Programming Implementation Guide for Standard Search Logic

Table of Contents

Statement of Purpose

The Information Technology Section of the International Association of Commercial Administrators adopted a Resolution at the Annual Conference of the Association in May 2005. The Resolution stated:

> *IACA's Information Technology Section resolves to develop, in collaboration with the membership of the Secured Transaction Section, a programming implementation guide to assist jurisdictions in developing consistent standard search logic. The guide should be used by filing offices in performing indexing and searching tasks in a manner compliant with Revised Article 9 and Administrative Rules as adopted by jurisdictions.*

This Implementation Guide has been created pursuant to this Resolution. A Summit meeting was held in San Antonio, Texas, USA, in August 2005 with a goal of contributing to the development of this Implementation Guide. It is only due to the dedicated effort of many IACA members, representatives of filing offices throughout the United States and Canada, and representatives of private entities that work in the area of secured transactions that this effort has culminated in this document. Special recognition for contributions to this effort is merited for the following individuals:

> Robert Lindsey, Virginia Corporation Commission
> Randy Moes, Texas Secretary of State's Office
> Tim Poulin, Maine Secretary of State's Office
> Kathy Sachs, Kansas Secretary of State's Office
> Lynette Wong, California Secretary of State's Office
> Trish Bogenrief, Corporation Service Company
> Wally Boggus, Capitol Services Company
> Carl Ernst, Ernst Publishing
> Paul Hodnefield, Corporation Service Company
> Tom Ose, Ose Micro Solutions

This document has been prepared for submission to IACA at its 2006 Annual Conference in Incline Village, Nevada, USA.

Trevor Timmons, Colorado Secretary of State's Office

Development of the Implementation Guide

This implementation guide has been developed to assist jurisdictions in developing and implementing indexing and searching procedures for use in performing duties required by the Uniform Commercial Code (UCC). Some of the noteworthy sections of the UCC[1] that inform the effort are (**emphasis added**):

> **SECTION 9-503. NAME OF DEBTOR AND SECURED PARTY.**
> (a) A financing statement sufficiently provides the name of the debtor:
>> **(1) if the debtor is a registered organization, only if the financing statement provides the name of the debtor indicated on the public record of the debtor's jurisdiction of organization which shows the debtor to have been organized;**
>> (2) if the debtor is a decedent's estate, only if the financing statement provides the name of the decedent and indicates that the debtor is an estate;
>> (3) if the debtor is a trust or a trustee acting with respect to property held in trust, only if the financing statement:
>>> (A) provides the name, if any, specified for the trust in its organic documents or, if no name is specified, provides the name of the settlor and additional information sufficient to distinguish the debtor from other trusts having one or more of the same settlors; and
>>> (B) indicates, in the debtor's name or otherwise, that the debtor is a trust or is a trustee acting with respect to property held in trust; and
>> (4) in other cases:
>>> (A) if the debtor has a name, only if it provides the individual or organizational name of the debtor; and
>>> (B) if the debtor does not have a name, only if it provides the names of the partners, members, associates, or other persons comprising the debtor.
> (b) A financing statement that provides the name of the debtor in accordance with subsection (a) is not rendered ineffective by the absence of:
>> (1) a trade name or other name of the debtor; or
>> (2) unless required under subsection (a)(4)(B), names of partners, members, associates, or other persons comprising the debtor.
> (c) A financing statement that provides only the debtor's trade name does not sufficiently provide the name of the debtor.
> (d) Failure to indicate the representative capacity of a secured party or representative of a secured party does not affect the sufficiency of a financing statement.
> (e) A financing statement may provide the name of more than one debtor and the name of more than one secured party.

---

[1] Citations from the Model Act as approved by NCCUSL, July 1998

SECTION 9-506. EFFECT OF ERRORS OR OMISSIONS.
(a) A financing statement substantially complying with the requirements of this part is effective, even if it includes minor errors or omissions, unless the errors or omissions make the financing statement seriously misleading.
(b) Except as otherwise provided in subsection (c), a financing statement that fails sufficiently to provide the name of the debtor in accordance with Section 9-503(a) is seriously misleading.
(c) If a search of the records of the filing office under the debtor's correct name, using the filing office's standard search logic, if any, would disclose a financing statement that fails sufficiently to provide the name of the debtor in accordance with Section 9-503(a), the name provided does not make the financing statement seriously misleading.
(d) For purposes of Section 9-508(b), the "debtor's correct name" in subsection (c) means the correct name of the new debtor.

Distilled to the essentials, these sections mean:

- Filers must use the correct debtor name on financing statements for it to be effective;
- If a debtor name is not indicated correctly, the financing statement in ineffective;
- UNLESS the filing office's standard search logic would disclose the financing statement.

The intent of Revised Article 9 is clearly to require that filers of financing statements provide the debtor's correct name. It is just as clear that the capabilities available to searchers as "standard search logic" are the crucial nexus between searchers discovering financing statements that are effective and filers being aware of the rigor of the standard to which their filings will be held.

This implementation guide has been drafted to provide authoritative guidance to filing offices that are either considering changes to the manner in which they index and retrieve UCC filings or are interested in their jurisdiction's position relative to IACA recommendations on the implementation of standard search logic.

It may be apparent but is worthy of restatement that standard search logic is inseparable from indexing practices. As one example, Step #5 entails accounting for specific ways in which your office has handled the word "THE" where it is the first word in an entity name. Perhaps your data entry procedures currently or in the past have resulted in the leading word "THE" being moved to the end of entity names (such that "THE COMPANY" would be indexed as "COMPANY, THE"). In this example, the proper execution of Step #5 for an SSL search would entail performing the same operation on the search criteria provided by a searcher, so that a search string of "THE BADGER PARTNERSHIP" would be modified by this step to "BADGER PARTNERSHIP, THE". If your office's data entry practices have involved similar movement of leading words such as "A" or "AN", then those words should also be a part of Step #5 as performed in your jurisdiction.

Authoritative Guidance on the Process of Implementing IACA Standard Search Logic

The implementation of a standard search logic or SSL requires that actions be performed in a specific order in a consistent manner. The actions must be performed at two different stages at a minimum:

- Initial indexing of debtor names for inclusion in a searchable index; and,
- Preparation of search criteria for performing searches against that searchable index.

Many jurisdictions have also created batch processes that can be used to regenerate searchable indices when conditions change (such as the addition of a new business entity name ending to the statutorily allowed name endings).

There are several entity names provided for purposes of illustration of the application of each of these steps. The entity names prior to any processing are shown below, and are also shown at the bottom of each of the steps to show the effects of each of the steps.

Entity Name Normalization Examples – Original Names	
Example # 1	The Partnership Company Ltd
Example # 2	Joe's Bar & Grill of Tahoe, Incorporated
Example # 3	ABC Partnership, Limited Liability Company
Example # 4	The #1 Best Company in America, Hands Down!, Inc.

The steps recommended to be performed, and the order in which they should be performed, are here laid out as authoritative guidance for use by filing offices:

For Organization Names:

STEP #1 – Convert the character case to a standard case (upper or lower)

This step should provide for consistent results regardless of the character case of debtor names. If your software or database provides for native case-insensitive searching, this step may not apply.

Entity Name Normalization Examples – After Step #1[2]	
Example # 1	THE PARTNERSHIP COMPANY LTD
Example # 2	JOE'S BAR & GRILL OF TAHOE, INCORPORATED
Example # 3	ABC PARTNERSHIP, LIMITED LIABILITY COMPANY
Example # 4	THE #1 BEST COMPANY IN AMERICA, HANDS DOWN!, INC.

---

[2] For purposes of this document, assume that the standard character case used for every transformation and comparison is upper case.

STEP #2 – Convert "&" to "AND"

This step is used to ensure that the ampersand symbol is treated as an
equivalent of the connecting word "AND". Since the next step involves
the removal of punctuation marks (as well as other special characters), this
step should be performed at this stage to provide that the common usage
of "&" as a substitute for "AND" is treated appropriately.

Entity Name Normalization Examples – After Step #2	
Example # 1	THE PARTNERSHIP COMPANY LTD
Example # 2	JOE'S BAR AND GRILL OF TAHOE, INCORPORATED
Example # 3	ABC PARTNERSHIP, LIMITED LIABILITY COMPANY
Example # 4	THE #1 BEST COMPANY IN AMERICA, HANDS DOWN!, INC.

**STEP #3 – Remove all period punctuation marks. Replace all remaining
characters outside of 0-9 and A-Z with single spaces**

This step serves to eliminate any special characters that may impact the
search criteria and the ultimate set of results. It is this step that removes
punctuation marks such as periods, exclamation points, dashes and
parentheses. This step will also remove any characters that are not part of
the basic ASCII character set, such as the US cent sign, the Euro sign,
accented characters and other characters not easily typed with one
keystroke on a keyboard.

This step treats periods differently than other punctuation marks. Due to
their common usage in abbreviations, removing period punctuation marks
altogether rather than replacing periods with a single space has been
shown to provide more sensible results than replacing periods with a
single space.

Entity Name Normalization Examples – After Step #3	
Example # 1	THE PARTNERSHIP COMPANY LTD
Example # 2	JOE S BAR AND GRILL OF TAHOE INCORPORATED
Example # 3	ABC PARTNERSHIP LIMITED LIABILITY COMPANY
Example # 4	THE 1 BEST COMPANY IN AMERICA HANDS DOWN INC

STEP #4 – Remove leading, trailing and multiple consecutive spaces

Spaces that occur prior to a name and spaces that appear after the visually perceivable characters of a name will be removed in this step. Multiple consecutive spaces in the interior of a debtor name will also be removed by this step. At the end of this step, the debtor name or search string will have been reduced to a word or set of words, separated at most by single spaces with no leading or trailing spaces around the word or set of words.

Entity Name Normalization Examples – After Step #4	
Example # 1	THE PARTNERSHIP COMPANY LTD
Example # 2	JOE S BAR AND GRILL OF TAHOE INCORPORATED
Example # 3	ABC PARTNERSHIP LIMITED LIABILITY COMPANY
Example # 4	THE 1 BEST COMPANY IN AMERICA HANDS DOWN INC

STEP #5 – Account for filing office treatment of "the" that prefaces the name based on current and past filing office practice

The word "THE" at the beginning of a business name may be treated differently than other words that begin a business name. The initial word "THE" may have been removed altogether, or it may have been moved to the end of the name. In either of these cases, the treatment of the initial word must be consistent for both index values and search criteria.

As mentioned earlier, if leading words such as "A" or "AN" have been treated in a similar way, they should also be accounted for in this step.

Entity Name Normalization Examples – After Step #5[3]	
Example # 1	PARTNERSHIP COMPANY LTD
Example # 2	JOE S BAR AND GRILL OF TAHOE INCORPORATED
Example # 3	ABC PARTNERSHIP LIMITED LIABILITY COMPANY
Example # 4	1 BEST COMPANY IN AMERICA HANDS DOWN INC

---

[3] For purposes of this illustration, assume that filing office practice for "THE" as a leading word on an entity is that it is removed prior to indexing.

STEP #6 – Remove "noise words" as established by the filing office of each state which occur at the end of the name. Create an iterative loop that removes the longest noise word phrase first then continue the loop for the number of times needed to remove all noise words.

> This step will remove any business entity name endings that may be required in the jurisdiction. The result of this step will be a normalized business name with any and all entries from a jurisdiction's noise word list having been removed.
>
> It is worth noting that evaluations of the ending words of an entity name prior to removing noise words should include a single space occurring prior to a potential noise word ending. This step should evaluate full word sequences for removal, not considering portions of words.
>
> One additional crucial element of this step is the iterative nature of the comparison and removal of noise words.

Entity Name Normalization Examples – After Step #6[4]	
Example # 1	PARTNERSHIP COMPANY (after all multi-word loop iterations and one (1) single-word loop iteration) PARTNERSHIP (after two (2) single-word loop iterations) "" (null value) (after three (3) single-word loop iterations)
Example # 2	JOE S BAR AND GRILL OF TAHOE (after all multi-word loop iterations and one (1) single-word loop iteration)
Example # 3	ABC PARTNERSHIP (after larger-than-three-word loop iterations and one (1) three-word loop iteration) ABC (after two-word loop iterations and one (1) single-word loop iteration)
Example # 4	1 BEST COMPANY IN AMERICA HANDS DOWN (after all multi-word loop iterations and one (1) single-word loop iteration)

STEP #7 – Remove all spaces from the entity name to concatenate and create the index value including a null value.

> As the final step of this normalization, any remaining spaces should be removed. It is possible, as a result of performing all of the steps in this procedure, that an entity name will have been reduced to a null value (i.e., all the words and characters of a name may have been removed). This is a known and acceptable result. For names that are reduced to a null value in such a way, they should be indexed and searchable by the null value.

---

[4] For purposes of this illustration, assume that "noise words" exist and are as processed in the "Sample Noise Word List" presented later in this guide.

Entity Name Normalization Examples – After Step #7	
Example # 1	"" (null value)
Example # 2	JOESBARANDGRILLOFTAHOE
Example # 3	ABC
Example # 4	1BESTCOMPANYINAMERICAHANDSDOWN[5]

Next, we consider the steps that should be performed in accordance with IACA recommendations on standard search logic for individual names. There are several individual names presented for the purpose of illustrating the application of the steps for normalization to individual names. The individual names prior to any processing are shown here:

Individual Name Normalization Examples – Original Names			
Example #	**Last Name**	**First Name**	**Middle Name**
Example #1	Cher		
Example #2	O'Neal	Scott	Jackson
Example #3	de la Hoya	Juan	Miguel
Example #4	James	Grant	Anderson

For Individual Names:

STEP #1 – Convert the character case to a standard case (upper or lower)

This step should provide for consistent results regardless of the character case of debtor names. If your software or database provides for native case-insensitive searching, this step may not apply.

Individual Name Normalization Examples – After Step #1[6]			
Example #	**Last Name**	**First Name**	**Middle Name**
Example #1	CHER		
Example #2	O'NEAL	SCOTT	JACKSON
Example #3	DE LA HOYA	JUAN	MIGUEL
Example #4	JAMES	GRANT	ANDERSON

---

[5] An interesting effect of the normalization is that an SSL search for "COMPANY" will not return this debtor name. The normalization of the search term (and debtor name indices evaluated as potential search results) "COMPANY" results in a null value, but does not match non-null names such as this. Therefore, a search for "COMPANY" will return results such as "THE CORPORATION COMPANY INC." and "PARTNERSHIP LIMITED" but not results such as "1BESTCOMPANYINAMERICAHANDSDOWN".

[6] For purposes of this document, assume that the standard character case used for every transformation and comparison is upper case.

STEP #2 – Replace all characters outside of 0-9 and A-Z with single spaces

This step serves to eliminate any special characters that may impact the search criteria and the ultimate set of results. It is this step that removes punctuation marks such as exclamation points, dashes, periods, parentheses. This step will also remove any characters that are not part of the basic ASCII character set, such as the US cent sign, the Euro sign, accented characters and other characters not easily typed with one keystroke on a keyboard.

Individual Name Normalization Examples – After Step #2			
Example #	**Last Name**	**First Name**	**Middle Name**
Example #1	CHER		
Example #2	O NEAL	SCOTT	JACKSON
Example #3	DE LA HOYA	JUAN	MIGUEL
Example #4	JAMES	GRANT	ANDERSON

STEP #3 – Remove leading, trailing and interior spaces from each name element

Spaces that occur prior to a name, after a name, and within a name will be removed in this step. At the conclusion of this step, all spaces will have been removed from every name element.

Individual Name Normalization Examples – After Step #3			
Example #	**Last Name**	**First Name**	**Middle Name**
Example #1	CHER		
Example #2	ONEAL	SCOTT	JACKSON
Example #3	DELAHOYA	JUAN	MIGUEL
Example #4	JAMES	GRANT	ANDERSON

STEP #4 – Create multiple search keys or search criteria

This step will provide the elements of an individual name to be used when performing searches. The manner in which searches of names of individuals named in financing statements are conducted is substantially different than the way that searches of entity names are performed, as indicated in Model Administrative Rule 503.7. As stated in that rule, searches are to be performed of indexed names based on the number of name elements provided by the searcher to be searched. There are essentially six different types of searches that may be performed based on the search criteria provided by the user. These different types of searches that may be performed are:

- Type 1 – Last name, full first name, full middle name OR last name only;
- Type 2 – Last name, full first name, middle initial;
- Type 3 – Last name, first initial, full middle name;
- Type 4 – Last name, first initial, middle initial;
- Type 5 – Last name, full first name; and,
- Type 6 – Last name, first initial.

The elements of an individual name should be normalized as indicated above prior to being indexed and stored for searching, and search strings should be treated in the same fashion prior to searches being executed.

An Example of the Iterative Process of SSL

The following was presented in August 2005 at the STS-ITS UCC Summit meeting in San Antonio, Texas, USA. It provides an example of the application of the normalization steps to one specific name.

Entity Debtor Name: ABC Associates & Partners Company, LLC

1st Step – Convert the character case to a standard case (upper or lower)[7]

      Result: ABC ASSOCIATES & PARTNERS COMPANY, LLC

2nd Step – Convert "&" to "AND"

      Result: ABC ASSOCIATES AND PARTNERS COMPANY, LLC

3rd Step – Replace all characters outside of 0-9 and A-Z with single spaces[8]

      Result: ABC ASSOCIATES AND PARTNERS COMPANY LLC

4th Step – Remove leading, trailing and multiple consecutive spaces

      Result: ABC ASSOCIATES AND PARTNERS COMPANY LLC

5th Step – Account for filing office treatment of "THE" that prefaces the name

      Result: ABC ASSOCIATES AND PARTNERS COMPANY LLC

6th Step – Remove "noise words" as established by the filing office of each state which occur at the end of the name.

      Process to remove noise words:
1. Order your ending noise words from longest phrase to shortest phrase[9]
2. Start the loop from the longest to shortest, repeating as many times as needed to process and remove any matching noise words.

      Result after all multi-word loop iterations and one single-word loop iteration:
            ABC ASSOCIATES AND PARTNERS COMPANY

---

[7] For purposes of this example, assume that the standard in use is that debtor names are converted to all uppercase.

[8] Although it may not be visually perceivable, there are two spaces between "COMPANY" and "LLC", the comma having been replaced by a single space.

[9] "Longest to shortest" is measured by the number of elements of a noise word ending. Using this definition, a hypothetical noise word ending of "L L L P' is actually longer than and is evaluated prior to "Limited Liability Company", since the former consists of four elements separated by spaces, while the latter consists of three elements separated by spaces.

Result after two single-word loop iterations:

ABC ASSOCIATES AND PARTNERS

Result after three single-word loop iterations:

ABC ASSOCIATES AND

Result after four single-word loop iterations (no changes from prior result, so the loop ends):

ABC ASSOCIATES AND

7th Step – Remove all spaces from the entity name to concatenate and create the index value including a null value

Result: ABCASSOCIATESAND

Sample Noise Word List

The following was presented in February 2001 and adopted by IACA as the official list of ending noise words just prior to enactment of Revised Article 9. The document is presented here as it was presented at that time.

IACA List of Ending Noise Words pursuant to Rule 503.4

The following words and abbreviations indicate the existence or nature of an entity. These business endings will be ignored in a UCC search.

- Agency
- Association
- Assn
- Associates
- Assc
- Assoc
- Attorneys at Law
- Bank
- National Bank
- Business Trust
- Charter
- Chartered
- Company
- Co
- Corporation
- Corp
- Credit Union
- CU
- Federal Savings Bank
- FSB
- General Partnership
- Gen part
- GP
- Incorporated
- Inc
- Limited
- Ltd
- Ltee
- Limited Liability Company
- LC
- LLC
- Limited Liability Partnership
- LLP
- Limited Partnership
- LP
- Medical Doctors Professional Association
- MDPA
- Medical Doctors Professional Corporation
- MDPC
- National Association
- NA
- Partners
- Partnership
- Professional Association
- Prof Assn
- PA
- Professional Corporation
- Prof Corp
- PC
- Professional Limited Liability Company
- Professional Limited Liability Co
- PLLC
- Railroad
- RR
- Real Estate Investment Trust
- REIT
- Registered Limited Liability Partnership
- RLLP
- Savings Association
- SA
- Service Corporation
- SC
- Sole Proprietorship
- SP
- SPA
- Trust
- Trustee
- As Trustee

Appendix A – Web Service Source Code from Ose Micro

This source code provided courtesy of Thomas Ose, Ose Micro Solutions, Inc.

```
// ====================================================================
// IACASearchKeyWS - this web service exposes the functionality of
//                   the IACASearchKeyInterface.
// --------------------------------------------------------------------
//             Copyright (c) 2005 Ose Micro Solutions, Inc.
//         This code has been released for use by the IACA states
// --------------------------------------------------------------------
//  [08/23/2005]  tmo 01.00 started tracking changes
// ====================================================================
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;

namespace IACASearchKey
{
        /// <summary>
        /// Summary description for Service1.
        /// </summary>
        [WebService(Namespace="http://localhost/")]
  public class IACASearchKeyWS : System.Web.Services.WebService
        {
    /// <summary>
    /// Class constructor for the web service
    /// </summary>
                public IACASearchKeyWS()
                {
                        InitializeComponent();
                }

                #region Component Designer generated code

                //Required by the Web Services Designer
                private IContainer components = null;

                /// <summary>
                /// Required method for Designer support - do not modify
                /// the contents of this method with the code editor.
                /// </summary>
                private void InitializeComponent()
                {
                }

                /// <summary>
                /// Clean up any resources being used.
                /// </summary>
                protected override void Dispose( bool disposing )
                {
                        if(disposing && components != null)
                        {
                                components.Dispose();
                        }
                        base.Dispose(disposing);
                }

                #endregion

    /// <summary>
```

```
    /// This method is used to call the interface to create
    /// the corporate search key based on the submitted string.
    /// </summary>
    /// <param name="inString">the string to use to build the key</param>
    /// <returns>the generated search key</returns>
                [WebMethod]
                public string BuildCorpSearchKey(string inString)
                {
      try                                                        // do
some minimal error catching
      {
        IACASearchKeyInterface myInterface = new IACASearchKeyInterface();     // create
a new instance of the class
        return myInterface.buildCorpKey(inString);                         // call
the buildKey method and return the results
      }
      catch (System.Exception e)
      {
        throw(e);
      }
                }// --- end of BuildCorpSearchKey()

    /// <summary>
    /// This method is used to call the interface to create
    /// the individual search keys based on the submitted string.
    /// </summary>
    /// <param name="FirstName">First name part of the individuals name</param>
    /// <param name="MiddleName">Middle name part of the individuals name</param>
    /// <param name="LastName">Last name part of the individuals name</param>
    /// <returns>an array containing all the generated search key</returns>
    [WebMethod]
    public ArrayList BuildIndividualSearchKey(string FirstName, string MiddleName, string
LastName)
    {
      try                                                        // do
some minimal error catching
      {
        IACASearchKeyInterface myInterface = new IACASearchKeyInterface();     // create
a new instance of the class
        return myInterface.buildIndividualKey(FirstName, MiddleName, LastName); // call
the buildKey method and return the results
      }
      catch (System.Exception e)
      {
        throw(e);
      }
    }// --- end of BuildIndividualSearchKey()
  }
}


// ====================================================================
// IACASearchKeyInterface - this class contains all the methods to
//                          generate a search keep as prescribed by
//                          the IACA model rules.
// --------------------------------------------------------------------
//            Copyright (c) 2005 Ose Micro Solutions, Inc.
//        This code has been released for use by the IACA states
// --------------------------------------------------------------------
//  [08/23/2005]  tmo 01.00 started tracking changes
//  [08/24/2005]  tmo 01.01 added noise word removal
//     [09/10/2005]   tmo    01.02 added additional documentation and NDoc file
//     [10/03/2005]   tmo    01.03 added logic to remove the leading THE that was left
out
// ====================================================================
using System;
using System.Collections;
```

```csharp
namespace IACASearchKey
{
  /// <summary>
  /// Summary description for IACASearchKeyInterface.
  /// </summary>
  public class IACASearchKeyInterface
  {
    /// <summary>
    /// Character array that holds the special characters
    /// </summary>
    private char [] specialChars;

    /// <summary>
    /// String array containing the noise words to remove
    /// </summary>
    private string [] noiseWords;

    /// <summary>
    /// This method is used to uppercase the submitted string
    /// </summary>
    /// <param name="inString">string to work on</param>
    /// <returns>string in upper case</returns>
    private string upperCaseString(string inString)
    {
      string returnString = inString.ToUpper();                        //
convert all characters to upper case
      return returnString;                                            // return
the results
    } //--- end of upperCaseString()

    /// <summary>
    /// This method is used to replace the &amp; (ampersand) with the word "and".
    /// It deals with the possibility that there may be spaces along
    /// with the character.
    /// <para>
    /// The IACA model only deals with the &amp; by itself and not
    /// surrounding spaces.
    /// </para>
    /// </summary>
    /// <param name="inString">the string to check</param>
    /// <returns>the results</returns>
    private string removeAmpersand(string inString)
    {
      string returnString = inString;                                  // set
the default return value to be what was sent in
      if(inString.IndexOf("&") >= 0)                                  // if we
have an ampersand then remove all variations
      {
        returnString = returnString.Replace(" & ", "AND");            //
surrounded by spaces
        returnString = returnString.Replace("& ", "AND");             //
trailing space
        returnString = returnString.Replace(" &", "AND");             //
leading space
        returnString = returnString.Replace("&", "AND");              // no
spaces
      }
      return returnString;                                            // return
the result
    } //--- end of removeAmperand()

    /// <summary>
    /// This method is used to replace the period with a null character
    /// </summary>
    /// <param name="inString">the string to manipulate</param>
    /// <returns>the results</returns>
    private string removePeriod(string inString)
    {
      string returnString = inString;                                  // set
the default return value to be what was sent in
```

```
        if(inString.IndexOf(".") >= 0)                                          // if we
have a period
        {
          returnString = returnString.Replace(".", "");                         //
replace the period with a null
        }
        return returnString;                                                    // return
the result
    } //--- end of removePeriod()

    /// <summary>
    /// This method is used to remove the leading and trailing spaces
    /// <para>
    /// This function does not reduce the number of spaces as suggested in the
    /// IACA recommendation since the trailing spaces are removed prior to testing
    /// for another noise words.  Also the noise words expect to start with a space.
    /// </para>
    /// </summary>
    /// <param name="inString">string to manipulate</param>
    /// <returns>the results</returns>
    private string removeSpaces(string inString)
    {
      string returnString = inString;                                           // set
the default return value to be what was sent in
      returnString = returnString.Trim();                                       // remove
leading and trailing spaces
      // TODO: need to add multi-space condensing
      return returnString;                                                      // return
the result
    } //--- end of removeSpaces()

    /// <summary>
    /// This method is used to strip all spaces
    /// </summary>
    /// <param name="inString">string to manipulate</param>
    /// <returns>the results</returns>
    private string stripSpaces(string inString)
    {
      string returnString = inString;                                           // set
the default return value to be what was sent in
        if(inString.IndexOf(" ") >= 0)                                          // if we
have a space
        {
          returnString = returnString.Replace(" ", "");                         //
replace all spaces with a null
        }
        return returnString;                                                    // return
the result
    } //--- end of stripSpaces()

    /// <summary>
    /// This method is used to remove the special characters
    /// from the search string and replace them with spaces
    /// <para>
    /// The special characters are defined in the specialChars array.
    /// </para>
    /// </summary>
    /// <param name="inString">string to manipulate</param>
    /// <returns>the results</returns>
    private string removeSpecialCharacters(string inString)
    {
      string returnString = inString;                                           // set
the default return value to be what was sent in
      foreach (char myChar in this.specialChars)
      {
        returnString = returnString.Replace(myChar.ToString(), " ");
      }
      return returnString;                                                      // return
the result
    } //--- end of removeSpecialCharacters()
```

```
/// <summary>
/// This method reiteratively removes the noise word endings from
/// the submitted string.  It uses the noiseWords array to determine
/// what is a noise word.
/// <para>
/// The noiseWords array is sorted so that longer words are removed first.
/// A word is not just a single word but can combine a number of words to
/// identify a noise phrase.
/// </para>
/// <para>
/// Since the periods are replaced with nulls instead of spaces we do
/// not need the additional spaced versions of the acronyms.
/// </para>
/// </summary>
/// <param name="inString">string to manipulate</param>
/// <returns>the result</returns>
private string removeNoiseWords(string inString)
{
   string returnString = inString;                                  // set
the default return string
   int startPos = 0;                                                // set
the start position of the noise word ending
   int wordPointer = 0;                                             //
pointer into the noise word list
   bool allDone = false;                                            // set
true if we find no more noise words or there is nothing left
   bool doMoreWords = true;                                         // set
false when we found a match

   while(!allDone)                                                  //
multiple reiterations
   {
      wordPointer = 0;                                              // the
pointer tot the beginning
      doMoreWords = true;                                           // reset
the match indicator
      while (doMoreWords)                                           // walk
through the list of noise words
      {
         if(returnString.EndsWith(" " + this.noiseWords[wordPointer]))       // if the
string ends with the noise word added starting space
         {
            startPos = returnString.LastIndexOf(this.noiseWords[wordPointer]); // get
the starting position of the noise word
            returnString = returnString.Substring(0,startPos);              // grab
what ever is left
            returnString = returnString.Trim();                             // remove
any leading or trailing spaces
            doMoreWords = false;                                            //
indicate that we are done with this iteration
         }
         else                                                              // if the
string does not end with the noise word
         {
            wordPointer += 1;                                              //
increment the word pointer
            if(wordPointer == this.noiseWords.Length )                     // if we
looped through the list with out doing anything
            {
               doMoreWords = false;                                        // we are
done looking for matches
               allDone = true;                                            // we are
done looking for noise words
            }
         }
      }
   }
   return returnString;                                             // return
the result
} //--- end of removeNoiseWords()
```

```
    /// <summary>
    /// This method is used to normalize the individual name parts.
    /// <para>
    /// It will uppercase the string, remove all special characters as well as spaces
from the string
    /// </para>
    /// </summary>
    /// <param name="inString">the string to normalize</param>
    /// <returns>normalized sring</returns>
    private string normalizeName(string inString)
    {
      string outName = inString;                                      // get
the string to use for creation
      outName = this.upperCaseString(outName);                       //
uppercase the in bound string
      outName = this.removePeriod(outName);
      outName = this.removeSpecialCharacters(outName);               // remove
any special characters
      outName = this.stripSpaces(outName);                          // strip
all the spaces
      return outName;                                               // return
the results
    } //--- end of processName()


    /// <summary>
    /// This method will remove the leading THE. it will also remove the trailing space
    /// behind the THE.
    /// </summary>
    /// <param name="inString">string to work on</param>
    /// <returns>result</returns>
    public string removeLeadingThe(string inString)
    {
      string outName = inString;                                      // get
the string to use for creation
      if (inString.StartsWith("THE"))                               // if it
starts with THE and a space
      {
        outName = inString.Substring(3,(inString.Length - 3));       // remove
it
      }
      return outName;                                               // return
the results
    } //--- end of removeLeadingThe()

    //=================================================================
    //    Public Methods and attributes
    //=================================================================
    /// <summary>
    /// Class constructor
    /// </summary>
    public IACASearchKeyInterface()
    {
      //  build the list of special characters
      this.specialChars = new char [] { '!',
                                        '@',
                                        '#',
                                        '$',
                                        '%',
                                        '^',
                                        '*',
                                        '(',
                                        ')',
                                        '-',
                                        '_',
                                        '=',
                                        '+',
                                        '[',
                                        ']',
                                        '{',
                                        '}',
```

```
                                        ';',
                                        ':',
                                        ',',
                                        '<',
                                        '>',
                                        '/',
                                        '?',
                                        '`',
                                        '~',
                                        (char)34,
                                        (char)39
                                    };

        // build the list of noise words - these are ordered by string length so that the
longest appears first
        this.noiseWords = new string [] { "REGISTERED LIMITED LIABILITY PARTNERSHIP",
                                        "MEDICAL DOCTORS PROFESSIONAL ASSOCIATION",
                                        "MEDICAL DOCTORS PROFESSIONAL CORPORATION",
                                        "PROFESSIONAL LIMITED LIABILITY COMPANY",
                                        "PROFESSIONAL LIMITED LIABILITY CO",
                                        "LIMITED LIABILITY PARTNERSHIP",
                                        "REAL ESTATE INVESTMENT TRUST",
                                        "LIMITED LIABILITY COMPANY",
                                        "PROFESSIONAL ASSOCIATION",
                                        "PROFESSIONAL CORPORATION",
                                        "NATIONAL ASSOCIATION",
                                        "FEDERAL SAVINGS BANK",
                                        "LIMITED PARTNERSHIP",
                                        "GENERAL PARTNERSHIP",
                                        "SAVINGS ASSOCIATION",
                                        "SERVICE CORPORATION",
                                        "SOLE PROPRIETORSHIP",
                                        "ATTORNEYS AT LAW",
                                        "BUSINESS TRUST",
                                        "NATIONAL BANK",
                                        "CREDIT UNION",
                                        "INCORPORATED",
                                        "ASSOCIATION",
                                        "CORPORATION",
                                        "PARTNERSHIP",
                                        "AS TRUSTEE",
                                        "ASSOCIATES",
                                        "PROF ASSN",
                                        "PROF CORP",
                                        "CHARTERED",
                                        "GEN PART",
                                        "PARTNERS",
                                        "RAILROAD",
                                        "CHARTER",
                                        "COMPANY",
                                        "LIMITED",
                                        "TRUSTEE",
                                        "AGENCY",
                                        "ASSOC",
                                        "TRUST",
                                        "ASSN",
                                        "ASSC",
                                        "BANK",
                                        "CORP",
                                        "LTEE",
                                        "MDPA",
                                        "MDPC",
                                        "PLLC",
                                        "REIT",
                                        "RLLP",
                                        "FSB",
                                        "INC",
                                        "LLC",
                                        "LTD",
                                        "LLP",
                                        "SPA",
```

```csharp
                                    "CO",
                                    "CU",
                                    "GP",
                                    "LC",
                                    "LP",
                                    "NA",
                                    "PA",
                                    "PC",
                                    "RR",
                                    "SA",
                                    "SC",
                                    "SP"
                                 };
    } //--- end of IACASearchKeyInterface()


    /// <summary>
    /// This method that used to call
    /// the private methods to produce the returned search key
    /// </summary>
    /// <param name="inString">the name string to use to create the search key</param>
    /// <returns>the generated search key</returns>
    public string buildCorpKey(string inString)
    {
      string outKey = inString;                                         // get
the string to use for creation
      outKey = this.upperCaseString(outKey);                            //
uppercase the in bound string
      outKey = this.removeAmpersand(outKey);                            // remove
the ampersand
      outKey = this.removePeriod(outKey);                               // deal
with the periods
      outKey = this.removeSpecialCharacters(outKey);                    // remove
any special characters
      outKey = this.removeSpaces(outKey);                               // remove
any leading or trailing spaces
      outKey = this.removeNoiseWords(outKey);                           // remove
the noise words
      outKey = this.removeLeadingThe(outKey);                           // remove
the leading THE and space
      outKey = this.stripSpaces(outKey);                                //
finally remove all spaces
      return outKey;                                                    // return
the results
    } //--- end of buildKey()

    /// <summary>
    /// This public method is used to build the keys for an
    /// individual name.
    /// <para>
    /// The suffix for an individual name is ignored and not
    /// used as part of the created indexes
    /// </para>
    /// <para>
    /// The resulting Array list will contain the 6 IACA specified name index
variations<BR/>
    /// </para>
    /// <list type="number">
    ///    <item>Full Last Name | Full First Name | Full Middle Name</item>
    ///    <item>Full Last Name | Full First Name | Middle Initial </item>
    ///    <item>Full Last Name | First Initial | Full Middle Name </item>
    ///    <item>Full Last Name | First Initial | Middle Initial </item>
    ///    <item>Full Last Name | Full First Name</item>
    ///    <item>Full Last Name | First Initial</item>
    /// </list>
    /// </summary>
    /// <param name="fName">First name part of the individuals name</param>
    /// <param name="mName">Middle name part of the individuals name</param>
    /// <param name="lName">Last name part of the individuals name</param>
    /// <returns></returns>
    public ArrayList buildIndividualKey(string fName, string mName, string lName)
```

```
    {
        ArrayList outKey = new ArrayList();                                    // get
the string to use for creation
        string strFName = normalizeName(fName);                               // get
the normalized first name
        string strMName = normalizeName(mName);                               // get
the normalized middle name
        string strLName = normalizeName(lName);                               // get
the normalized last name

        outKey.Add(strLName.Trim() + strFName.Trim() + strMName.Trim());
        outKey.Add(strLName.Trim() + strFName.Trim() + ((strMName.Length > 0) ?
strMName.Substring(0,1) : ""));
        outKey.Add(strLName.Trim() + ((strFName.Length > 0) ? strFName.Substring(0,1) : "")
+ strMName.Trim());
        outKey.Add(strLName.Trim() + ((strFName.Length > 0) ? strFName.Substring(0,1) : "")
+ ((strMName.Length > 0) ? strMName.Substring(0,1) : ""));
        outKey.Add(strLName.Trim() + strFName.Trim());
        outKey.Add(strLName.Trim() + ((strFName.Length > 0) ? strFName.Substring(0,1) :
""));
        return outKey;                                                         // return
the results
    } //--- end of buildIndividualKey()

        }//--- end of class
}//--- end of file
```