



Address Standardization in Java

A State Machine Implementation

Celeritas Technologies LLC

January 11, 2005

Prepared By:

Celeritas Technologies, LLC

7101 College Blvd, Sixth Floor

Overland Park, KS 66210

(913)491-9000



**Copyright © 2005 Celeritas Technologies,
L.L.C.**

This work was created by Celeritas Technologies, L.L.C. ("Creator").

This work and all rights therein and thereto, including copyright rights and/or patent rights, are owned by Creator and/or another entity (collectively, "Owner"). This shall serve as notice of such ownership as of the date of this and associated files or subject matter, if any, as depicted above and/or as depicted with an electronic file date stamp and/or any versions thereof and their associated dates, if any. This work may not be reproduced for any purpose, distributed, modified, reverse-engineered, stored in a retrieval system, transmitted, used, made, offered for sale, or sold, in whole or part, in any form or by any means, electronic, mechanical, audio, photocopying, recording, or otherwise, without the prior written permission of Owner. This work may not be exported unless in compliance with the applicable technology export laws. While this information is presented in good faith and believed to be accurate, Creator does not guarantee satisfactory or any results from reliance upon such information. Creator reserves the right, without notice, to alter or improve the designs, specifications, creations, or works of the subject matter herein. Nothing herein is to be construed as a warranty or guarantee, express or implied, against infringement, or regarding performance, merchantability, fitness, or any other matter with respect to products, processes, or any other subject matter herein, and such warranties and guaranties are expressly disclaimed. Nothing herein is to be construed as a recommendation to use any product or process in conflict with any third party rights in any intellectual property. All products, languages, or trademarked names that are mentioned in this work are acknowledged to be the proprietary property of the respective owner.



Copyright © 2005

Celeritas Technologies, LLC Address Standardization - 2

Introduction

There are many software systems in existence today that make use of a mailing address value. A mailing address is a common attribute of a customer, business, contact or a host of other objects in a system. At times the actual validity of this address is not important. However, some systems require that the address be valid. There are a number of ways to measure the validity of an address, such as whether it exists, or whether mail can actually be sent to the address. A similar operation on an address is used in Geographic Information Systems (GIS) in which the address needs to be geocoded.

In all these scenarios, third party software can be used to perform the desired operation but the address needs to first be standardized into a format known by these systems. Many systems include address standardization operations by default but these systems can be expensive, especially if the number of operations needed is very large. This short article examines the problem of providing a mailing address in a standard format so that a geocoding operation can be performed on that address.

Examples

To illustrate the problem clearly several examples may be in order. Let's assume a system encounters an address:

1200 Main Street North

In order to use this address in a Geocoding operation the system must first determine if it is in a standard format. What do the different tokens in the above String represent? Is 1200 the number? Is the street name 'Main', 'Main Street' or 'Main Street North'? Should any of the address line be abbreviated? Consider a deviation:

Main Street North 1200

Does the ordering make this address invalid?

To solve the problem, this Address Line should be standardized to:

1200 MAIN ST N

United State Postal Service Publication 28

The United States Postal Service has published a document that tackles this very issue. The document is called *Publication 28* or *Pub28*. The point of *Pub28* is to detail address standardization rules for software systems. This document attempts to lay out comprehensive rules for address standardization. When implementing a system to perform address standardization it should adhere to the rules defined in *Pub28*.



Copyright © 2005

Celeritas Technologies, LLC Address Standardization - 3

One common aspect of an address that *Pub28* defines is the valid street suffix values for US streets and their valid abbreviation mappings. Thus “ROAD” is a valid suffix and should be abbreviated as RD. “STREET” is valid and maps to ST. “MAIN” is not a standard suffix but an actual street name.

Solution

To illustrate the solution we will look at parsing the Address Delivery Line as defined by *USPS Pub28*. The Address Delivery Line is in the format of:

[St Num] [Pre] [St Name(s)] [Suffix][Post] [2nd Addr Ind] [2nd Addr Range]

- **St Num** – The Street number
- **Pre** – a Predirectional value
- **St Name** – The name of the street. This could be multi-tokened names.
- **Suffix** – a value from a predefined set of Street suffix values.
- **Post** – a Postdirectional value
- **2nd Addr Ind** – A token from a set of defined Secondary Address Indicators.
- **2nd Addr Range** – A range associated with the 2nd Addr Ind.

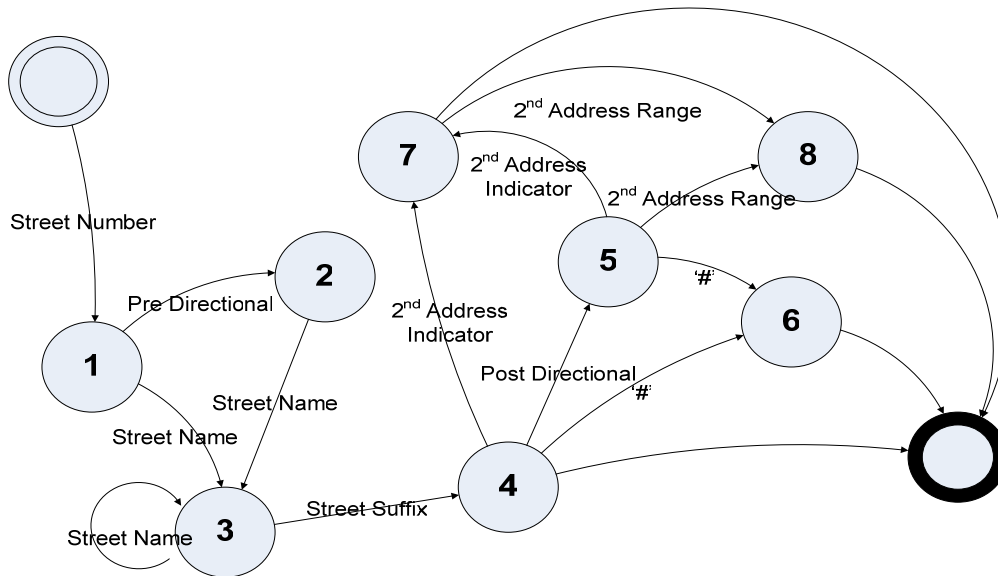
Please consult *Pub28* for a comprehensive explanation and exhaustive examples but to illustrate consider these examples of a standardized address:

- 120 N MAIN ST
- 120 MAIN ST S APT 5A
- 120 N ROCK CREED RD W APT 10

As shown not all the tokens are required.



Address Delivery Line State Chart



Address Delivery Line State Parsing Implementation

This section is the Java source solution for the above state chart. Some implementation details have been removed to concentrate on the core problem described in the previous sections. Notice upon inspection of the different states that Java Collections are used to define the standardization mappings for Suffixes, Directional and Secondary Address Indicators. A `StandardizationException` is thrown when a state is reached with additional tokens remaining but no valid State transition is defined.

```

/**
 * This method takes a USPS Delivery Address Line and parses the string
 * into street number, street name, pre-directional, post-directional,
 * street suffix, secondary address indicator, secondary address range.
 *
 * @param psDeliveryAddressLine The USPS Delivery Address Line to parse
 * and Standardize. Value cannot be null. An empty String value just
 * returns in essence doing nothing.
 * @throws StandardizationException Thrown when the address line
 * is not in a valid format and cannot be parsed.
 */
public void setDeliveryAddressLine(String psDeliveryAddressLine)
    throws StandardizationException {

    //
    // ... perform parameter validation and preprocessing steps
    //

    //
    // first make uppercase and parse the string based on white space
    //
    String[] rParsedAddrElements = psDeliveryAddressLine.toUpperCase().split("\\s");
    //
    // finite state machine implementation of parsing Delivery Address Line
  
```



Copyright © 2005

Celeritas Technologies, LLC Address Standardization - 5

```

// and setting StandardizedAddress attributes.
//
// Address line is processed from left to right
// The beginning state is 0.
// state 0:
//   Street Number          ---> State 1
// state 1:
//   Pre-directional        ---> State 2
//   Street Name            ---> State 3
// state 2:
//   Street Name            ---> State 3
// state 3:
//   Street Name            ---> State 3
//   Suffix                  ---> State 4
// state 4:
//   Post-directional       ---> State 5
//   '#'                      ---> State 6
//   2nd Address Ind         ---> State 6
//   <end of tokens>         ---> State 7
// state 5:
//   '#'                      ---> State 6
//   2nd Address Ind         ---> State 6
//   <end of tokens>         ---> State 7
// state 6:
//   2nd Address Ranges     ---> State 7
// state 7:
//   <end of tokens>         ---> <end>
//
int nState = 0;
boolean bContinueParse = true;
//
// Determine the index position of the Last occurrence of a Street suffix.
//
int nLastSuffix = getLastSuffixIndexPos(rParsedAddrElements);
for ( int nIndex = 0; nIndex < rParsedAddrElements.length; nIndex++ ) {
switch (nState) {
case 0:
if ( isAlphaNumeric(rParsedAddrElements[nIndex]) ) {
this.sStreetNumber = rParsedAddrElements[nIndex];
nState = 1;
//
// this default branch assumes an error if the first
// token is not a numeric string.
//
} else {
throw new StandardizationException("Invalid State " + nState
+ " encountered processing token " + rParsedAddrElements[nIndex]
+ " in Delivery Address Line: " + psDeliveryAddressLine);
}
break;
case 1:
if ( DIRECTIONALS.get(rParsedAddrElements[nIndex]) != null ) {
this.sStreetPreDirectional =
(String)DIRECTIONALS.get(rParsedAddrElements[nIndex]);
nState = 2;
} else {
this.sStreetName = rParsedAddrElements[nIndex];
nState = 3;
}
break;
case 2:
this.sStreetName = rParsedAddrElements[nIndex];
nState = 3;
break;
case 3:
//
// If the element is a Suffix
// abbreviation and it is the last Suffix parse the

```



```

        // street suffix, otherwise add to the street name.
        //
    if ( SUFFIX_ABBR_MAP.get(rParsedAddrElements[nIndex]) != null
        && nLastSuffix == nIndex ) {
        this.sStreetSuffix =
            (String)SUFFIX_ABBR_MAP.get(rParsedAddrElements[nIndex]);
        nState = 4;
    } else {
        this.sStreetName += " " + rParsedAddrElements[nIndex];
        nState = 3;
    }
}
break;
case 4:
    if ( SECONDARY_INDICATOR_NO_RANGE_MAP.get(rParsedAddrElements[nIndex])
        != null ) {
        this.sSecondaryAddrIndicator =
            (String)
                SECONDARY_INDICATOR_NO_RANGE_MAP
                    .get(rParsedAddrElements[nIndex]);
        nState = 7;
    } else if ( this.SECONDARY_INDICATOR_MAP.get(rParsedAddrElements[nIndex])
        != null ) {
        this.sSecondaryAddrIndicator =
            (String)SECONDARY_INDICATOR_MAP.get(rParsedAddrElements[nIndex]);
        nState = 6;
    } else if ( DIRECTIONALS.get(rParsedAddrElements[nIndex]) != null ) {
        this.sStreetPostDirectional =
            (String)DIRECTIONALS.get(rParsedAddrElements[nIndex]);
        nState = 5;
    } else if ( "#" .equals( rParsedAddrElements[nIndex] ) ) {
        this.sSecondaryAddrIndicator = rParsedAddrElements[nIndex];
        nState = 6;
        //
        // If a street suffix is encountered here then a street suffix
        // was apart of the street name. Append the suffix to the name
        // and set the suffix to the current token.
        //
    } else if ( SUFFIX_ABBR_MAP.get(rParsedAddrElements[nIndex]) != null ) {
        this.sStreetName += " " + rParsedAddrElements[nIndex-1];
        this.sStreetSuffix =
            (String)SUFFIX_ABBR_MAP.get(rParsedAddrElements[nIndex]);
        nState = 4;
    } else if ( this.isAlphaNumeric(rParsedAddrElements[nIndex])
        && nIndex == rParsedAddrElements.length-1 ) {
        this.sSecondaryAddrIndicator = "#";
        this.sSecondaryAddressRange = rParsedAddrElements[nIndex];
        nState = 7;
    } else {
        throw new StandardizationException("Invalid State " + nState
            + " encountered processing token " + rParsedAddrElements[nIndex]
            + " in Delivery Address Line: " + psDeliveryAddressLine);
    }
}
break;
case 5:
    if ( SECONDARY_INDICATOR_NO_RANGE_MAP.get(rParsedAddrElements[nIndex])
        != null ) {
        this.sSecondaryAddrIndicator =
            (String)
                SECONDARY_INDICATOR_NO_RANGE_MAP
                    .get(rParsedAddrElements[nIndex]);
        nState = 7;
    } else if ( this.SECONDARY_INDICATOR_MAP.get(rParsedAddrElements[nIndex])
        != null ) {
        this.sSecondaryAddrIndicator =
            (String)SECONDARY_INDICATOR_MAP.get(rParsedAddrElements[nIndex]);
        nState = 6;
    } else if ( "#" .equals( rParsedAddrElements[nIndex] ) ) {
        this.sSecondaryAddrIndicator = rParsedAddrElements[nIndex];
    }
}

```



Copyright © 2005

Celeritas Technologies, LLC Address Standardization - 7

```

        nState = 6;
        //
        // throw exception if none of the cases above match
        //
    } else {
        throw new StandardizationException("Invalid State " + nState
            + " encountered processing token " + rParsedAddrElements[nIndex]
            + " in Delivery Address Line: " + psDeliveryAddressLine);
    }
    break;
case 6:
    this.sSecondaryAddressRange = rParsedAddrElements[nIndex];
    nState = 7;
    break;
case 7:
    //
    // This State 7 should really never be reached as the
    // loop should run out of tokens at the previous State.
    //
    nState = -1;
    bContinueParse = false;
    break;
default:
    throw new StandardizationException("Invalid State encountered "
        + "parsing Delivery Address Line: "
        + psDeliveryAddressLine);
}

if ( !bContinueParse ) {
    break;
}
}
}

```

Resources

1. *Postal Address Standards, Publication 28*. United States Postal Service.
<http://pe.usps.gov/cpim/ftp/pubs/Pub28/pub28.pdf>

