Exposing Locale Properties

Overview

"Locale" values are language-identification codes that are temporarily attached to documents, fields, and field values during ingestion. The locale properties are consulted during text analysis to determine which language-specific tools should be applied to a particular field value.

The locale properties are normally invisible and ephemeral. They are not stored in the AIE index. This page demonstrates how to control locale detection, and how to make the resulting locale values persistent and visible. This is accomplished by creating a simple ingest field transformer in Java.

Related Topics

Language identification is performed by both Attivio Core Language Identification and by ALM Language Identification.

Locale codes are mapped to language-specific tokenizers in **<project-dir>\conf\features\core\TokenizerModel...** files.

Non-English Queries also use language codes to guide text analysis of the query.

The RouteDocumentByLocale

ingest router can be used to send documents with specific locale values to special workflows.

View incoming links. **>>**

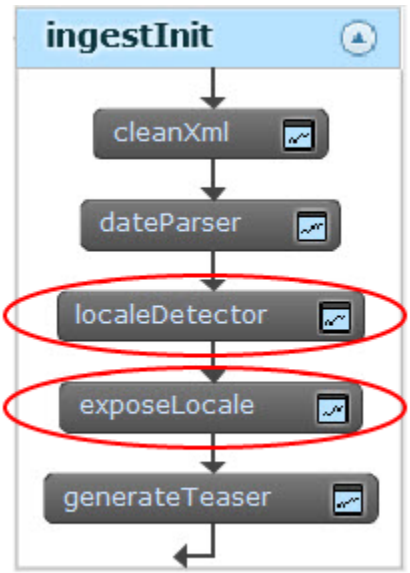
ALM Language Identification , Attivio Core Language Identification , Non-English Queries

Creating Custom Code

This section presents a simple example of extending the AIE ingestion process with a field transformer that makes locale values persistent and visible in search results. Tasks include:

- Setting up appropriate multi-language data to run the test on. We based the exercise on the Quick Start Tutorial Factbook demo by changing the list of RSS feeds loaded by the **news** connector in that project.
- Creating a new field transformer in Java. The transformer retrieves the locale settings of the document, the text field, and a title field value, and copies them into persistent string fields.
- Wrapping the new transformer in a new component in the AIE Administrator.
- Putting the new component in the **ingestInit** workflow (downstream of **localeDetector**).
- Configuring the **localeDetector** component to perform locale detection on the **text** and **title** fields of each incoming news article.
- Viewing news articles in SAIL to see the locale values.

The **localeDetector** component (in the **ingestInit** workflow) is the site of locale detection for both Attivio Core Language Identification and for ALM Language Identification. This exercise will insert a new component, **exposeLocale**, after **localeDetector** in the workflow.



Set up Multilingual News Feeds

To see locale detection in action, we need some documents written in multiple languages. The easy way to do this in AIE is to implement the Quick Start Tutorial Factbook demo, but change the RSS feeds that are loaded by the **news** connector.

You can edit **<install-dir>\conf\sdk\content\news.csv** in Excel or in any simple text editor. You can copy our feeds or find your own:

<install-dir>\conf\sdk\content\news.csv

```
source_s,uri
BBC,http://feeds.bbci.co.uk/news/rss.xml?edition=int
BBC,http://www.bbc.co.uk/zhongwen/simp/index.xml
LeMonde,http://www.lemonde.fr/rss/tag/ameriques.xml
Spiegel,http://www.spiegel.de/schlagzeilen/eilmeldungen/index.rss
ElUniversal,http://www.eluniversal.com.mx/rss/mexico.xml
JornalDeNoticias,http://feeds.jn.pt/JN-ULTIMAS
```

Note that the comma in each line must not have a space after it.

Create a Custom Field Transformer in Java

Based on DemoFieldTransformer.java

This example is based on the DemoFieldTransformer.java example from Creating Custom Ingest Transformers. All of the setup is the same up to the moment when you type in the name of the new class and begin to edit the file. Use the class name "ExposeLocaleSettings" based on the **DocumentModifyingTransformer** interface. Then implement the code shown below.

The ExposeLocaleSettings.java file is attached to this page.

A field transformer is one that operates on the fields of an IngestDocument. In this case we are going to extract the hidden locale properties of the document, the fields and the field values, and then copy them into persistent string fields for later examination.

The top of the file declares the package and imported classes, and then opens the class definition:

ExposeLocaleSettings.java

```
package com.acme.examples;

import java.util.Locale;
import com.attivio.model.AttivioException;
com.attivio.sdk.ingest.IngestDocument;
import com.attivio.model.document.Field;
import com.attivio.model.document.FieldValue;
import com.attivio.sdk.server.component.ingest.DocumentModifyingTransformer;

public class ExposeLocaleSettings implements DocumentModifyingTransformer  {
```

All ingest transformers have a **processDocument()** method which is automatically invoked by AIE:

ExposeLocaleSettings.java

```
    public boolean processDocument(IngestDocument doc) throws AttivioException {
```

The first transformation is to get the document locale setting and put it into a string field for indexing.

ExposeLocaleSettings.java

```
            // Get locale of entire document
            doc.setField("docLocale_s",doc.getLocale().getLanguage());
```

The **getLocale()** method returns a Locale object, which contains more information than we need. All we want is the two-letter language code, so we add **getLanguage()** to extract the code from the Locale object.

Note that you can make up a field name (such as "docLocale"), but it will not be indexed without adding that field to the project's schema.xml file. That involves editing the schema file and restarting AIE. The short cut is to add the suffix "_s" to the end of the field name. This suffix tells AIE to treat the field as an indexed string field, and bypasses the need to edit the schema. (See Indexing Dynamic Fields with Wildcards for additional schema shortcuts.)

The next section gets the locale of the text field.

ExposeLocaleSettings.java

```
            // Get locale of the text field
            IngestField textField = doc.getField("text");
            doc.setField("textFieldLocale_s", textField.getLocale().getLanguage());
```

A Field is an object with properties. We use **getLocale().getLanguage()** to get the two-letter language code, and then put it in a new string field, **textFieldLocale_s**. Again, this is a new string field that will be indexed.

Finally, we'll get the locale of the value of the **title** field.

ExposeLocaleSettings.java

```
            // Get locale of an individual field value
            IngestField titleField = doc.getField("title");
            IngestFieldValue firstValue = titleField.getValue(0);
            doc.setField("titleValueLocale_s", firstValue.getLocale().getLanguage());
```

First retrieve the **title** field of the document. From that, we'll extract the first (usually the only) value. Once we have the value, getLocale().getLanguage() will retrieve the two-letter code. The code goes into the new **titleValueLocale_s** string field.

The processDocument() method must return a Boolean value.

ExposeLocaleSettings.java

```
            return true; // to continue processing document
            }
        }
```

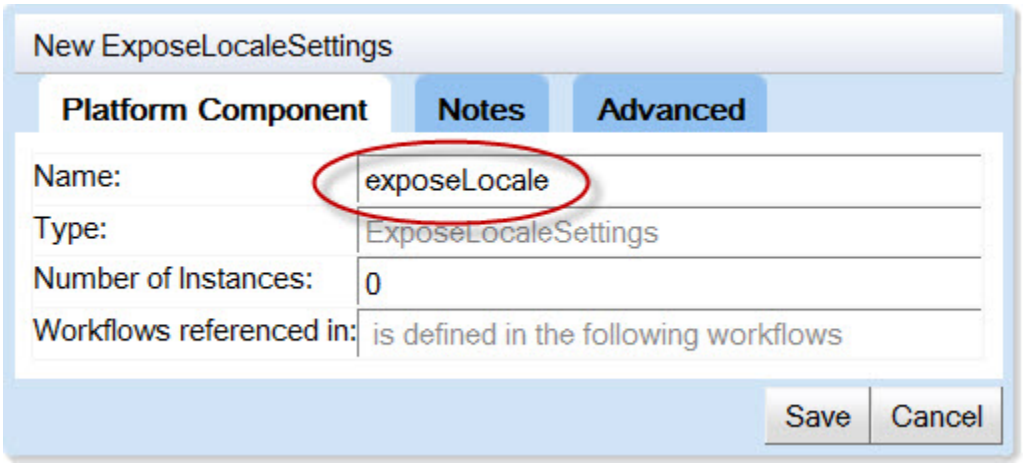
"True" indicates that the document can be released to the next component for further processing. "False" means that the document has suffered some kind of error and must be dropped.

Save the file. Compile the project and then deploy it to the AIE nodes.

Create a Component

We need to configure a new component, **exposeLocale**, to encapsulate the **ExposeLocaleSettings** instance.

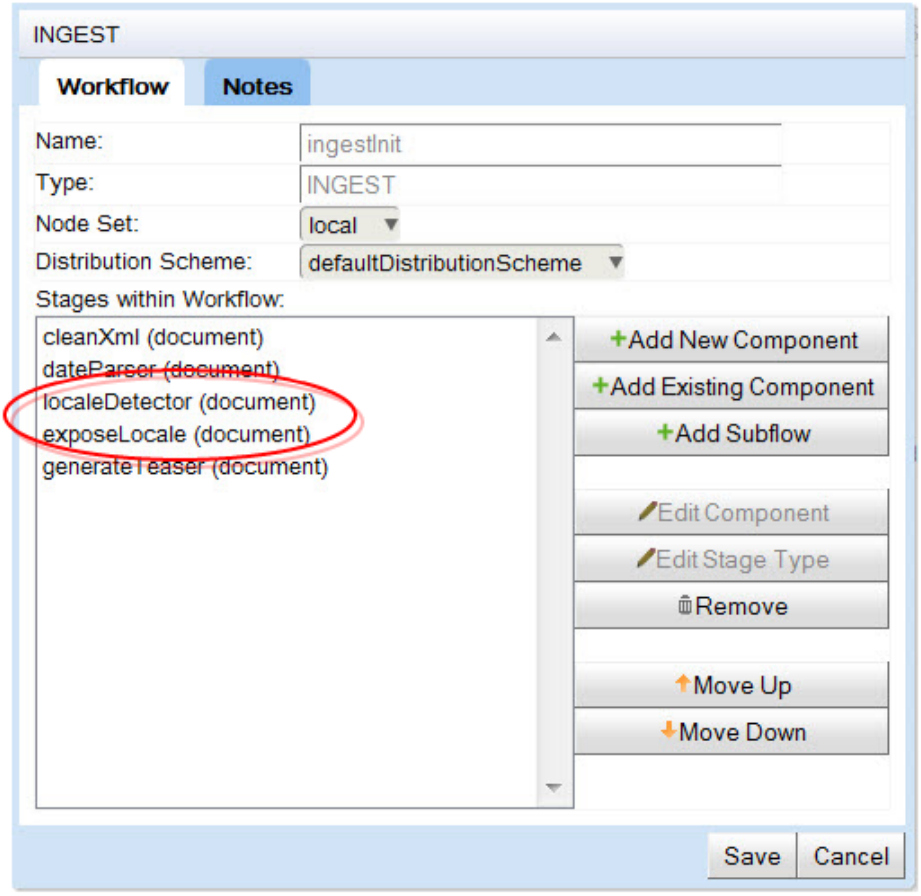
1. From the AIE CLI, execute the **start all** command.
2. Direct your browser to **http://localhost:17000/admin** (or substitute your host and port).
3. Open the **Palette**.
4. Click the **New** button. This opens a dialog box to select a platform component type.
5. Open the list of **Document Transformers**.
6. Scroll down to **ExposeLocaleSettings.** Select it and click **OK**. This opens a **Component Editor**.
7. On the **Platform Component** tab of the editor, enter the component's name. We used "exposeLocale."
8. When finished, click the **Save** button.



Modify the IngestInit Workflow

The next step is to insert the component into the **ingestInit** workflow, just after the **localeDetector** component.

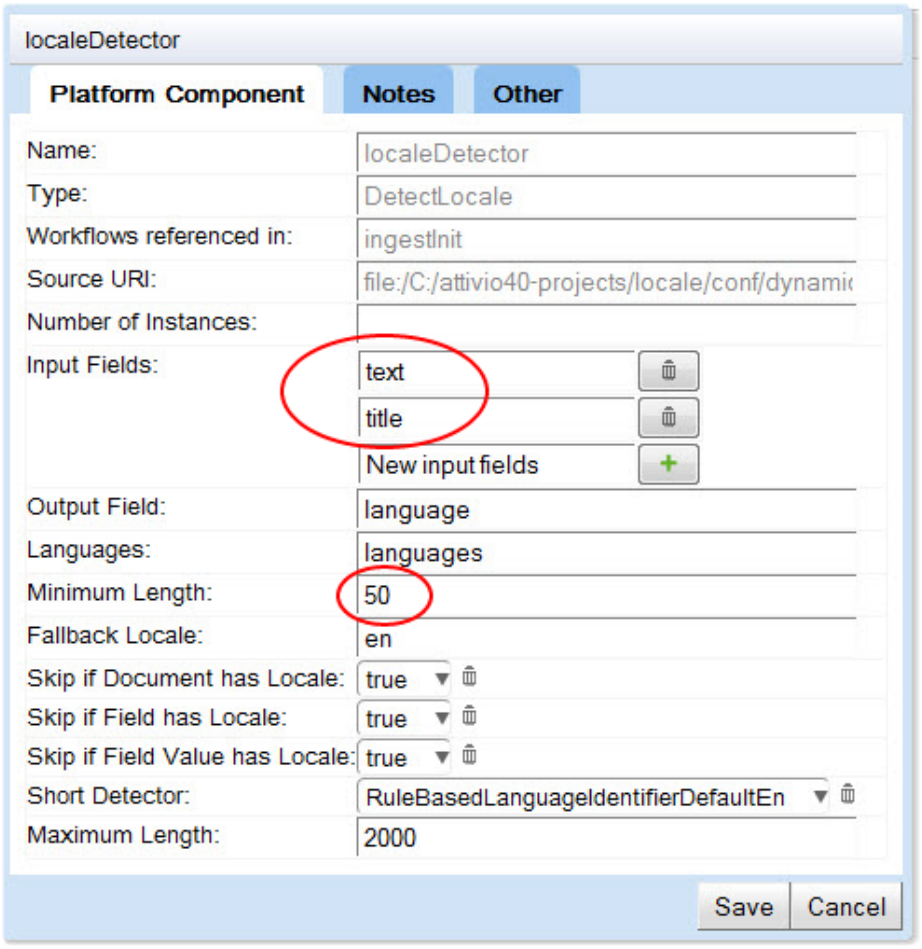
1. Navigate to **System Manager > Workflows > Ingest** in the AIE Administrator interface.
2. Open the **ingestInit** workflow in an editor.
3. Use the **Add Existing Component** button to add the **exposeLocale** component to the workflow.
4. Use the **Move Up** button to move **exposeLocale** to a position just beneath the **localeDetector**..
5. Click the **Save** button.



Configure the LocaleDetector Component

By default, **localeDetector** operates on the document's **text** field only. We want it to set the locale of the **title** field, too, so we'll have to reconfigure it.

Navigate to **System Management > Palette** in the AIE Administrator. Open the **localeDetector** component in an editor.



The default input field is **text**. Add the **title** field as a second input field and save the component.

The **Minimum Length** parameter is important (although you do not need to change the setting at this time). Locale detection relies on a significant sample of text. This parameter indicates the smallest sample of text that can produce a reliable result. If the available text is under 50 characters, locale detection is unreliable and is not attempted. Instead, the text falls through to a "Short Detector" function.

The default Short Detector is called **RuleBasedLanguageIdentifierDefaultEn**. This is an instance of RuleBasedLanguageIdentifier, configured in **<project-dir>\conf\beans\RuleBasedLanguageIdentifierDefaultEn.xml**, which can identify Chinese, Japanese, Korean, Hebrew, Russian and Arabic from very small samples of the characters used. It also recognized Latin characters, which it assumes indicate English.

If the Short Detector fails to assign a locale, the **Fallback Locale** is used.

Once the locale of the first field value is determined, it is automatically assigned as the locale of the field and of the document. At the same time, the human-readable name of the language overwrites the value in the **language** field, and is added to the list of languages in the **languages** field.

Load News Feeds

To try out the new transformer, run the Factbook **news** connector.

1. AIE should already be running at this point, because the previous step involved dynamic component editing.
2. Browse to **System Management -> Connectors**.
3. Check the box beside **news** connector and click the **Start** link. It will claim to have loaded six documents, but each one is an RSS feed containing numerous news articles.

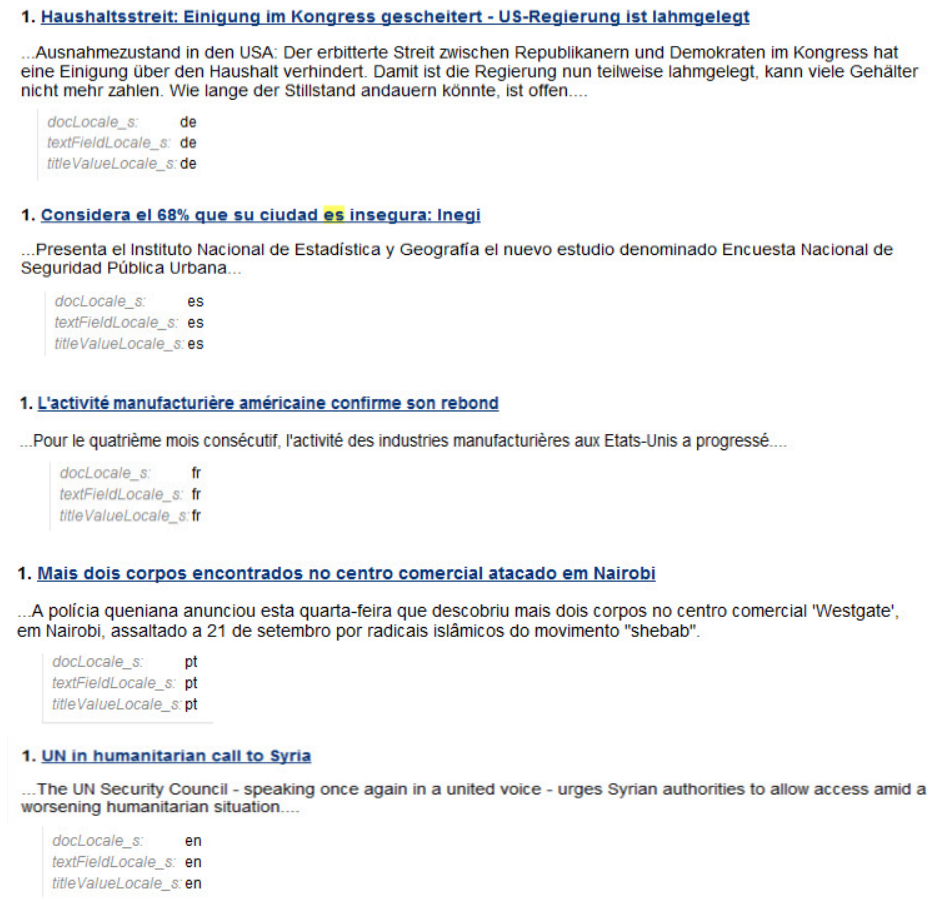
Search for News Articles

Open the Factbook search page and look for the new locale string fields.

1. Browse to the **Query -> SAIL** search page.
2. Using the **Simple Query Language**, search for ***:***.

3. Open the **Search Options** area and check the **Debug** checkbox. Click the **Search** button.
4. Look for the **docLocale_s**, **textFieldLocale_s**, and **titleValueLocale_s** fields in the results. The values of these fields are the locale settings for the respective document, field and value.

The following image is a mosaic of search results in various languages and their corresponding locale fields.




```
package com.acme.examples;

import com.attivio.model.AttivioException;
import com.attivio.sdk.ingest.IngestDocument;
import com.attivio.sdk.ingest.IngestField;
import com.attivio.sdk.ingest.IngestFieldValue;
import com.attivio.sdk.server.component.ingest.DocumentModifyingTransformer;

public class ExposeLocaleSettings implements DocumentModifyingTransformer {

    public boolean processDocument(IngestDocument doc) throws AttivioException {
```