Link} , United States Postal Service, Standard Mail, United States Post Office, USPS, ZIP Code, and ZIP + 4. This list is not exhaustive of the trademarks belonging to the Postal Service.

Pitney Bowes Inc. is a non-exclusive licensee of USPS® for NCOA^{Link}® processing.

Prices for Pitney Bowes Software's products, options, and services are not established, controlled, or approved by USPS® or United States Government. When utilizing RDI™ data to determine parcel-shipping costs, the business decision on which parcel delivery company to use is not made by the USPS® or United States Government.

Data Provider and Related Notices

Data Products contained on this media and used within Pitney Bowes Software applications are protected by various trademarks and by one or more of the following copyrights:

© Copyright United States Postal Service. All rights reserved.

© 2014 TomTom. All rights reserved. TomTom and the TomTom logo are registered trademarks of TomTom N.V.

© 2016 HERE

Fuente: INEGI (Instituto Nacional de Estadística y Geografía)

Based upon electronic data © National Land Survey Sweden.

© Copyright United States Census Bureau

© Copyright Nova Marketing Group, Inc.

Portions of this program are © Copyright 1993-2007 by Nova Marketing Group Inc. All Rights Reserved

© Copyright Second Decimal, LLC

© Copyright Canada Post Corporation

This CD-ROM contains data from a compilation in which Canada Post Corporation is the copyright owner.

© 2007 Claritas, Inc.

The Geocode Address World data set contains data licensed from the GeoNames Project (**www.geonames.org**) provided under the Creative Commons Attribution License ("Attribution

License") located at **http://creativecommons.org/licenses/by/3.0/legalcode**. Your use of the GeoNames data (described in the Spectrum™ Technology Platform User Manual) is governed by the terms of the Attribution License, and any conflict between your agreement with Pitney Bowes Software, Inc. and the Attribution License will be resolved in favor of the Attribution License solely as it relates to your use of the GeoNames data.

3001 Summer Street
Stamford CT 06926-0700
USA

www.pitneybowes.com