Spectrum™ Technology Platform

Version 8.0.0 SP3

Templates Guide

Pitney Bowes Software

© 2013 Pitney Bowes Software Inc. All rights reserved. MapInfo and Group 1 Software are trademarks of Pitney Bowes Software Inc. All other marks and trademarks are property of their respective holders.

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.

Centrus Notices

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

© Copyright United States Postal Service. All rights reserved.

© 2011 TomTom. All rights reserved. TomTom and the TomTom logo are registered trademarks of TomTom N.V.

© Copyright NAVTEQ. All rights reserved

© 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 Canada Post Corporation

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

© 2007 Claritas, Inc.

ICU Notices

Copyright © 1995-2011 International Business Machines Corporation and others.

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

GeoNames Notices

The World Geocoding 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.

Pitney Bowes Software
Documentation Team
pbbidocs@pb.com

July 16, 2013

Contents

1

Introduction

In this section:

What are Dataflow Templates?

Dataflow templates illustrate ways in which you can use Spectrum™ Technology Platform and its modules to meet your business needs. They show how particular modules solve various requirements, such as parsing, standardizing, and validating names and addresses, geocoding addresses, and so on.

Dataflow templates are delivered with each module that you license. For instance, if you are licensed for the Data Normalization Module, you receive the Standardizing Personal Names dataflow template. If you are licensed for the Universal Addressing Module, you receive the Validating U.S. and Canadian Addresses dataflow templates.

The supporting documentation for the dataflow templates is broken down into three chapters: Name Data Quality, Address Data Quality, and Location Data Quality.

The dataflow templates and corresponding documentation are delivered with Spectrum™ Technology Platform in the following modules and chapters:

Table 1: Dataflow Templates

Module	Chapter	Dataflow Template
Universal Name Module	Name Data Quality	Parsing Personal Names
Data Normalization Module	Name Data Quality	Standardizing Personal Names
Advanced Matching Module	Name Data Quality	Identifying Members of a Household
Advanced Matching Module	Name Data Quality	Determining if a Prospect is a Customer
Data Normalization Module	Address Data Quality	Parsing Addresses
Universal Addressing Module	Address Data Quality	Validating U.S. and Canadian Addresses
Address Now Module	Address Data Quality	Validating International Addresses
Enterprise Geocoding Module	Location Data Quality	Geocoding Addresses
Enterprise Tax Module	Location Data Quality	Determining Tax Jurisdiction
Location Intelligence Module	Location Data Quality	Determining Insurance Rating Territory

Depending on the purpose of each template, it may be delivered as a job with sample data or it may be delivered as a service with no sample data. You can use dataflows in their original state and run those that are delivered as jobs to see how they function. Alternatively, you can manipulate the dataflows by changing input and output files or by bringing services into your own jobs and adding input and output files.

Note: These samples are intended as illustrations of various Spectrum™ Technology Platform features. They are not intended to be complete solutions to your particular business environment.

Creating a Dataflow Using a Template

Dataflow templates are delivered with each module that you license. To create a dataflow using a template,

- In Enterprise Designer go to **File** > **New** > **Dataflow** > **From Template** .
- Or, you can click the **New** icon and select New Dataflow From Template.

A list of templates available for the modules you have installed is displayed.

Running Dataflow Template Jobs

Follow the steps below to run a dataflow template job.

1. In Enterprise Designer, go to **File** > **New** > **Dataflow** > **From Template**. Alternatively, click the **New** icon and select New Dataflow From Template.
2. Select the dataflow template job you want to work with.
3. If necessary, add to or modify the dataflow template job to include your own input and output files.
4. Click **Run** > **Run Current Flow** or click the **Run** button.

Running Dataflow Template Services

Follow the steps below to run a dataflow template service.

1. In Enterprise Designer, go to **File** > **New** > **Dataflow** > **From Template**. Alternatively, click the **New** icon and select New Dataflow From Template.
2. Select the dataflow template service you want to work with.
3. If necessary, add to or modify the service to include your own input and output files.
4. Expose the service by selecting **File** > **Expose/Unexpose** and clicking **Save**.
5. Access the service in Management Console or Interactive Driver:
6. Open Management Console.
7. Under the Modules node, navigate to the service you exposed.

2

Name Data Templates

In this section:

Parsing Personal Names

This sample demonstrates how to take personal name data (for example "John P. Smith"), parse it into first name, middle name, and last name parts, and add gender data.

Template name: ParsePersonalName.df

Sample input file name: CDQ_Core_test.csv

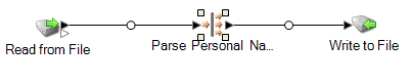
Modules required: Universal Name Module

Business Scenario

You work for an insurance company that wants to send out personalized quotes based on gender to prospective customers. Your input data include name data as full names and you want to parse the name data into First, Middle, and Last name fields. You also want to determine the gender of the individuals in your input data.

Solution

The following dataflow provides a solution to the business scenario:



In this dataflow, data is read from a file and processed through the Name Parser stage. (Name Parser is part of the Universal Naming Module.) For each name, Name Parser will do the following:

Read from File

This stage identifies the file name, location, and layout of the file that contains the names you want to parse. The file contains both male and female names.

Name Parser

In this template, the Name Parser stage is named Parse Personal Name. Parse Personal Name stage examines name fields and compares them to name data stored in the Spectrum™ Technology Platform name database files. Based on the comparison, it parses the name data into First, Middle, and Last name fields, assigns an entity type, and a gender to each name. It also uses pattern recognition in addition to the data.

In this template the Parse Personal Name stage is configured as follows.

- Parse personal names is selected and Parse business names is cleared. When you select these options, first names are evaluated for gender, order, and punctuation and no evaluation of business names is performed.
- Gender Determination Source is set to default. For most cases, Default is the best setting for gender determination because it covers a wide variety of names. However, if you are processing names from a specific culture, select that culture. Selecting a specific culture helps ensure that the proper gender is assigned to the names. For example, if you leave Default selected, then the name Jean will be identified as a female name. However, if you select French, it will be identified as a male name.
- Order is set to natural. The name fields are ordered by Title, First Name, Middle Name, Last Name, and Suffix.
- Retain periods is cleared. Any punctuation in the name data is not retained.

Write to File

The template contains one Write to File stage. In addition to the input fields, the output file contains the FirstName, MiddleName, LastName, EntityType, GenderCode, and GenderDeterminationSource fields.

Standardizing Personal Names

This sample demonstrates how to take personal name data (for example "John P. Smith"), identify common nicknames of the same name, and create a standard version of the name that can then be used to consolidate redundant records. It also show how you can add Title of Respect data based on Gender data.

Template name: StandardizePersonalName.df

Sample input file name: CDQ_Core_test.csv

Modules required: Universal Name Module and Data Normalization Module

Business Scenario

You work for a non-profit organization that wants to send out invitations for a gala event. Your input data include name data as full names and you want to parse the name data into First, Middle, and Last name fields and add a Title of Respect field to make your invitations more formal. You also want to replace any nicknames in your name data to use a more formal variant of the name.

Solution

The following dataflow provides a solution to the business scenario:

In this dataflow, data is read from a file and processed through the **Name Parser** on page 14, **Transformer** on page 14, and **Standardization** on page 15 stages. (Name Parser is part of the Universal Naming Module.) For each data row in the input file, this data flow will do the following:

Read from File

This stage identifies the file name, location, and layout of the file that contains the names you want to parse. The file contains both male and female names.

Name Parser

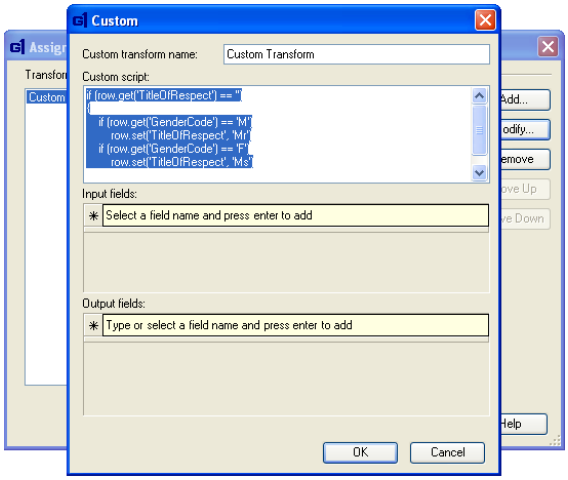
In this template, the Name Parser stage is named Parse Personal Name. Parse Personal Name stage examines name fields and compares them to name data stored in the Spectrum™ Technology Platform name database files. Based on the comparison, it parses the name data into First, Middle, and Last name fields, assigns an entity type, and a gender to each name. It also uses pattern recognition in addition to the name data.

In this template the Parse Personal Name stage is configured as follows.

- Parse personal names is selected and Parse business names is cleared. When you select these options, first names are evaluated for gender, order, and punctuation and no evaluation of business names is performed.
- Gender Determination Source is set to default. For most cases, Default is the best setting for gender determination because it covers a wide variety of names. However, if you are processing names from a specific culture, select that culture. Selecting a specific culture helps ensure that the proper gender is assigned to the names. For example, if you leave Default selected, then the name Jean will be identified as a female name. However, if you select French, it will be identified as a male name.
- Order is set to natural. The name fields are ordered by Title, First Name, Middle Name, Last Name, and Suffix.
- Retain periods is cleared. Any punctuation in the name data is not retained.

Transformer

In this template, the Transformer stage is named Assign Titles. Assign Titles stage uses a custom script to search each row in the data stream output by the Parse Personal Name stage and assign a **TitleOfRespect** value based on the **GenderCode** value.



The custom script is:

```
if (row.get('TitleOfRespect') == '')
{
 if (row.get('GenderCode') == 'M')
  row.set('TitleOfRespect', 'Mr')
 if (row.get('GenderCode') == 'F')
  row.set('TitleOfRespect', 'Ms')
```

Every time the Assign Titles stage encounters `M` in the **GenderCode** field it sets the value for **TitleOfRespect** as `Mr`. Every time the Assign Titles stages encounters `F` in the **GenderCode** field it sets the value of **TitleOfRespect** as `Ms`.

Standardization

In this template, the Standardization stage is named Standardize Nicknames. Standardize Nickname stage looks up first names in the Nicknames.xml database and replaces any nicknames with the more regular form of the name. For example, the name Tommy is replaced with Thomas.

Write to File

The template contains one Write to File stage. In addition to the input fields, the output file contains the TitleOfRespect, FirstName, MiddleName, LastName, EntityType, GenderCode, and GenderDeterminationSource fields.

Identifying Members of a Household

This sample demonstrates how to identify members of the same household by comparing information in the Name and AddressLine1 field and creating an output file of household collections.

Template name: HouseholdRelationships.df

Sample input file name: CDQ_Core_test.csv

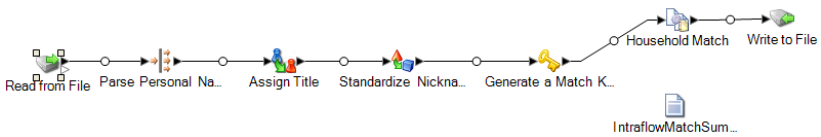
Modules required: Advanced Matching Module, Data Normalization Module, and Universal Name Module

Business Scenario

As data steward for a new internet credit card company and you want to analyze your customer database and find out which addresses occur multiple times and under what names so that you can minimize that number of duplicate mailings and credit card offers sent to the same address.

Solution

The following dataflow provides a solution to the business scenario:



In this dataflow, data is read from a file and processed through the **Name Parser** on page 14, **Transformer** on page 14, **Standardization** on page 15, **Match Key Generator** on page 17, and **Intraflow Match** on page 18 stages. For each data row in the input file, this data flow will do the following:

Read from File

This stage identifies the file name, location, and layout of the file that contains the names you want to parse. The file contains both male and female names.

Name Parser

In this template, the Name Parser stage is named Parse Personal Name. Parse Personal Name stage examines name fields and compares them to name data stored in the Spectrum™ Technology Platform name database files. Based on the comparison, it parses the name data into First, Middle, and Last name fields, assigns an entity type, and a gender to each name. It also uses pattern recognition in addition to the name data.

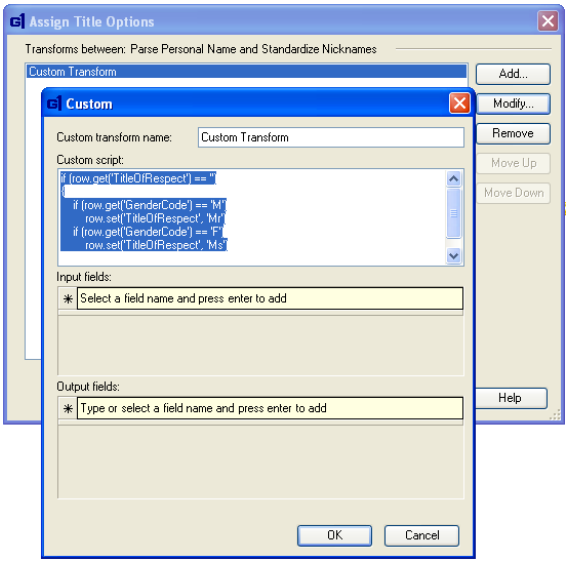
In this template the Parse Personal Name stage is configured as follows.

- Parse personal names is selected and Parse business names is cleared. When you select these options, first names are evaluated for gender, order, and punctuation and no evaluation of business names is performed.
- Gender Determination Source is set to default. For most cases, Default is the best setting for gender determination because it covers a wide variety of names. However, if you are processing names from a specific culture, select that culture. Selecting a specific culture helps ensure that the proper gender is assigned to the names. For example, if you leave Default selected, then the name Jean will be identified as a female name. However, if you select French, it will be identified as a male name.
- Order is set to natural. The name fields are ordered by Title, First Name, Middle Name, Last Name, and Suffix.

- Retain periods is cleared. Any punctuation in the name data is not retained.

Transformer

In this template, the Transformer stage is named Assign Titles. Assign Titles stage uses a custom script to search each row in the data stream output by the Parse Personal Name stage and assign a **TitleOfRespect** value based on the **GenderCode** value.



The custom script is:

```
if (row.get('TitleOfRespect') == '')
{
 if (row.get('GenderCode') == 'M')
  row.set('TitleOfRespect', 'Mr')
 if (row.get('GenderCode') == 'F')
  row.set('TitleOfRespect', 'Ms')
```

Every time the Assign Titles stage encounters `M` in the **GenderCode** field it sets the value for **TitleOfRespect** as `Mr`. Every time the Assign Titles stages encounters `F` in the **GenderCode** field it sets the value of **TitleOfRespect** as `Ms`.

Standardization

In this template, the Standardization stage is named Standardize Nicknames. Standardize Nickname stage looks up first names in the Nicknames.xml database and replaces any nicknames with the more regular form of the nickname. For example, the name Tommy is replaced with Thomas.

Match Key Generator

The Match Key Generator processes user-defined rules that consist of algorithms and input source fields to generate the match key field. A match key is a non-unique key shared by like records that identify records as potential duplicates. The match key is used to facilitate the matching process by only comparing

records that contain the same match key. A match key is comprised of input fields. Each input field specified has a selected algorithm that is performed on it. The result of each field is then concatenated to create a single match key field.

In this template, two match key fields are defined: SubString (LastName (1:3)) and SubString (PostalCode (1:5)).

For example, if the incoming address was:

FirstName - Fred

LastName - Mertz

PostalCode - 21114-1687

And the rules specified that:

Input Field	Start Position	Length
LastName	1	3
PostalCode	1	5

Then the key, based on the rules and the input data shown above, would be:

Mer21114

Intraflow Match

The Intraflow Match component locates matches between similar data records within a single input stream. Matched records can also be qualified by using non-name/non-address information. The matching engine allows you to create hierarchical rules based on any fields that have been defined or created in other components.

A stream of records to be matched as well as settings that specify what fields should be compared, how scores should be computed, and generally what constitutes a successful match.

In this template, you create a custom matching rule that compares LastName and AddressLine1. Select the **Generate data for analysis** check box to generate data for the **Intraflow Summary Report** on page 19.

Here are some guidelines to follow when creating your matching hierarchy:

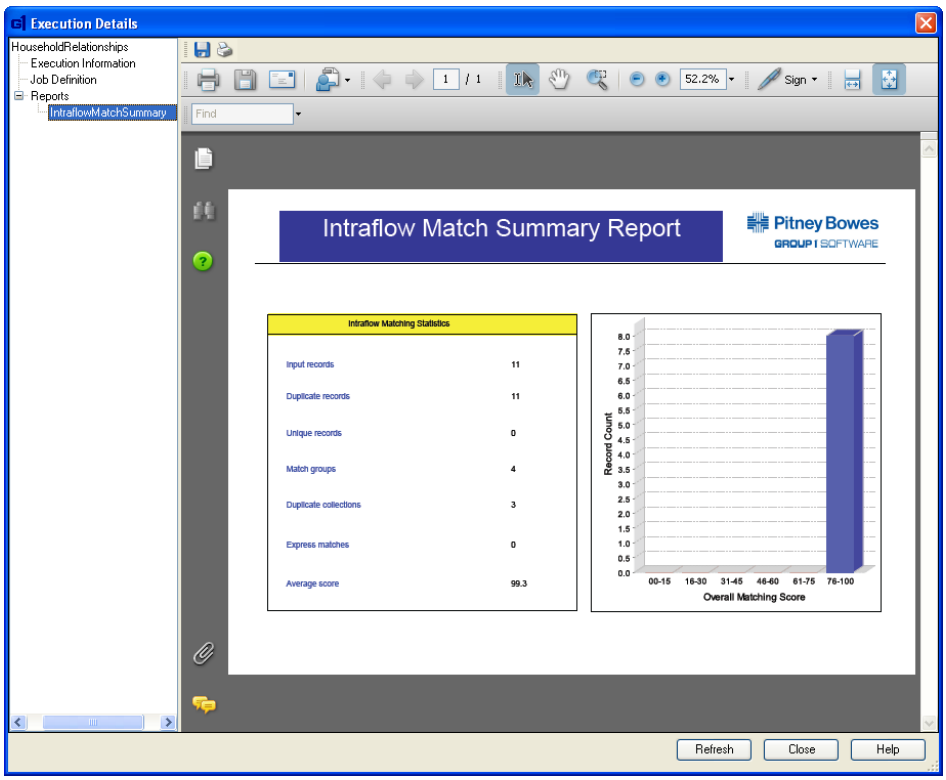
- A parent node must be given a unique name. It can not be a field.
- The child field must be a Spectrum™ Technology Platform data type field, that is, one available through one or more components.
- All children under a parent must use the same logical operators. To combine connectors you must first create intermediate parent nodes.
- Thresholds at the parent node could be higher than the threshold of the children.
- Parent nodes do not have to have a threshold.

Write to File

The template contains one Write to File stage that creates a text file that shows the addresses as a collection of households.

Intraflow Summary Report

The template contains the Intraflow Match Summary Report. After you run the job, expand **Reports** in the Execution Details window, and then click **IntraflowMatchSummary**.



The Intraflow Match Summary Report lists the statistics for the records processed and shows a bar chart that graphically illustrates the record count and overall matching score.

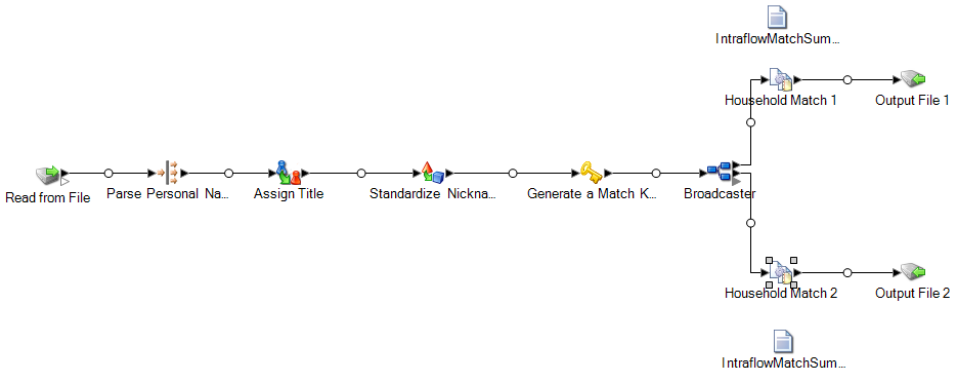
Alternate Solution

Template name: HouseholdRelationshipsAnalysis.df

Sample input file name: CDQ_Core_test.csv

Modules required: Advanced Matching Module and Universal Name Module

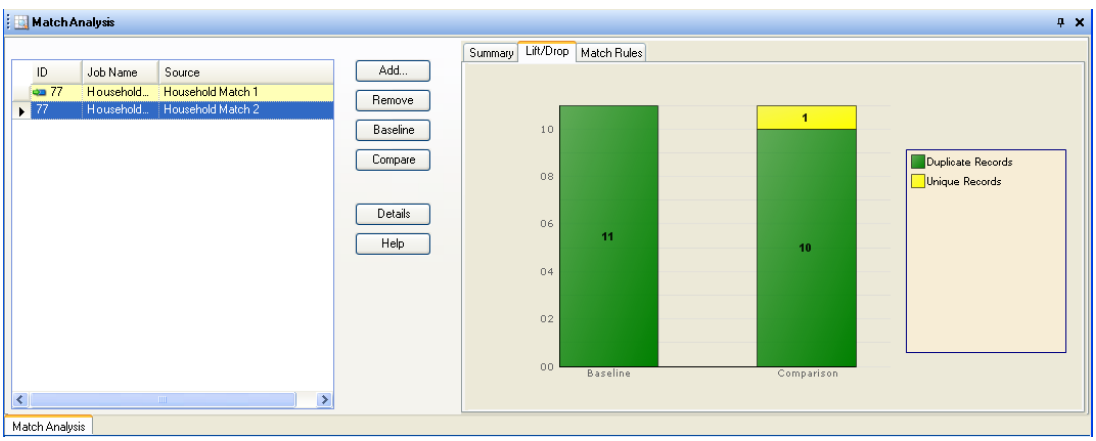
The following dataflow provides a solution to the business scenario:



In this dataflow, data is read from a file and processed through the **Name Parser** on page 14, **Transformer** on page 14, **Standardization** on page 15, and **Match Key Generator** on page 17. Before the data is sent through a matcher, it is split into two streams using a Broadcaster. Each stream is then sent through an Intraflow Match stage. Each data stream includes identical copies of the processed data. Each Intraflow Match stage uses different matching algorithm and generates Match Analysis data that you can use to compare the lift/drop of various matches.

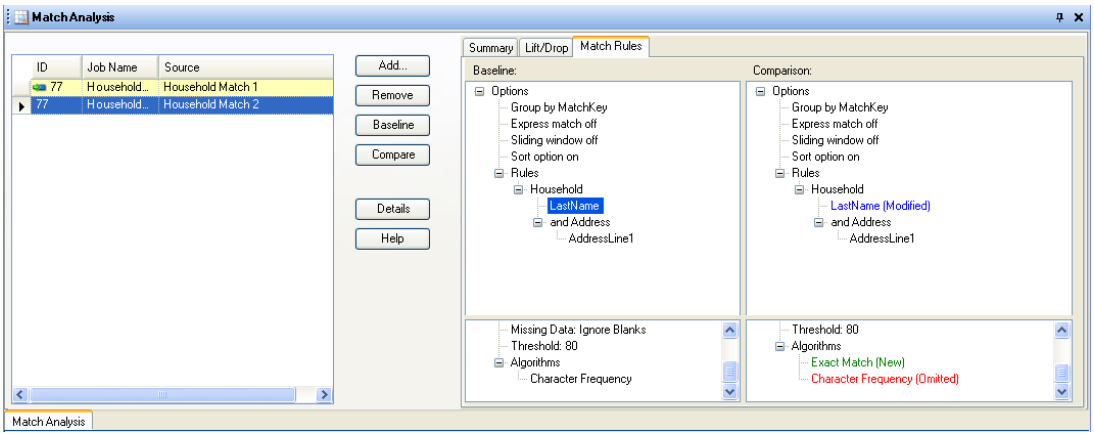
To use Match Analysis:

1. Run the dataflow template listed above.
2. From the **Tools** menu, select **Match Analysis**.
3. From Browse Match Results window, expand **HouseholdRelationshipAnalysis**, select **Household Match 1** and **Household Match 2** from the Source list, and then click **Add**.
4. Select **Household Match 1** in the Match Results List and click **Compare**. The Summary Results display.
5. Click the **Lift/Drop** tab. The Lift/Drop chart displays.



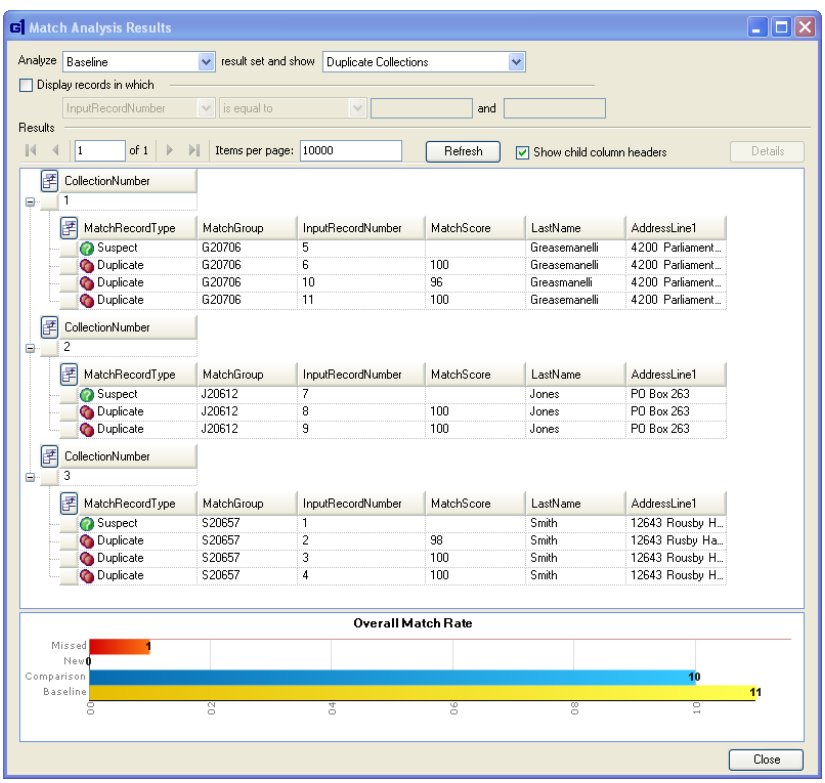
This chart shows the differences between the duplicate and unique records generated for the different match rules used.

6. Click the **Match Rules** tab. The match rules comparison displays.



From this tab you can see that the algorithm has been changed; Character Frequency is omitted and Exact Match has been added.

7. Click **Details**.
8. Select **Duplicate Collections** from the show list and then click **Refresh**.
9. Expand each **CollectionNumber** to view the Suspect and Duplicate records for each duplicate collection.



10. Compare the collections in the Detail view to the output file created.

Determining if a Prospect is a Customer

This sample service demonstrates how to evaluate prospect data in an input file to customer data in a customer database to determine if a prospect is a customer. In order to use this template, you will need to follow the steps for creating a database as detailed in **Candidate Finder** on page 23 and the Management Console documentation.

Template name: ProspectMatching.df

Sample input file name: N/A (this template is a service)

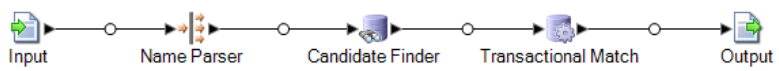
Modules required: Advanced Matching Module and Universal Name Module

Business Scenario

As a sales executive for an online sales company you want to determine if an online prospect is an existing customer or a new customer.

Solution

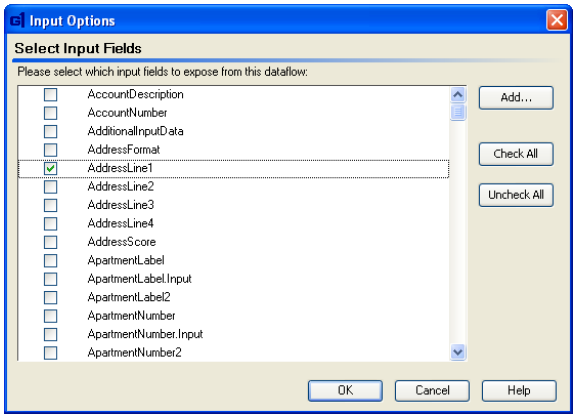
The following dataflow service provides a solution to the business scenario:



In this dataflow, data is read from a file and processed through the **Name Parser** on page 23, **Candidate Finder** on page 23, and **Transactional Match** on page 25 stages. For each data row in the input file, this data flow will do the following:

Input

The selected input fields for this template are AddressLine1, City, Name, PostalCode, and StateProvince. AddressLine1 and Name are the fields that are key to the dataflow processing in this template.



Name Parser

In this template, the Name Parser stage is named Parse Personal Name. Parse Personal Name stage examines name fields and compares them to name data stored in the Spectrum™ Technology Platform name database files. Based on the comparison, it parses the name data into First, Middle, and Last name fields, assigns an entity type, and a gender to each name. It also uses pattern recognition in addition to the name data.

In this template the Parse Personal Name stage is configured as follows.

- Parse personal names is selected and Parse business names is cleared. When you select these options, first names are evaluated for gender, order, and punctuation and no evaluation of business names is performed.
- Gender Determination Source is set to default. For most cases, Default is the best setting for gender determination because it covers a wide variety of names. However, if you are processing names from a specific culture, select that culture. Selecting a specific culture helps ensure that the proper gender is assigned to the names. For example, if you leave Default selected, then the name Jean will be identified as a female name. However, if you select French, it will be identified as a male name.
- Order is set to natural. The name fields are ordered by Title, First Name, Middle Name, Last Name, and Suffix.
- Retain periods is cleared. Any punctuation in the name data is not retained.

Candidate Finder

The Candidate Finder component is used in combination with the Transactional Match component.

The Candidate Finder component obtains the candidate records that will form the set of potential matches that the Transactional Match component will evaluate. In addition, depending on the format of your data, Candidate Finder may need to parse the name or address of the suspect record, the candidate records, or both.

As part of configuring Candidate Finder, you select the database connection through which the specified query will be executed. You can select any connection configured in Management Console. To connect to a database not listed, configure a connection to that database in Management Console, and then

close and reopen CandidateFinder to refresh the connection list. For more information on creating database connections, see the Management Console documentation.

To define the SQL query you can type any valid SQL select statement into the text box on the Candidate Finder Options view. For example, assume you have a table in your database called Customer_Table that has the following columns:

Customer_Table
Cust_Name
Cust_Address
Cust_City
Cust_State
Cust_Zip

Note: You can type any valid SQL select, however, `Select *` is not valid in this control.

To retrieve all the rows from the database, you might construct a query similar to the following:

```
select Cust_Name, Cust_Address, Cust_City, Cust_State, Cust_Zip from
Customer_Table;
```

However, it is unlikely that you would want to match your transaction against all the rows in the database.To return only relevant candidate records, you will want to add a WHERE clause using variable substitution.Variable substitution refers to a special notation that you will use to cause the Candidate Selection engine to replace the variable with the actual data from your suspect record.

To use variable substitution, enclose the field name in braces preceded by a dollar sign using the form **${FieldName}**. For example, the following query will return only those records that have a value in Cust_Zip that matches the value in PostalCode on the suspect record.

```
select Cust_Name, Cust_Address, Cust_City, Cust_State,Cust_Zip
from Customer_Table
where Cust_Zip = ${PostalCode};
```

Next you need to map database columns to stage fields if the column names in your database do not match the Component Field names exactly. If they do match they will be automatically mapped to the corresponding Stage Fields. You will need to use the Selected Fields (columns from the database) to map to the Stage Fields (field names defined in the dataflow).

Again consider the Customer_Table from the above example:

Customer_Table
Cust_Name
Cust_Address

Cust_City
Cust_State
Cust_Zip

When you retrieve these records from the database, you need to map the column names to the field names that will be used by the Transactional Match component and other components in your dataflow.For example, Cust_Address might be mapped to AddressLine1, and Cust_Zip would be mapped to PostalCode.

1. Select the drop-down list under **Selected Fields** in the candidate Finder Options view. Then, select the database column Cust_Zip.
2. Select the drop-down list under **Stage Fields**. Then, select the field to which you want to map.

For example, if you want to map Cust_Zip to Postal Code, first select Cust_Zip under Selected fields and then select PostalCode on the corresponding Stage Field row.

In addition to mapping fields as described above, you can use special notation in your SQL query to perform the mapping.To do this, you will enter the name of the Stage Field, enclosed in braces, after the column name in your query.When you do this, the selected fields will be automatically mapped to the corresponding stage fields.

An example of this using the query from the previous example follows:

```
select Cust_Name {Name}, Cust_Address {AddressLine1},
 Cust_City {City}, Cust_State {StateProvince},
 Cust_Zip {PostalCode}
from Customer
where Cust_Zip = ${PostalCode};
```

Transactional Match

The Transactional Match component is used in combination with the Candidate Finder component.

The Transactional Match component allows you to match suspect records against potential candidate records that are returned from the Candidate Finder Stage.

Transactional Match uses matching rules to compare the suspect record to all candidate records with the same candidate group number (assigned in Candidate Finder) to identify duplicates. If the candidate record is a duplicate, it is assigned a collection number, the match record type is labeled a Duplicate, and the record is then written out. Any unmatched candidates in the group are assigned a collection number of 0, labeled as Unique and then written out as well.

In this template, you create a custom matching rule that compares LastName and AddressLine1.

Here are some guidelines to follow when creating your matching hierarchy:

- A parent node must be given a unique name. It can not be a field.
- The child field must be a Spectrum™ Technology Platform data type field, that is, one available through one or more components.
- All children under a parent must use the same logical operators. To combine connectors you must first create intermediate parent nodes.

- Thresholds at the parent node could be higher than the threshold of the children.
- Parent nodes do not have to have a threshold.

Output

As a service, this template sends all available fields to the output. You can limit the output based on your needs.

3

Address Data Templates

In this section:

Parsing Addresses

This template illustrates how to break up a U.S. address into its individual components using a regular expression. The primary focus of this template is to show how you can use regular expressions in the Data Normalization Module's Advanced Transformer stage.

Note: This template is intended only to demonstrate how you can use Advanced Transformer to parse an address. It does not provide address parsing for all types of addresses.

Template name: ParsingAddresses

Sample input file name: N/A (this template is a service)

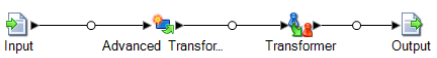
Modules required: Data Normalization Module

Business Scenario

You have a list of addresses that have had the spaces removed from the first address line. For example, the address "12 N MAIN ST" is in the form "12NMAINST." You need to correct the problem by inserting spaces in the appropriate places.

Solution

To solve this problem, you must first parse the address into its individual components, then reassemble the parts, adding spaces between them. The following dataflow provides a solution to the business scenario:



Note: You could also use the Universal Addressing Module to parse addresses. However, using the Data Normalization Module's Advanced Transformer stage allows you to control the parsing. This template is designed to show you how regular expressions can be used to parse addresses.

In this dataflow, the first address line of the address is sent to the Advanced Transformer. The Advanced Transformer uses a regular expression to parse the elements of the address into individual fields. These fields are then sent to the Transformer stage which concatenates the fields into a new AddressLine1 field and sends it to the output.

Each stage in this template is described below.

Input

The input is set up to accept only one field: AddressLine1. This is the first line of the address and typically contains a building number, street name, directionals (for example, N, S, E, and W) and a street suffix. Note that there are limitations to the types of address data that will work with this template. For more information, see the description of the Advanced Transformer stage below.

Advanced Transformer

This stage parses the address into the following fields:

- HouseNumber
- Predirectional
- StreetName
- Suffix

To accomplish this, the stage's parsing rules are defined using regular expressions. Regular expressions are strings that describe the format of a string that you want to parse. To understand this stage, a knowledge of regular expressions is required. There are several sources on the Internet with information on regular expressions.

There are two rules used for parsing: Extract from AddressLine1 using Regular Expression ([0-9]+)[ ]?((?i)(EAST| and Extract from StreetName using regular Expression (?i)=|AVE|ST|AVENUE|.

The first rule parses the input into HouseNumber, Predirectional, and StreetName fields. The stage recognizes the following predirectionals:

- EAST
- WEST
- NORTH
- SOUTH
- E
- W
- N
- S
- NW
- SW
- NE
- SW
- No predirectional

The StreetName field produced by the first rule is used by the second rule to identify the street suffix. The rule recognizes the following suffixes:

- AVE
- ST
- AVENUE
- STREET
- CT
- COURT
- RD
- ROAD
- LN
- LANE
- DR
- DRIVE

- No suffix

You can modify the regular expression to include additional suffixes.

Note that if you have an address whose street name happens to end with the same letters as a suffix, and there is no suffix, this stage will parse the last letters of the street into the suffix field. For example, the address 123ELOCUST would be parsed as follows:

- HouseNumber—123
- Predirectional—E
- StreetName—LOCU
- Suffix—ST

Transformer

This stage makes all the output fields uppercase for consistency. Since you can specify input addresses in any case, this field makes the output casing for all records the same.

Output

This stage specifies the fields to return as output from this service. The following fields are returned:

- HouseNumber
- Predirectional
- StreetName
- Suffix

Validating U.S. and Canadian Addresses

This template standardizes U.S. and Canadian addresses. U.S. addresses are standardized to conform to United States Postal Service (USPS) addressing standards as defined by the USPS Coding Accuracy Support System (CASS). Addresses that are standardized according to CASS regulations have a greatly improved deliverability rate and can qualify for postal discounts. Canadian addresses are standardized according to Canada Post standards.

Template name: ValidateUSAndCanadianAddresses

Sample input file name: CanadianAndUSAddresses.csv

Modules required: Universal Addressing Module

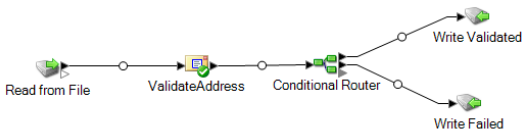
Business Scenario

You work for a financial services firm that sends critical customer communications, such as statements, privacy notices, and proxy notifications, to customers via the USPS. You want to take advantage of postal discounts, and as a first step you need to validate that the address for each customer is a deliverable address and that the address conforms to the conventions required by CASS.

The address data is in line sequential format. You need to read the data from this file. Validated addresses will be written to a line sequential file; addresses that cannot be validated will be written to a separate file for further research.

Solution

The following dataflow provides a solution to the business scenario:



In this dataflow, data is read from a file and processed through the ValidateAddress stage. (ValidateAddress is part of the Universal Addressing Module.) For each address, ValidateAddress will do one of the following:

- Verify that the address is correct.
- Modify the address so that it is correct (for example, adding the ZIP + 4 code if it is missing).
- If the address cannot be validated or modified to be correct, then the address is formatted to the correct layout and returned as a "failed" address.

A conditional router then sends the address to one of two locations, depending on whether or not ValidateAddress was able to return a valid address or not. If ValidateAddress was able to either validate the address or modify it to be a valid address, then the successfully validated address is routed to the Write Validated stage, which writes the address back to the source database, thus updating the source database with the standardized and validated address data. If ValidateAddress was not able to return a valid address, then the "failed" address is routed to the Write Failed stage and written to a file that contains all the failed address.

Each stage in this template is described in detail below.

Read from File

This stage identifies the file name, location, and layout of the file that contains the addresses you want to validate. The file contains both U.S. and Canadian addresses.

ValidateAddress

The ValidateAddress stage examines addresses and compares them to address data from the appropriate postal authority. In the case of U.S. addresses, ValidateAddress uses data from the USPS to validate the address. It corrects the address if possible so that it contains the correct information, and formats it according to the standards of the postal authority.

In this template the ValidateAddress stage is configured as follows.

On the **Output Data Options** tab:

- **Include matched address elements** is selected. When you select this option, each part of the address, such as house number, street name, street suffix, directionals, and so on, is returned in a separate field.
- **Return normalized data when no match is found** is selected. This will cause ValidateAddress to include failed addresses in the output and apply formatting standards to addresses that it cannot validate, resulting in addresses that are at least in the correct format even if they could not be validated. If you do not select this option, ValidateAddress will not return any address data for failed addresses. Since this template creates a flat file that contains failed addresses, this option must be selected.
- **Return street name alias** is selected. This option will use a street's alias instead of the "base" street name. A street alias is an alternate name for a street and typically applies only to a specific range of addresses on the street. If you do not allow street aliases in the output then the street's "base" name will appear in the output regardless of whether or not there is an alias for the street. The base name is the name that applies to the entire street.

On the **Default Options** tab, all the options are left at their default settings:

On the **U.S. Address Options** tab, the options are set to a CASS Certified configuration. (This is done by simply clicking the **Enable CASS** button, which sets all the options to their appropriate settings.)

Click the **Configure CASS 3553** button. The window that appears prompts you for information that is needed to complete the USPS CASS 3553 form. USPS Form 3553 summarizes a mailing and contains information about who the mailer is. This form must accompany a mailing when it is dropped off at a USPS facility. ValidateAddress always produces USPS Form 3553 when it runs in CASS CertifiedÂ™ mode. The information you enter in this window will appear on the USPS CASS 3553 form.

Since this template is intended to process an address list that contains only U.S. addresses, Canadian address processing and International address processing is disabled.

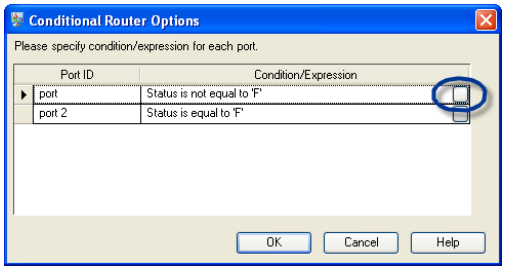
The **U.S. Barcode Options** tab contains settings for the Intelligent Mail Barcode. This is a barcode that can be printed on a mailpiece to facilitate mail tracking and delivery. Since you are not producing a mailing in this scenario, you do not need to modify any of the options on this tab.

Conditional Router

The output from the ValidateAddress stage will be either a valid, deliverable address or a "failed" address. To separate the validated addresses from the ones that couldn't be validated, the template contains a conditional router to route the address to the appropriate output file.

The Conditional Router looks at output fields from the preceding stage and routs records based on the values in the field. The output field name that is used in the template is called Status. The Status field contains an "F" if the address was not validated, and is null if it was validated. So, the Conditional Router stage will need to look at the Status field of each address returned from ValidateAddress and route those with a status of "F" to the flat file and those that do not have a status of "F" to the database.

To see how this is set up, double-click the Conditional Router stage then click the small button for the first port in the **Condition/Expression** column. The first port represents the condition that indicates a successful match and the second port represents failed records. You can view the condition setup for each port by clicking the small button the right of each port's setting.



The Expression Editor window appears. The **Field** field lists all the output fields from ValidateAddress.

Write to File

The template contains two Write to File stages: one writes validated addresses to a file, the other writes failed addresses to another file.

Validating International Addresses

This template uses the Address Now Module to validate addresses. You can validate U.S. and international addresses with the Address Now Module, but the module's strength is international addresses. Likewise, you can validate international addresses with the Universal Addressing Module, but if you have an extensive list of international addresses, consider using the Address Now module for enhanced international address coverage.

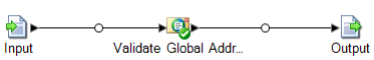
- **Template dataflow file name:** AddressNow_ValidateAddresses
- **Sample input file name:** N/A (this template is a service)
- **Modules required:** Address Now Module

Business Scenario

Your company has many customers in Europe and South America. You need to validate these addresses so that customer communications are delivered quickly and reliably. You want to provide real-time access to the address validation processing, so you need to set up a Spectrum™ Technology Platform service to perform the address validation.

Solution

The following dataflow provides a solution to the business scenario:



In this dataflow, data is entered through a call to the service and processed through the Validate Global Address stage. (Validate Global Address is part of the Address Now Module.) For each address, Validate Global Address will do one of the following:

- Verify that the address is correct.
- Modify the address so that it is correct (for example, adding the postal code if it is missing).
- If the address cannot be validated or modified to be correct, then the address elements are formatted using the appropriate postal standards and returned as a "failed" address.

Each stage in this template dataflow is described in detail below.

Input

This stage defines the input fields for the service. The input can be up to eight address lines, city, state/province, postal code, firm name, and country.

Validate Global Address

Validate Global Address provides enhanced address standardization and validation for addresses outside the U.S. and Canada. Validate Global Address can also validate addresses in the U.S. and Canada but its strength is validation of addresses in other countries. The Validate Global Address stage examines addresses and compares them to address data from the appropriate postal authority. It corrects the address if possible so that it contains the correct information, and formats it according to the standards of the postal authority.

In this template the Validate Global Address options are set to their default values, with the exception of **Return standardized data when no match is found**, which is enabled. This will cause Validate Global Address to include failed addresses in the output. If you do not select this option, Validate Global Address will not some address elements for failed addresses.

Output

The Output stage defines which fields the service should return. You can use the output stage to limit the output to only those fields that are useful to your particular situation. In the template, all the fields are enabled.

Validating International Addresses with Candidates

This template uses the Address Now Module to validate addresses and return candidate addresses when there are multiple possible matches for a given input address. You can validate U.S. and international addresses with the Address Now Module, but the module's strength is international addresses. Likewise, you can validate international addresses with the Universal Addressing Module, but if you have an extensive list of international addresses, consider using the Address Now module for enhanced international address coverage.

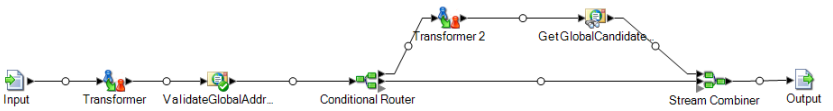
- **Template dataflow file name:** AddressNow_ValidateAddressWithCandidates
- **Sample input file name:** N/A (this template is a service)
- **Modules required:** Address Now Module

Business Scenario

Your company has many customers in South America and Australia. You need to allow business users to validate addresses as they are entered into your system. If the business user enters an address that cannot be positively matched to a single address, you want to provide a list of candidate addresses for the user to choose from. Since you want to provide real-time access to the address validation processing, you need to set up a Spectrum™ Technology Platform service to perform the address validation.

Solution

The following dataflow provides a solution to the business scenario:



In this dataflow, data is entered through a call to the service and processed through the Transformer stage which creates passthrough fields necessary to preserve the original address data as it moves through the dataflow. The address then moves to the Validate Global Address stage. This stage, which is part of the Address Now Module, will do one of the following:

- Verify that the address is correct.
- Modify the address so that it is correct (for example, adding the postal code if it is missing).
- If the address cannot be validated or modified to be correct, then the address elements are formatted using the appropriate postal standards and returned as a "failed" address.

The Conditional Router sends addresses that failed to the Get Global Candidate Addresses stage. This stage returns a list of candidate addresses which is sent to the Stream Combiner and returned to the user.

Each stage in this template dataflow is described in detail below.

Input

This stage defines the input fields for the service. The input can be up to eight address lines, city, state/province, postal code, firm name, and country.

Transformer

This stage copies the input address fields to new passthrough fields. This step is necessary to preserve the original input address which will be needed to look up candidate addresses in Get Global Candidate Addresses. This passthrough data is only needed if Validate Global Address cannot validate the address because multiple candidates are identified.

Validate Global Address

Validate Global Address provides enhanced address standardization and validation for addresses outside the U.S. and Canada. Validate Global Address can also validate addresses in the U.S. and Canada but

its strength is validation of addresses in other countries. The Validate Global Address stage examines addresses and compares them to address data from the appropriate postal authority. It corrects the address if possible so that it contains the correct information, and formats it according to the standards of the postal authority.

In this template the Validate Global Address options are set to their default values, with the exception of **Return standardized data when no match is found**, which is enabled. This will cause Validate Global Address to include failed addresses in the output. If you do not select this option, Validate Global Address will not return some address elements for failed addresses.

Conditional Router

This stage identifies addresses that failed or that do not have a confidence of 100 and routes them toward the Get Global Candidate Addresses stage. Addresses that were positively validated are returned to the user via the Output stage.

Transformer 2

This stage reverts the address back to the original input address using the passthrough fields to overwrite the output from Validate Global Address. This step is necessary because Get Global Candidate Addresses needs the original input address to perform the lookup.

Get Global Candidate Addresses

This stage returns a list of candidate addresses for the addresses that Validate Global Address was not able to validate.

Stream Combiner

This stage allows both validated address and candidate lists to be routed to the output.

Output

The Output stage defines which fields the service should return. You can use the output stage to limit the output to only those fields that are useful to your particular situation. In the template, all the fields are enabled except the passthrough fields, which are used internally within the dataflow but are not useful to the business user.

4

Location Data Templates

In this section:

Geocoding U.S. Addresses

"Geocoding" is the process of determining the geographical coordinates (latitude/longitude) for a location. Geocoding can be the first step to a more detailed spatial analysis of a given address. Once you know the coordinates for an address, you can match it against a variety of geographic data to learn about the location. For example, you can determine what school district the address resides in, whether or not the address is in a hurricane-prone area, demographic information, and many other characteristics of a location.

This template illustrates one way of using Spectrum™ Technology Platform for geocoding.

Template name: GeocodeAddress

Sample input file name: USAddresses.csv

Modules required: Enterprise Geocoding Module

Business Scenario

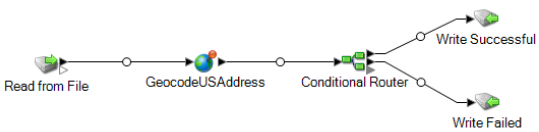
You work for a credit card company that has partnered with certain retailers to provide targeted advertising on cardholders' monthly statements (both printed and online statements). Part of the targeted message will include the address of the retailer's closest store.

Your database of cardholder data includes the cardholder's address. The address data has already been standardized and validated, so you are confident in the quality of the address data. While this address data is useful for mail delivery, it is not useful for determining a retailer's closest store. For that, you need to know the coordinates of the address.

In this example, you will use Spectrum™ Technology Platform to enhance your database of cardholder data to include the latitude/longitude coordinates of the address. This data will then be used by the billing application to include targeted advertisements in cardholders' statements, including the location of the closest store.

Solution

The following dataflow provides a solution to the business scenario:



This dataflow reads a file that contains the addresses you want to geocode. The addresses are U.S. addresses only. The Enterprise Geocoding Module's GeocodeUSAddress component determines the address' latitude and longitude and writes the successfully geocoded addresses to one file, and the failed addresses to another file.

Each of the stages is described below.

Read from File

The first stage reads the input data from a file. It identifies the file layout and location.

GeocodeUSAddress

GeocodeUSAddress takes an address and assigns the latitude and longitude to the address.

Note: To use GeocodeUSAddress, you must specify a database in the **Database** field. If you do not specify a database the dataflow will not run successfully. You set up databases using the Database Resources tool in Management Console and give the database a name of your choosing. This is the name you select in the **Database** field. For more information, see the *Spectrum™ Technology Platform User's Guide*.

While GeocodeUSAddress can validate addresses, this template is set up on the assumption that the incoming address data has already been validated, for example through the Universal Addressing Module's ValidateAddress component. So, several address validation options disabled.

The other option to note is **Match mode**. This option controls how closely an input address must match a valid address in the postal data used to validate addresses. In this template, the match mode is set to Close since we are fairly confident in the quality of the address data. If we were less confident, we would use the default setting of Relax.

The other geocoding options are left at their default values.

On the **Output Format** tab, all options are left at their default values.

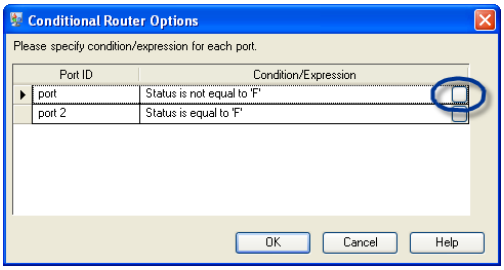
On the **Output Data** tab, the **Latitude/Longitude** option is selected so that we get the latitude/longitude in the output. By default, we will also get the address and geocode in the output. We do not specify any extra output fields because the selected options will return all the fields we need to determine the latitude/longitude location of the address.

Conditional Router

The output from the GeocodeUSAddress stage will be either a valid address with latitude/longitude assigned, or a "failed" address for which GeocodeUSAddress could not determine a latitude/longitude. To separate the geocoded addresses from the ones that couldn't be geocoded, the template contains a conditional router to route the address to the appropriate output file.

The Conditional Router looks at output fields from the preceding stage and routes records based on the values in the field. The output field name that is used in the template is called Status. The Status field contains an "F" if the address was not validated, and is null if it was validated. So, the Conditional Router stage will need to look at the Status field of each address returned from ValidateAddress and route those with status of "F'" to the flat file and those with status that is not "F'" to the database.

To see how this is set up, double-click the Conditional Router stage then click the small button for the first port in the **Condition/Expression** column. The first port represents the condition that indicates a successful match and the second port represents failed records. You can view the condition setup for each port by clicking the small button the right of each port's setting.



The Expression Editor window appears. The **Field** field lists all the output fields from GeocodeUSAddress.

Write Successful

This stage writes the addresses and their newly-assigned latitude/longitude and geocode to a file.

Write Failed

This stage writes the addresses that could not be geocoded to a separate file.

Determining Insurance Rating Territory

Insurance companies are expected to accurately assign the right territory to every policy in order to calculate the correct premium and avoid market-conduct fines. Territories can be geographically complex and quite often do not follow ZIP Code boundaries. This frequently leads to territory assignment errors. Assignment errors mean that incorrect premiums are being quoted or charged, which creates both customer retention problems (when overcharging occurs), and premium revenue loss (when undercharging occurs).

This template illustrates one way of using Spectrum™ Technology Platform to improve the accuracy of rating territory assignment.

Template name: DetermineInsuranceRatingTerritory

Sample input file name: LatLong.csv

Modules required: Location Intelligence Module

Business Scenario

You work for an insurance company. A recent audit has revealed that rating territories for homeowner policies are being incorrectly assigned at an unacceptably high rate. Your company has undertaken an initiative to analyze existing policies and correct the territory assignments where necessary.

In order to determine the insurance rating territory, you need to have a valid address and the latitude/longitude coordinates of that address. You have an existing process in place that corrects and validates addresses and determines the latitude and longitude. You need to take the latitude/longitude and compare it against your insurance rating territories.

Solution

The following dataflow provides a solution to the business scenario:

This dataflow reads file that contains addresses and policy numbers. The Location Intelligence Module's Point In Polygon stage determines which insurance rating territory each location resides in. Point In Polygon identifies the insurance rating territory based on a user-defined boundary file that contains the locations of your company's territories. Finally, the address, latitude/longitude, policy number, and rating territory information are written to a file.

Each of the stages is described below.

Read from File

The first stage reads the file that contains the addresses you want to match to insurance rating territories. In addition, the input file contains the policy number associated with each address. This policy number is assigned to the InputKeyValue field, which is for user-defined data.

Point In Polygon

The Point In Polygon stage is where the insurance rating territory is assigned. In order for Point In Polygon to identify the insurance rating territory, you must supply a user-defined boundary file. A user-defined boundary file is a file that describes the location of your territories. This file must be in a specific format. Once you have created this file, you must define it as a database on your system using the Management Console. For additional information, see the *Spectrum™ Technology Platform User's Guide*.

Once you have created and defined your boundary file of territories, you select the file in the **Database** field. This indicates which areas you want Point In Polygon to match against.

Note: If you do not select a database in the **Database** field the dataflow will not run successfully.

Since the latitude/longitude format from GeocodeUSAddress is in decimal format, the **Coordinate Format** field is set to Decimal.

Write Territory Assignment

This stage defines the location and layout of the addresses with their geocode, policy number (the InputKeyValue field), and insurance rating territory information. Note that you will have to specify the specific fields to include based on your territory data file.

Determining Tax Jurisdiction

Identifying which tax jurisdictions apply to a given location can be a time-consuming and complex process. A location may have multiple tax jurisdictions, including not only federal, state, and municipality jurisdictions, but also special tax districts, property tax districts, and others.

This template illustrates one way of using Spectrum™ Technology Platform to determine tax jurisdictions for an address.

Template name: DetermineTaxJurisdictions

Sample input file name: USAddresses.csv

Modules required: Universal Addressing Module, Enterprise Tax Module

Business Scenario

You work for a telecommunications company. Your company needs to ensure that it is including the correct taxes on customers' bills. This is a very complex problem because there can be multiple tax jurisdictions that apply to a customer's location. For example, there may be a federal, state, city and special district tax on the services you provide.

Your database of customer data contains each customer's address. However, the address data is of questionable quality, so you want to first standardize and validate the address data, since a quality address is the first step in determining tax jurisdictions.

Your company uses tax software from Taxware® to determine tax rates and you need to interface with the Taxware system to identify rates.

In this example, you will use Spectrum™ Technology Platform to first standardize and validate your address data. Then, you will use the validated address data to determine the applicable tax jurisdictions for the customer. The tax jurisdiction will be represented by a code that you can provide to your Taxware system to determine the tax rates that apply.

Note: This template is set up assuming that you are using Taxware software to determine the tax rate based on the jurisdiction returned by Spectrum™ Technology Platform. Assign GeoTAX Info can also return the codes necessary to determine tax rates using Vertex® software. Alternatively, you could use your own company's custom tables to determine tax rates. If you take this approach, modify this template to return the following fields: StateCode County.Code GNISCode SPDn.DistrictCode To enable these fields, check the **Census** check box on the Assign GeoTAX Info **Output Data** tab and also select `Special Purpose Districts` in the **Tax district** field on the same tab. For more information on these fields, see the *Spectrum™ Technology Platform User's Guide*.

Solution

The following dataflow provides a solution to the business scenario:

In this dataflow, the address data is read from a file. The address data is validated through the Universal Addressing Module's ValidateAddress component. This validation process is critical to ensure accurate tax jurisdiction assignments. After the address is validated, it is sent to the Enterprise Tax Module's Assign GeoTAX Info component, where the tax jurisdictions are assigned. Finally, the address, along with the appropriate tax district information, is sent to the output file.

Each of these stages is described below.

Read from File

The input stage defines the file and fields that the job will take as input. Along with address data, the input file contains a field called InputKeyValue, which can contain any value you choose, such as an account number.

ValidateAddress

In this template the ValidateAddress stage is configured as follows.

On the **Output Data Options** tab:

- **Include a standard address** is selected. When you select this option, ValidateAddress returns 1 to 4 lines of address data plus city, state, postal code, firm name, and urbanization name information. Each address line represents an actual line of the address as it would appear on an envelope.If ValidateAddress could validate the address, the address lines contain the standardized address. When addresses are standardized, punctuation is removed, directionals are abbreviated, street suffixes are abbreviated, and address elements are corrected. If ValidateAddress could not validate the address, the address lines contain the address as it appeared in the input ("pass through" data). Non-validated addresses are always included as pass through data in the address line fields even if you uncheck this option.
- **Return street name alias** is selected. This option will use a street's alias instead of the "base" street name. A street alias is an alternate name for a street and typically applies only to a specific range of addresses on the street. If you do not allow street aliases in the output then the street's "base" name will appear in the output regardless of whether or not there is an alias for the street. The base name is the name that applies to the entire street.
- **Return normalized data when no match is found** is selected. This option returns a formatted address when an address cannot be validated. If this option is not selected, the output address fields are blank when ValidateAddress cannot validate the address.

On the **Default Options** tab, the **Dual address logic** is set to `Street Match`. This setting will ensure that addresses that contain both street and PO Box/Rural Route/Highway Contract information. If this option is not selected, it is possible that ValidateAddress could return an address that consists only of a PO Box/Rural Route/Highway Contract number and city, state, and ZIP Code. This type of address, while appropriate for mail delivery, is not the optimal form to use when determining tax jurisdictions.

All other options are left at their default settings:

On the **U.S. Address Options** tab, U.S. address processing is enabled but none of the processing options are enabled. This provides basic address validation and the quickest performance.

Since this template is intended to process an address list that contains only U.S. addresses, Canadian address processing and International address processing is disabled.

The **U.S. Barcode Options** tab contains settings for the Intelligent Mail Barcode. This is a barcode that can be printed on a mailpiece to facilitate mail tracking and delivery. Since you are not producing a mailing in this scenario, all of the barcoding options are disabled.

Assign GeoTAX Info

Assign GeoTAX Info takes the validated addresses and determines which tax jurisdictions the address resides it. This template uses the default options, with the exception of some options on the **Output Data** tab. On this tab, **Tax Jurisdiction** is selected in the **Include Data** list. This returns the **GeoTAXKey** field in the output. This is the key that contains the Taxware code that you will use to look up the actual tax rate in your Taxware system. The related option **GeoTAX key** is set to Taxware. This indicates that Assign GeoTAX Info will use the Taxware key; if you were using Vertex® software to determine tax rates, you would specify Vertex in this field.

Output

The output stage returns the validated address as well as the GeoTAXKey field, which contains the tax jurisdiction code to use with Taxware software to determine the tax rate.

Determining Flood Risk

Spectrum™ Technology Platform comes with two templates that illustrate how to determine the relationship of an address to known flood zones. The two templates are FloodRiskAnalysis and FloodRiskDetailAnalysis. Both these templates take an address or intersection and determine how prone the location is to flooding. The difference is in the method of comparison. FloodRiskAnalysis locates all the nearby flood zones then performs a union operation on those flood zones, which creates a new geometry that encompasses the flood zones. The address is then compared to this single geometry and an overlap percentage is returned. FloodRiskDetailAnalysis, however, compares the address to all nearby flood zones individually, returning separate overlap percentages for each nearby flood zone and allowing for a more detailed analysis of risk.

Both FloodRiskAnalysis and FloodRiskDetailAnalysis use two other dataflows, which are also provided as dataflow templates:

- **GeoConfidenceSurface**—This template, which is exposed as a service, creates the geoconfidence surface that can be used for further analysis. The geocofidence surface is a buffered point or polygon that represents the area in the immediate vicinity of the address or intersection. The size of the shape is configurable. This template uses a buffer distance of 1 mile.
- **CreatePointsConvexHull**—This is a subflow that is used by the GeoConfidenceSurface template. You should not need to make any changes to this subflow.

Template names: FloodRiskAnalysis, FloodRiskDetailAnalysis

Sample input file name: None (template is a service dataflow)

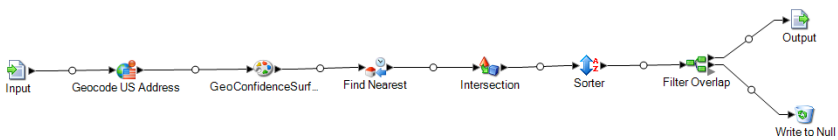
Modules required: GeoConfidence Module, Enterprise Geocoding Module, Location Intelligence Module

Business Scenario

You work for an insurance company as an underwriter for homeowner policies. As part of the underwriting process you need to know how likely it is that a home will experience flooding, which means you need to determine how close the home is to a flood zone. You know the address of home and from that you need to determine the flood risk.

Solution: FloodRiskDetailAnalysis

FloodRiskDetailAnalysis compares the location to each nearby flood zone individually, returning separate overlap percentages for each nearby flood zone. The following dataflow provides a solution to the business scenario:



In this dataflow, the input address is first geocoded, meaning the latitude/longitude coordinates of the address are determined. Then the location is converted to a geoconfidence shape, which represents the area around the address. Next the nearest flood zones are located. Then the location's shape is compared to each flood zone shape and the percentage overlap is determined. Finally any records with a percentage of less than one are discarded, leaving the more significant results to be returned. Using these percentages you could then determine the degree of flood risk associated with the location.

Each stage is described in detail below.

Input

The input to this dataflow is an address. Since this dataflow is a service, the first stage is an input stage. Note that there is an ID field which is used to preserve the order of the records in the output. The records are sorted in ascending order based on the value in the ID field. So for example, the first record could have an ID of 1, the second and ID of 2, and so forth.

Geocode US Address

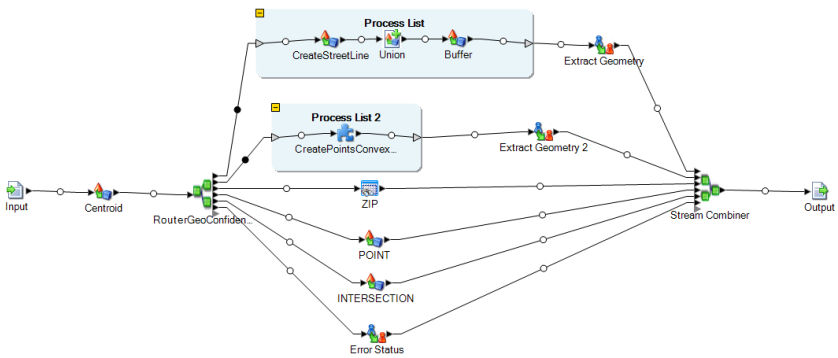
This stage determines the latitude/longitude coordinates of the address and returns the information needed to construct a geoconfidence surface. A geoconfidence surface is a buffered point, line, or polygon, that encompasses the immediate area around the location. (Note that the **Geo Confidence** check box on the **Output Data** tab must be checked in order for Geocode US Address to return this information.) The type of shape returned as the geoconfidence surface depends on the precision of the match:

- **Point**—If the geocoder was able to match the address using point data, then Geocode US Address returns a point. A point is also returned if an intersection is entered instead of an address.

- **Street segment**—If the geocoder matched the address to a street segment, then Geocode US Address returns an array of points representing the street segment. This array of points is converted to a line later in the dataflow.
- **ZIP + 4**—If the geocoder matched the address to a ZIP + 4 area, then Geocode US Address returns an array of points representing all the street segments in the ZIP + 4. This array of points is converted to a polygon later in the dataflow.
- **ZIP + 2**—If the geocoder matched the address to a ZIP + 2, then Geocode US Address returns an array of points representing the centroids of the ZIP + 4 areas contained in the ZIP + 2. (Note that a ZIP + 2 area is made up of multiple ZIP + 4 areas.) This array of points is converted to a polygon later in the dataflow.
- **5-digit ZIP Code centroid**—If the geocoder matched the address to a 5-digit ZIP Code, then the centroid (center point) of the area is returned. The ZIP Code polygon is determined later in the dataflow.

GeoConfidenceSurface

The GeoConfidenceSurface stage creates a shape that represents the area immediately surrounding the address or intersection. The GeoConfidenceSurface stage is a service. To view it, right-click the stage and select **Edit This Dataflow**.



The first stage in this dataflow after input is **Centroid**. This stage converts the point represented by the GeoConfidenceLatitude and GeoConfidenceLongitude values to a point geometry. This is necessary because the operations performed later in the subflow require the point to be represented as a geometry datatype as opposed latitude/longitude strings.

The **RouterGeoConfidence** stage then directs each record to the appropriate path based on the type geoconfidence shape determined by Geocode US Address:

- **Street segments and ZIP + 4**—Records geocoded to a street segment or ZIP + 4 level (those with a value of either ADDRESS or POSTAL3 in the GeoConfidenceCode field) are routed to the first port. This path takes them to a series of stages. The first, **CreateStreetLine**, converts the array of points to a line. The lines are then passed to the **Union** stage which combines the points into a single line geometry. The **Buffer** stage then creates a buffer of one mile around the streets. Because the street segment or ZIP + 4 street network is an array of points, these three stages are wrapped in a Process List area, which tells the stages to iterate through each set of points in the array. Finally, the **Extract Geometry** stage converts the data type to a geometry.

- **ZIP + 2**—Records geocoded to a ZIP + 2 level (those with a value of POSTAL2 in the GeoConfidenceCode field) are routed to the second port. This path sends the records to another subflow, **CreatePointsConvexHull**. In this subflow, the **CreatePoint** stage converts the array of latitude/longitude points to point geometries. Then the **Union** stage combines all the points into a single geometry. The **ConvexHull** stage then performs a convex hull operation, which is like stretching a rubber band around the shapes to produce a new shape. Click the **Help** button in this stage for an illustration. Finally, the **Extract Geometry** stage converts the data type to a geometry.
- **5-digit ZIP Code**—Records geocoded to a ZIP Code centroid (records with a value of POSTAL1 in the GeoConfidenceCode field) are routed to the third port. This path takes the record to the **ZIP** stage, which looks up the ZIP code's shape in the 5-digit ZIP spatial database. This database is included as part of the GeoConfidence Module.

Note: To use this dataflow you must first specify a database in the ZIP stage. To do this, install the database, then define a database resource for the database in the Management Console. For instructions, see the *Spectrum™ Technology Platform User's Guide*. Then select the database resource in the **Spatial database** field.

- **Point**—Records that were geocoded using point data (those with a value of POINT int he GeoConfidenceCode field) are routed to the fourth port. In this path, the record goes to the **POINT** stage which creates a buffer of 1 mile around the point.
- **Intersection**—Records that were geocoded to intersections (those with a value of (INTERSECTION) in the GeoConfidenceCode field) are routed to the fifth port. In this path the record goes to the **INTERSECTION** stage which creates a buffer of 1 mile around the point.

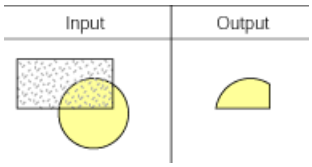
Find Nearest

The Find Nearest stage locates the closest flood plains to the geoconfidence surface. Before running this dataflow you must specify a spatial database containing the flood plain data. To do this, first define the database resource in Management Console, then select the database resource in the **Spatial database** field. For instructions on defining a database resource, see the *Spectrum™ Technology Platform User's Guide*.

The Find Nearest stage is configured to search up to 1 mile for nearby flood plains, and will return up to 50 flood plains. (You can reduce this if you want to limit the number of nearby flood plains returned.)

Intersection

The intersection stage compares the geoconfidence surface with the flood planes and determines the percentage of the geoconfidence surface that overlaps with each flood plain. For example if the geoconfidence surface is the yellow circle and the flood plain is the rectangle, the overlapping area would be:



The percentage returned would be the percentage of the circle represented by the shape shown under "Output".

Filter Overlap

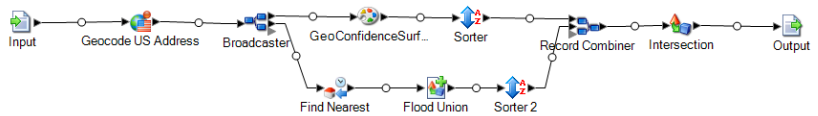
The Filter Overlap stage removes records where there is an overlap of less than 1%. This can be modified as needed to filter out those intersections that are not significant enough to warrant your attention.

Output

The Output stage returns a variety of information, including the address, percentage of the geoconfidence surface that overlaps with each flood plain, the number of nearby flood plains, the latitude/longitude of the address, and more.

Alternative Solution: FloodRiskAnalysis

FloodRiskAnalysis differs from FloodRiskDetailAnalysis in that a single overlap percentage is determined by merging all nearby flood zones into a single geometry using a union operation, then comparing the location to that single geometry.



This dataflow is similar to FloodRiskDetailAnalysis but has some important differences:

- There are two paths in this dataflow, one for determining the geoconfidence shape and one creating a shape that encompasses all nearby flood plains.
- There is a **Flood Union** stage that combines all nearby flood zones into a single geometry by performing a spatial union operation. This is necessary because the dataflow later performs an intersection operation comparing the address or intersection's geoconfidence surface with a geometry representing all the nearby flood plains.
- There is a **Record Combiner** takes the geoconfidence surface and the polygon representing the nearby flood plains and combines them so that each record now contains the address, a buffered point or polygon representing the area around the address, and a polygon representing the nearby flood plains.

5

Open Parser

In this section:

Parsing Arabic Names

This template demonstrates how to parse westernized Arabic names into component parts. The parsing rule separates each token in the **Name** field and copies each token to five fields: **Kunya**, **Ism**, **Laqab**, **Nasab**, **Nisba**. These output fields represent the five parts of an Arabic name and are described in the business scenario.

Template name: ParseArabicNames.df

Sample input file name: ArabicNames.csv

Sample output file name: ParseArabicNames-Output.csv

Modules required: Data Normalization Module

Business Scenario

You work for a bank that wants to better understand the Arabic naming system in an effort to improve customer service with Arabic-speaking customers. You have had complaints from customers whose billing information does not list the customer's name accurately. In an effort to improve customer intimacy, the Marketing group you work in wants to better address Arabic-speaking customers through marketing campaigns and telephone support.

In order to understand the Arabic naming system, you search for and find these resources on the internet that explain the Arabic naming system:

- **en.wikipedia.org/wiki/Arabic_names**
- **heraldry.sca.org/laurel/names/arabic-naming2.htm**

Arabic names are based on a naming system that includes these name parts: Ism, Kunya, Nasab, Laqab, and Nisba.

- The ism is the main name, or personal name, of an Arab person.
- Often, a kunya referring to the person's first-born son is used as a substitute for the ism.
- The nasab is a patronymic or series of patronymics. It indicates the person's heritage by the word ibn or bin, which means son, and bint, which means daughter.
- The laqab is intended as a description of the person. For example, al-Rashid means the righteous or the rightly-guided and al-Jamil means beautiful.
- The nisba describes a person's occupation, geographic home area, or descent (tribe, family, and so on). It will follow a family through several generations. The nisba, among the components of the Arabic name, perhaps most closely resembles the Western surname. For example, al-Filistin means the Palestinian.

Solution

The following dataflow provides a solution to the business scenario:

In this dataflow, data is read from a file and processed through the Open Parser stage. For each data row in the input file, this dataflow will do the following:

Read from File

This stage identifies the file name, location, and layout of the file that contains the names you want to parse. The file contains both male and female names.

Open Parser

This stage defines whether to use a culture-specific domain grammar created in the Domain Editor or to define a domain-independent grammar. A culture-specific parsing grammar that you create in the Domain Editor is a validated parsing grammar that is associated with a culture and a domain. A domain-independent parsing grammar that you create in Open Parser is a validated parsing grammar that is not associated with a culture and domain. For more information about the two types of parsing grammars, see the Spectrum™ Technology Platform User's Guide.

In this template, the parsing grammar is defined as a domain-independent grammar.

The Open Parser stage contains a parsing grammar that defines the following commands and expressions:

- `%Tokenize` is set to the space character (`\s`). This means that Open Parser will use the space character to separate the input field into tokens. For example, Abu Mohammed al-Rahim ibn Salamah contains five tokens: Abu, Mohammed, al-Rahim, ibn and Salamah.
- `%InputField` is set to parse input data from the **Name** field.
- `%OutputFields` is set to copy parsed data into five fields: **Kunya**, **Ism**, **Laqab**, **Nasab**, and **Nisba**.
- The `<root>` expression defines the pattern for Arabic names:
- Zero or one occurrence of **Kunya**
- Exactly one or two occurrences of **Ism**
- Zero or one occurrence of **Laqab**
- Zero or one occurrence of **Nasab**
- Zero or more occurrences of **Nisba**

The rule variables that define the domain must use the same names as the output fields defined in the required `OutputFields` command.

The parsing grammar uses a combination of regular expressions and expression quantifiers to build a pattern for Arabic names. The parsing grammar uses these special characters:

- The "?" character means that a regular expression can occur zero or one time.
- The "*" character means that a regular expression can occur zero or more times
- The ";" character means end of a rule.

Use the **Commands** tab to explore the meaning of the other special symbols you can use in parsing grammars by hovering the mouse over the description.

By default, quantifiers are greedy. Greedy means that the expression accepts as many tokens as possible, while still permitting a successful match. You can override this behavior by appending a '?' for reluctant matching or '+' for possessive matching. Reluctant matching means that the expression accepts as few tokens as possible, while still permitting a successful match. Possessive matching means that the expression accepts as many tokens as possible, even if doing so prevents a match.

To test the parsing grammar, click the Preview tab. Type the names shown below in the **Name** field and then click **Preview**.

Name	Kunya	Ism	Laqab	Nasab	Nisba
Abu Karim Muhammad al-Jamil ibn Nidal ibn Abdulaziz al-Filistini	Abu Karim	Muhammad	al-Jamil	ibn Nidal ibn Abdulaziz	al-Filistini
Layla bint Zuhayr ibn Yazid al-Nahdiyah		Layla		bint Zuhayr ibn Yazid	al-Nahdiyah
Yazid ibn Abi Hakim		Yazid		ibn Abi Hakim	
Abu Bishr al-Yaman ibn Abi al-Yaman al-Bandaniji	Abu Bishr	al-Yaman		ibn Abi	al-Yaman al-Bandaniji
Abu al-Tayyib ʿAbd al-Rahim ibn Ahmad al-Harrani	Abu al-Tayyib	ʿAbd	al-Rahim	ibn Ahmad	al-Harrani
Ahmad ibn Saʿid al-Bahili		Ahmad		ibn Saʿid	al-Bahili
Abu al-ʿAbbas Muhammad ibn Yaʿqub ibn Yusuf al-Asamm al-Naysaburi	Abu al-ʿAbbas	Muhammad		ibn Yaʿqub ibn Yusuf	al-Asamm al-Naysaburi
Abu al-Qasim Mansur ibn al-Zabriqan ibn Salamah al-Namari	Abu al-Qasim	Mansur		ibn al-Zabriqan ibn Salamah	al-Namari
ʿUbayd ibn Muʿawiyah ibn Zayd ibn Thabit ibn al-Dahhak		ʿUbayd		ibn Muʿawiyah ibn Zayd ibn Thabit ibn al-Dahhak	
Umm Jaʿfar Zubaydah	Umm Jaʿfar	Zubaydah			

You can also type other valid and invalid names to see how the input data is parsed.

You can use the Trace feature to see a graphical representation of either the final parsing results or to step through the parsing events. Click the link in the **Trace** column to see the Trace Details for the data row.

Write to File

The template contains one Write to File stage. In addition to the input field, the output file contains the **Kunya**, **Ism**, **Laqab**, **Nasab**, **and Nisba** fields.

For complete information about Open Parser, the Domain Editor, and the parsing grammar reference, see Open Parser in the Data Normalization Module help or in the Spectrum™ Technology Platform User's Guide.

Parsing Chinese Names

This template demonstrates how to parse Chinese names into component parts. The parsing rule separates each token in the **Name** field and copies each token to two fields: **LastName** and **FirstName**.

Template name: ParseChineseNames.df

Sample input file name: ChineseNames.csv

Sample output file name: ParseChineseNames-Output.csv

Modules required: Data Normalization Module

Business Scenario

You work for a financial service company that wants to explore if it is feasible to include the Chinese characters for its Chinese-speaking customers on various correspondence.

In order to understand the Chinese naming system, you search for and find this resource on the internet, which explains how Chinese names are formed:

en.wikipedia.org/wiki/Chinese_names

Solution

The following dataflow provides a solution to the business scenario:

In this dataflow, data is read from a file and processed through the Open Parser stage. For each data row in the input file, this data flow will do the following:

Read from File

This stage identifies the file name, location, and layout of the file that contains the names you want to parse. The file contains both male and female names.

Open Parser

This stage defines whether to use a culture-specific domain grammar created in the Domain Editor or to define a domain-independent grammar. A culture-specific parsing grammar that you create in the Domain Editor is a validated parsing grammar that is associated with a culture and a domain. A domain-independent parsing grammar that you create in Open Parser is a validated parsing grammar that is not associated with a culture and domain. For more information about the two types of parsing grammars, see the Spectrum™ Technology Platform User's Guide.

In this template, the parsing grammar is defined as a domain-independent grammar.

The Open Parser stage contains a parsing grammar that defines the following commands and expressions:

- `%Tokenize` is set to None. When `Tokenize` is set to `None`, the parsing grammar rule must include any spaces or other token separators within its rule definition.
- `%InputField` is set to parse input data from the **Name** field.
- `%OutputFields` is set to copy parsed data into two fields: **LastName** and **FirstName**.

The `<root>` expression defines the pattern for Chinese names:

- One occurrence of **LastName**
- One to three occurrences of **FirstName**

The rule variables that define the domain must use the same names as the output fields defined in the required `OutputFields` command.

The `CJKCharacter` rule variable defines the character pattern for Chinese/ Japanese/Korean (CJK). The character pattern is defined so as to only use characters that are letters.The rule is:

```
<CJKCharacter> = @RegEx("([\p{InCJKUnifiedIdeographs}&&\p{L}])");
```

- The regular expression `\p{InX}` is used to indicate a Unicode block for a certain culture, in which `X` is the culture. In this instance the culture is CJKUnifiedIdeographs.
- In regular expressions, a character class is a set of characters that you want to match. For example, [aeiou] is the character class containing only vowels. Character classes may appear within other character classes, and may be composed by the union operator (implicit) and the intersection operator (&&). The union operator denotes a class that contains every character that is in at least one of its operand classes. The intersection operator denotes a class that contains every character that overlaps the intersected Unicode blocks.
- The regular expression `\p{L}` is used to indicate the Unicode block that includes only letters.

To test the parsing grammar, click the Preview tab. Type the names shown below in the **Name** field and then click **Preview**.

Name	FirstName	LastName
王若琳	若琳	王
刘耕宏	耕宏	刘
许惠欣	惠欣	许
蔡依林	依林	蔡
水沫	沫	水
萧亞軒	亞軒	萧
郑元畅	元畅	郑
林依晨	依晨	林

You can also type other valid and invalid names to see how the input data is parsed.

You can use the Trace feature to see a graphical representation of either the final parsing results or to step through the parsing events. Click the link in the **Trace** column to see the Trace Details for the data row.

Write to File

The template contains one Write to File stage. In addition to the input field, the output file contains the **LastName**, and **FirstName** fields.

For complete information about Open Parser and the Domain Editor, see Open Parser in the Data Normalization Module help or in the Spectrum™ Technology Platform User's Guide.

Parsing E-mail Addresses

This template demonstrates how to parse e-mail addresses into component parts. The parsing rule separates each token in the **Email** field and copies each token to three fields: **Local-Part**, **DomainName**, and **DomainExtension**. **Local-Part** represents the domain name part of the e-mail address, **DomainName** represents the domain name of the e-mail address, and **DomainExtension** represents the domain

extension of the e-mail address. For example, in `pb.com`, "`pb`" is the domain name and "`com`" is the domain extension.

The internet is a great source of public domain information that can aid you in your open parsing tasks. In this example, e-mail formatting information was obtained from various internet resources and was then imported into Table Management to create a table of domain values. The domain extension task that you will perform in this template activity demonstrates the usefulness of this method.

This template also demonstrates how to effectively use table data that you load into Table Management to perform table look-ups as part of your parsing tasks.

Template name: ParseEmail.df

Sample input file name: Email_Test.csv, Email_Domains.txt

Sample output file name: Email_Test_Output.csv

Modules required: Data Normalization Module

Business Scenario

You work for an insurance company that wants to do its first e-mail marketing campaign. Your database contains e-mail addresses of your customers and you have been asked to find a way to make sure that those e-mail addresses are in a valid SMTP format.

Solution

Before you create this dataflow, you will need to load a table of valid domain names extensions in Table Management so that you can look up domain name extensions as part of the validation process.

The following dataflow provides a solution to the business scenario:

In this dataflow, data is read from a file and processed through the Open Parser stage. For each data row in the input file, this dataflow will do the following:

Create a Domain Extension Table

The first task is to create an Open Parser table in Table Management that you can use to check if the domain extensions in your e-mail addresses are valid.

1. From the **Tools** menu, select **Table Management**.
2. In the **Type** list, select **Open Parser**.
3. Click **New**.
4. In the **Add User Defined Table** dialog box, type `EmailDomains` in the **Table Name** field, make sure that **None** is selected in the **Copy from** list, and then click **OK**.
5. With **EmailDomains** displayed in the **Name** list, click **Import**.

6. In the **Import** dialog box, click **Browse** and locate the source file for the table. The default location is: `<drive>:\Program Files\Pitney Bowes\Spectrum\server\modules\coretemplates\data\ Email_Domains.txt`. Table Management displays a preview of the terms contained in the import file.
7. Click **OK**. Table Management imports the source files and displays a list of internet domain extensions.
8. Click **Close**. The `EmailDomains` table is created. Now create the dataflow using the ParseEmail template.

Read from File

This stage identifies the file name, location, and layout of the file that contains the eÂmail addresses you want to parse.

Open Parser

The Open Parser stage parsing grammar defines the following commands and expressions:

- `%Tokenize` is set to None. When `Tokenize` is set to `None`, the parsing grammar rule must include any spaces or other token separators within its rule definition.
- `%InputField` is set to parse input data from the **Email_Address** field.
- `%OutputFields` is set to copy parsed data into three fields: **Local-Part**, **DomainName**, and **DomainExtension**.
- The root expression defines the pattern of tokens being parsed:

```
<root> = <Local-Part>"@"<DomainName>"."<DomainExtension>;
```

The rule variables that define the domain must use the same names as the output fields defined in the required `OutputFields` command.

- The remainder of the parsing grammar defines each of the rule variables as expressions.

```
<Local-Part> = (<alphanum> ".")* <alphanum> | (<alphanum> "_")* <alphanum>
;
<DomainName> = (<alphanum> ".")? <alphanum>;
<DomainExtension> = @Table("EmailDomains")* "."? @Table("EmailDomains");
<alphanum>=@RegEx("[A-Za-z0-9]+");
```

The `<Local-Part>` variable is defined as a string of text that contains the `<alphanum>` variable, the period character, and another `<alphanum>` variable.

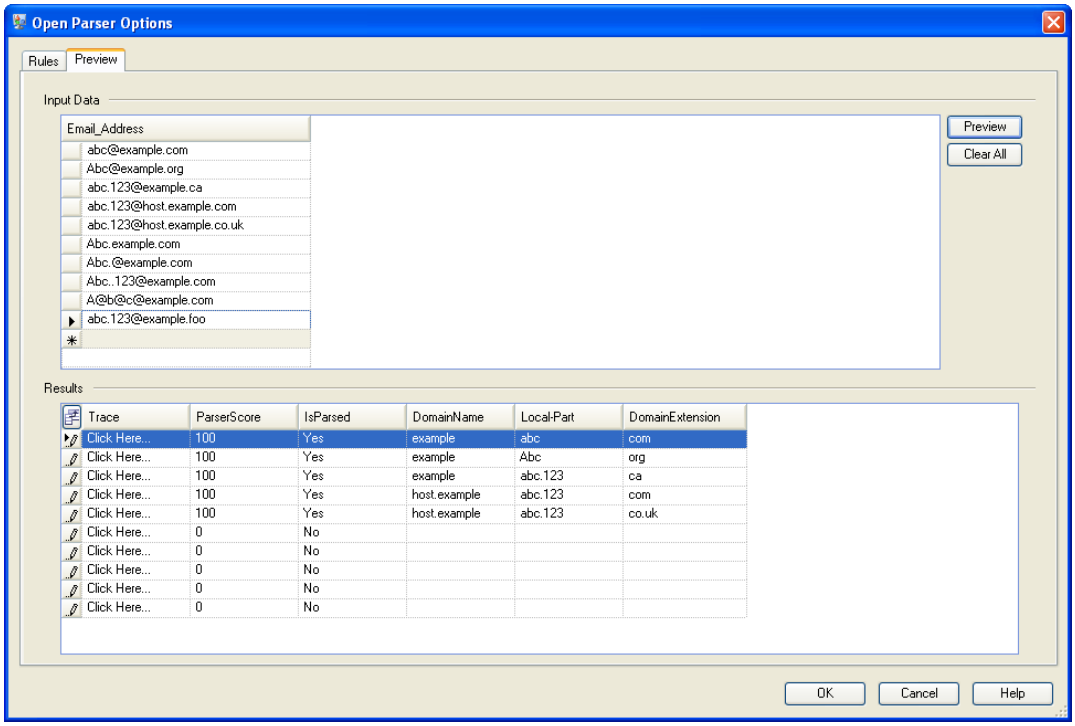
The `<alphanum>` variable definition is a regular expression that means any string of characters from A to Z, a to a, and 0-9. The `<alphanum>` variable is used throughout this parsing grammar and is defined once on the last line of the parsing grammar.

The parsing grammar uses a combination of regular expressions and literal characters to build a pattern for e-mail addresses. Any characters in double quotes in this parsing grammar are literal characters, the name of a table used for lookup, or a regular expression. The parsing grammar uses these special characters:

- The "+" character means that a regular expression can occur one or more times.
- The "?" character means that a regular expression can occur zero or one time.
- The "|" character means that the variable has an OR condition.
- The ";" character means end of a rule.

Use the **Commands** tab to explore the meaning of the other special symbols you can use in parsing grammars by hovering the mouse over the description.

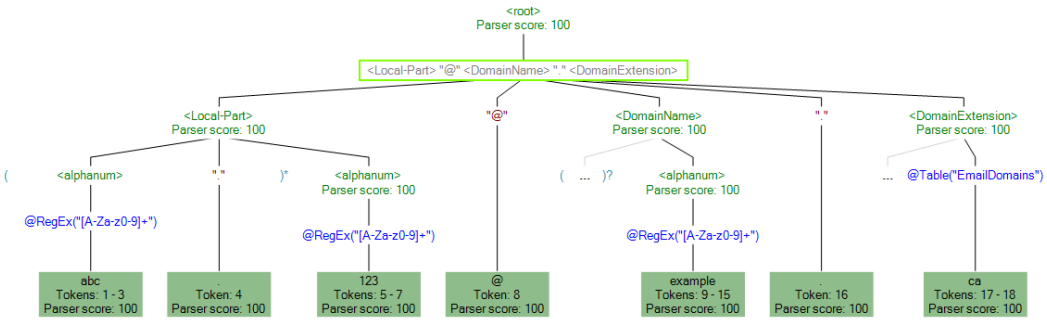
To test the parsing grammar, click the Preview tab. Type the e-mail addresses shown below in the **Email Address** field and then click **Preview**.



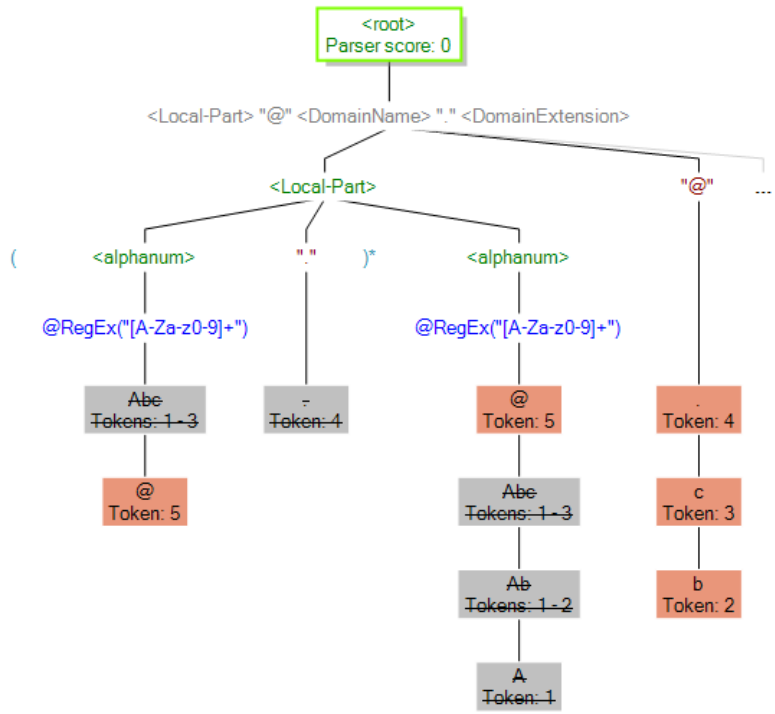
You can also type other e-mail addresses to see how the input data is parsed.

You can also use the Trace feature to see a graphical representation of either the final parsing results or to step through the parsing events. Click the link in the **Trace** column to see the Trace Details for the data row.

Trace Details shows a matching result. Compare the tokens matched for each expression in the parsing grammar.



You can also use Trace to view non-matching results. The following graphic shows a non- matching result. Compare the tokens matched for each expression in the parsing grammar. The reason that this input data (`Abc.example.com`) did not match is because it did not contain all of the required tokens to match—there is no @ character separating the Local- Part token and the Domain tokens.



Write to File

The template contains one Write to File stage. In addition to the input field, the output file contains the **Local-Part**, **DomainName**, **DomainExtension, IsParsed, and ParserScore** fields.

For complete information about Open Parser, the Domain Editor, and the parsing grammar reference, see Open Parser in the Data Normalization Module help or in the Spectrum™ Technology Platform User's Guide.

Parsing Spanish and German Names

This template demonstrates how to parse mixed-culture names, such as Spanish and German names, into component parts. The parsing rule separates each token in the **Name** field and copies each token to the fields defined in the Personal and Business Names parsing grammar. For more information about this parsing grammar, select **Tools** > **Open Parser Domain Editor** and then select the **Personal and Business Names** domain and either the **German (de)** or **Spanish (es)** cultures.

This template also applies gender codes to personal names in using table data contained in Table Management. For more information about Table Management, select **Tools** > **Table Management**.

Template name: ParseSpanish&GermanNames.df

Sample input file name: MixedNames.csv

Sample output file name: ParseSpanishGermanNames-BusinessNames-Output.csv and ParseSpanishGermanNames-PersonalNames-Output.csv

Tables: Base and Enhanced

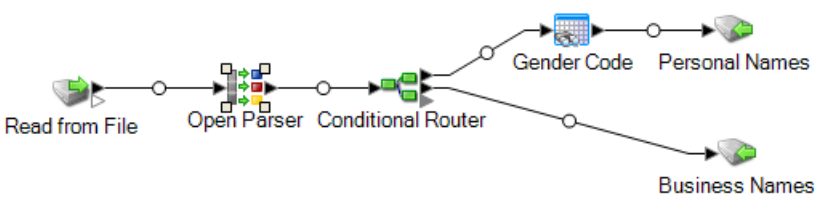
Modules required: Data Normalization Module

Business Scenario

You work for a pharmaceuticals company based in Brussels that has consolidated its Germany and Spain operations. Your company wants to implement a mixed-culture database containing name data and it is your job to analyze the variations in names between the two cultures.

Solution

The following dataflow provides a solution to the business scenario:



In this dataflow, data is read from a file and processed through the Open Parser stage. For each data row in the input file, this data flow will do the following:

Read from File

This stage identifies the file name, location, and layout of the file that contains the names you want to parse. The file contains both male and female names and includes CultureCode information for each name. The CultureCode information designates the input names as either German (de) or Spanish (es).

Open Parser

This stage defines whether to use a culture-specific domain grammar created in the Domain Editor or to define a domain-independent grammar. A culture-specific parsing grammar that you create in the Domain Editor is a validated parsing grammar that is associated with a culture and a domain. A domain-independent parsing grammar that you create in Open Parser is a validated parsing grammar that is not associated with a culture and domain. For more information about the two types of parsing grammars, see the Spectrum™ Technology Platform User's Guide.

In this template, the option is set to use a culture-specific domain grammar.

The Open Parser option is set to use the Personal and Business Names domain and the default culture is Spanish. Input data that does not contain a culture code will use the default culture.

To test this culture-specific domain grammar, click the Preview tab and then type the names shown in the **Write to File** on page 61 output and then click **Preview**. You can also type other valid and invalid names to see how the input data is parsed.

You can also use the Trace feature to see a graphical representation of either the final parsing results or to step through the parsing events. Click the link in the **Trace** column to see the Trace Details for the data row.

Conditional Router

This stage routes the input so that personal names are routed to the Gender Codes stage and business names are routed to the Business Names stage.

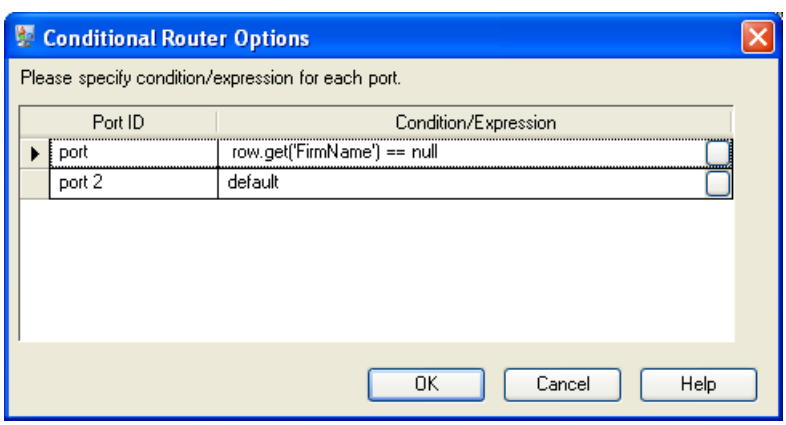


Table Lookup

Double-click this stage on the canvas and then click **Modify** to display the table lookup rule options.

The **Categorize** option uses the Source value as a key and copies the corresponding value from the table entry into the field selected in the Destination list. In this template, **Complete field** is selected and **Source** is set to use the **FirstName** field. Table Lookup treats the entire field as one string and flags the record if the string as a whole can be categorized.

The **Destination** is set to the **GenderCode** field and uses the lookup terms contained in the **Gender Codes** table to perform the categorization of male and female names. If a term in the input data is not found, Table Lookup assigns a value of **U**, which means unknown. To better understand how this works, select **Tools** > **Table Management** and select the Gender Codes table.

Write to File

The template contains one Write to File stage. In addition to the input field, the personal names output file contains the **Name**, **TitleOfRespect, FirstName, MiddleName, LastName, PaternalLastName, MaternalLastName, MaturitySuffix, GenderCode, CultureUsed, and ParserScore** fields.

The business names output file contains the **Name**, **FirmName**, **FirmSuffix**, **CulureUsed**, and **ParserScore** fields.

The personal names output is as follows:

Name	Title	FirstName	MiddleName	LastName	PaternalLastNam	MaternalLastNa	MaturitySuf	GenderCode
Karl-Theodor Anton Maria von Dahlberg		Karl-Theodor	Anton Maria	von Dahlberg				U
Friedrich Emil Ferdinand Heinrich Graf von Nollend...		Friedrich	Emil Ferdinand H...	Graf von Nollend...				M
Bruno Hans Otto Friedrich von Görschen		Bruno	Hans Otto Friedri...	von Görschen				M
Johann Christoph Bach III		Johann	Christoph	Bach			III	M
Berhardine Baronin von Hammerstein		Berhardine		Baronin von Ha...				U
Frau Margret Gertru Viehof	Frau	Margret	Gertru	Viehof				F
Herr Robert Michael Kandleinsberger, Sen	Herr	Robert	Michael	Kandleinsberger			Sen	M
Ángela López de Portillo		Ángela			López	de Portillo		U
José María Álvarez del Manzano y López del Hierro		José María			Álvarez del Manzano	y López del Hierro		U
José Luis Pablo Rodríguez Zapatero		José Luis Pa...			Rodríguez	Zapatero		U
Abraham Lorenzo de Jesus Gonzalez de la Fuente		Abraham Lor...			Gonzalez	de la Fuente		M

The business names output is as follows:

Name	FirmName	FirmSuffix
Deutsche Telekom	Deutsche Telekom	
Ruhr Universität	Ruhr Universität	
Reiner Brach Gmbh & Co KG	Reiner Brach	Gmbh & Co KG
ZXY DISEÑO ARQUITECTONICO S A DE C V	ZXY DISEÑO ARQUITECTONICO	S A DE C V
VEHICULOS AUTOMOTICES Y MARINOS S A DE...	VEHICULOS AUTOMOTICES Y MARINOS	S A DE C V
VAZQUEZ Y VIVE ABOGADOS ASOCIADOS S C	VAZQUEZ Y VIVE ABOGADOS ASOCIADOS	S C
VARGAS MARTINEZ PALACIOS S C	VARGAS MARTINEZ PALACIOS	S C

For complete information about Open Parser, the Domain Editor, and the parsing grammar reference, see Open Parser in the Data Normalization Module help or in the Spectrum™ Technology Platform User's Guide.

Parsing U.S. Phone Numbers

This template demonstrates how to parse U.S. phone numbers into component parts. The parsing rule separates each token in the **PhoneNumber** field and copies each token to four fields: **CountryCode**, **AreaCode**, **Exchange**, and **Number**.

Template name: ParseUSPhoneNumbers.df

Sample input file name: PhoneNumbers.csv

Sample output file name: ParsePhoneNumbers-Output.csv

Modules required: Data Normalization Module

Business Scenario

You work for a wireless provider and have been assigned a project to analyze incoming phone number data for a growing region of your business.

Solution

The following dataflow provides a solution to the business scenario:

In this dataflow, data is read from a file and processed through the Open Parser stage. For each data row in the input file, this data flow will do the following:

Read from File

This stage identifies the file name, location, and layout of the file that contains the phone numbers you want to parse.

Open Parser

This stage defines whether to use a culture-specific domain grammar created in the Domain Editor or to define a domain-independent grammar. A culture-specific parsing grammar that you create in the Domain Editor is a validated parsing grammar that is associated with a culture and a domain. A domain-independent parsing grammar that you create in Open Parser is a validated parsing grammar that is not associated with a culture and domain. For more information about the two types of parsing grammars, see the Spectrum™ Technology Platform User's Guide.

In this template, the parsing grammar is defined as a domain-independent grammar.

The Open Parser stage contains a parsing grammar that defines the following commands and expressions:

- `%Tokenize` is set to None. When `Tokenize` is set to `None`, the parsing grammar rule must include any spaces or other token separators within its rule definition.
- `%InputField` is set to parse input data from the **PhoneNumber** field.
- `%OutputFields` is set to separate parsed data into four fields: **CountryCode**, **AreaCode**, **Exchange**, and **Number**.
- The `<root>` expression defines pattern of tokens being parsed and includes OR statements (|), such that a valid phone number is:
- **CountryCode**, **AreaCode**, **Exchange**, and **Number** OR
- **AreaCode**, **Exchange**, and **Number** OR
- **Exchange** and **Number**

The parsing grammar uses a combination of regular expressions and literal characters to build a pattern for phone numbers. Any characters in double quotes in this parsing grammar are literal characters or a regular expression.

The plus character (+) used in this `<root>` command is defined as a literal character because it is encapsulated in quotes. You can use single or double quotes to indicate a literal character. If the plus character is used without quotes, it means that the expression it follows can occur one or more times.

The phone number domain rules are defined to match the following character patterns:

- Zero or one occurrence of a "+" character.
- The **CountryCode** rule, which is a single digit between 0-9.
- Zero or one occurrence of an open parentheses or a hyphen or a space character. Two of these characters occurring in sequence results in a non-match, or in other words, an invalid phone number.
- The **AreaCode** rule, which is a sequence of exactly three digits between 0-9.
- Zero or one occurrence of an open parentheses or a hyphen or a space character. Two of these characters occurring in sequence results in a non-match, or in other words, an invalid phone number.
- The **Exchange** rule, which is a sequence of exactly three digits between 0-9.
- Zero or one occurrence of an open parentheses or a hyphen or a space character. Two of these characters occurring in sequence results in a non-match, or in other words, an invalid phone number.
- The **Number** rule, which is a sequence of exactly four digits between 0-9.

The rule variables that define the domain must use the same names as the output fields defined in the required `OutputFields` command.

Regular Expressions and Expression Quantifiers

The parsing grammar uses a combination of regular expressions and expression quantifiers to build a pattern for U.S. phone numbers. The parsing grammar uses these special characters:

- The "?" character means that a regular expression can occur zero or one time.
- The (|) character indicates an OR condition.
- The ";" character means end of a rule.

Use the **Commands** tab to explore the meaning of the other special symbols you can use in parsing grammars by hovering the mouse over the description.

Using the Preview Tab

To test the parsing grammar, click the Preview tab. Type the phone numbers shown below in the **PhoneNumber** field and then click **Preview**.

PhoneNumber	CountryCode	AreaCode	Exchange	Number
1(410)286-7334	1	410	286	7334
14042867534	1	404	286	7534
(410)286-7256		410	286	7256
301-868-9999		301	868	9999
1-222-458-7799	1	222	458	7799
+1(410)286-7334	1	410	286	7334
901 888 9990		901	888	9990
1 410 888 2345	1	410	888	2345
234-4567			234	4567
234 6789			234	6789

You can also type other valid and invalid phone numbers to see how the input data is parsed.

You can also use the Trace feature to see a graphical representation of either the final parsing results or to step through the parsing events. Click the link in the **Trace** column to see the Trace Details for the data row.

Write to File

The template contains one Write to File stage. In addition to the input field, the output file contains the **CountryCode**, **AreaCode**, **Exchange**, and **Number** fields.

For complete information about Open Parser, the Domain Editor, and the parsing grammar reference, see Open Parser in the Data Normalization Module help or in the Spectrum™ Technology Platform User's Guide.

Index

G

I

J

L

M

N

O

P

R

S

T

U

V

W