

## AddressDoctor Software Library

Product Documentation

Version 5.2.7

AddressDoctor GmbH

**Roentgenstr. 9**  
**67133 Maxdorf**  
**Germany**

+49 (6237) 9774 0

USA

208 S WILMINGTON ST STE 200  
Raleigh NC 27601-1434

United States

+1 (866) 402 2800

UK FreeCall

0800-0328-276

France Numéro Vert

0800 917113

India FreeCall

000800 1003486

Singapore FreeCall

800 1301756

**info@AddressDoctor.com**  
**www.AddressDoctor.com**

Released: July 7, 2011

## Foreword

This documentation explains features and functions of the AddressDoctor software library for postal address validation. You have selected a leading data quality product allowing you to easily ensure superb address quality for postal addresses from all around the world.

This documentation is meant to cater to the information needs of beginners and advanced users alike. It covers all platforms supported by the AddressDoctor software library and all available interfaces. The Introduction chapter (2) provides a general overview of the AddressDoctor components and concepts and looks at them from a business perspective. While chapter 3 describes the installation process, chapter 4 should help you get started right away, both when you are new to AddressDoctor or migrating from the previous version. The Concepts chapter (5) helps you understand important features of the AddressDoctor library. We recommend this chapter for all user groups. Advanced users will find chapter 6 “How do I...” helpful. It provides sample code for common tasks.

This PDF document provides embedded bookmarks for fast access to document chapters, please open the bookmark view of your PDF viewer in case it was not opened by default. Additionally, all chapter number references in the text provide hyperlink access to the reference targets.

Please do always check the release note section included with the API documentation, see chapter 10.2. The AddressDoctor documentation is a work in progress and we always appreciate user feedback to improve this document. Please email your suggestions and comments to [Documentation@AddressDoctor.com](mailto:Documentation@AddressDoctor.com).

## Contents

Foreword.....	2
Contents.....	3
1 Document Conventions.....	8
2 Introduction.....	9
2.1 Functional Overview.....	10
2.1.1 Character Set Mapping and Transliteration.....	10
2.1.2 Address Parsing, Formatting and Standardization.....	11
2.1.3 Global Address Validation.....	12
2.1.4 The AddressDoctor Software Library in a Nutshell.....	12
2.2 Supported Platforms.....	13
2.3 System Requirements.....	13
3 Setup.....	15
3.1 General Remarks.....	15
3.1.1 Software packages.....	15
3.1.2 Documentation package.....	16
3.1.3 Postal reference data packages.....	16
3.2 Installing the Library Files.....	16
3.2.1 C Installation.....	16
3.2.2 Java Installation.....	17
3.3 Installing the Reference Databases.....	18
3.3.1 Installation Notices.....	19
3.3.2 Special Remarks for USA CASS Certified Mode.....	19
3.3.3 Special remarks for Canadian SERP Certified Mode.....	20
3.3.4 Special Remarks for Australian AMAS Certified Mode.....	20
4 Quick Start Guide.....	21
4.1 First-Time Use of AddressDoctor 5.....	21
4.2 Migration from Version 4.....	26
4.2.1 New in Version 5.....	26
4.2.2 Key API Changes.....	26
4.2.3 Setting up the AddressDoctor Engine.....	27
4.2.4 AddressObjects.....	27
4.2.5 Direct API.....	28
4.2.6 Transliteration (formerly UniString Object).....	31
4.2.7 Unlock Code Mechanism.....	31
5 Concepts.....	32
5.1 Character Set Mapping.....	32
5.2 Transliteration.....	33
5.3 Address Element Abstraction.....	34
5.4 Address Parsing.....	36
5.5 Address Validation.....	37

5.6	The AddressDoctor Engine .....	38
5.7	AddressObjects .....	39
5.8	Input and Output Encoding .....	40
5.9	AddressElement Items and AddressLines .....	40
5.10	Address Item Types .....	42
5.11	Process Modes .....	43
5.11.1	Batch .....	44
5.11.2	Interactive .....	44
5.11.3	Fast Completion .....	44
5.11.4	Certified .....	45
5.11.5	Parse Only .....	45
5.11.6	Country Recognition .....	45
5.12	Process Parameters .....	46
5.12.1	The PreferredScript Parameter .....	46
5.12.2	The PreferredLanguage Parameter .....	46
5.12.3	The ForceCountryISO3 and DefaultCountryISO3 Parameters .....	47
5.12.4	The CountryType and CountryofOriginISO3 Parameters .....	47
5.12.5	The MatchingAlternatives and MatchingScope Parameters .....	47
5.13	Output Formatting .....	48
5.14	Output Standardization .....	48
5.15	Process Status Values .....	50
5.16	Mailability Scores .....	52
5.17	Geocoding Status Values .....	53
5.18	CASS Status Values .....	54
5.19	SERP Status Values .....	54
5.20	SNA Status Values .....	54
5.21	AMAS Status Values .....	54
5.22	Country Specific Enrichment .....	55
5.22.1	Country Specific Enrichment Status Values .....	55
5.22.2	Country Specific Enrichment Output fields .....	55
5.23	Element Status and Relevance Values .....	57
5.23.1	ElementInputStatus .....	58
5.23.2	ElementResultStatus .....	58
5.23.3	ElementRelevance .....	59
5.24	ResultPercentage Values .....	59
5.25	Return Codes .....	60
5.26	OptimizationLevel .....	62
5.27	Preloading .....	63
5.28	Caching .....	65
5.29	Multithreading .....	65
5.30	Memory Management .....	67
6	How do I .....	69
6.1	...initialize AddressDoctor? .....	69

6.2	...determine the AddressDoctor version? .....	71
6.3	...specify processing or input parameters and a result format? .....	71
6.4	...handle unlock codes? .....	74
6.5	...configure reference databases? .....	75
6.6	...determine the current engine settings? .....	76
6.7	...assign an address to the AddressObject? .....	76
6.7.1	General Overview .....	77
6.7.2	Fielded address input .....	78
6.7.3	Partially fielded address input.....	80
6.7.4	Unfielded address input .....	80
6.8	...validate an address? .....	81
6.9	...parse an address?.....	82
6.10	...check the process mode? .....	82
6.11	...retrieve a suggested correction? .....	82
6.12	...retrieve the result status and additional information? .....	84
6.13	...retrieve address enrichments? .....	85
6.14	...analyze error conditions? .....	85
6.15	...assign and process addresses in non-latin script? .....	86
6.16	...use AddressDoctor with multiple processor cores? .....	89
6.17	...produce valid AddressDoctor XML? .....	89
6.18	...use AddressDoctor XML for flexible Business Processes?.....	89
6.19	...use AddressDoctor for Master Data Management? .....	90
6.20	...use AddressDoctor in an eBusiness Environment? .....	91
6.21	...use the Quick Address Entry Feature?.....	91
6.22	...use AddressDoctor in a multi-tenant hosted environment?.....	92
6.23	...use AddressDoctor for Web Services?.....	93
6.24	...validate an address in CERTIFIED mode? .....	94
6.24.1	...process an address following the rules for CASS certification? .....	94
6.24.2	...process an address following the rules for SERP certification? .....	97
6.24.3	...process an address following the rules for AMAS certification? .....	98
6.24.4	...process an address following the rules for SNA certification? .....	99
6.25	...optimize performance? .....	99
7	Demo Applications .....	103
7.1	AddressDoctor 5 Console Demo .....	103
7.2	AddressCheck (Windows only).....	104
8	Sample Address Data for Testing .....	106
8.1	Addresses with Status Code Vx.....	106
8.1.1	Correct Address .....	106
8.1.2	Address with Exonym replaced.....	106
8.2	Addresses with Status Code Cx.....	107
8.2.1	Address with missing Postal Code .....	107
8.2.2	Address with Misspellings in Street and City Name .....	108
9	Miscellaneous Topics .....	109

9.1	Background on the (Postal) Reference Database .....	109
9.1.1	Database Format.....	110
9.1.2	Database Size .....	111
9.1.3	Database Updates.....	111
9.2	Postal Certifications .....	111
9.3	Support Information .....	112
9.4	Recommended Database Layout for International Addresses.....	112
10	Appendix .....	115
10.1	API Document Type Definitions.....	115
10.2	API Reference .....	115
10.3	Schematic AddressDoctor Processing Flow .....	116
10.4	AddressElement Output Examples.....	117
10.4.1	Argentina.....	117
10.4.2	Australia .....	117
10.4.3	Austria .....	118
10.4.4	Belarus.....	118
10.4.5	Belgium .....	118
10.4.6	Bulgaria .....	119
10.4.7	Canada.....	119
10.4.8	Chile .....	119
10.4.9	China.....	120
10.4.10	Croatia.....	120
10.4.11	Cyprus .....	120
10.4.12	Czech Republic .....	121
10.4.13	Denmark.....	121
10.4.14	Ecuador .....	121
10.4.15	Estonia.....	122
10.4.16	Finland.....	122
10.4.17	Germany.....	122
10.4.18	Greece.....	123
10.4.19	Guatemala.....	123
10.4.20	Hong Kong .....	123
10.4.21	Hungary.....	124
10.4.22	India .....	124
10.4.23	Italy.....	124
10.4.24	Japan .....	125
10.4.25	Jordan.....	125
10.4.26	Luxembourg .....	125
10.4.27	Madagascar .....	126
10.4.28	Malaysia.....	126
10.4.29	Mexico.....	126
10.4.30	Netherlands .....	127
10.4.31	New Zealand.....	127

10.4.32	Norway.....	127
10.4.33	Pakistan.....	128
10.4.34	Panama.....	128
10.4.35	Philippines.....	128
10.4.36	Poland.....	129
10.4.37	Portugal.....	129
10.4.38	Russia.....	129
10.4.39	Singapore.....	130
10.4.40	Slovakia.....	130
10.4.41	Slovenia.....	130
10.4.42	South Africa.....	131
10.4.43	Sweden.....	131
10.4.44	Switzerland.....	131
10.4.45	Turkey.....	132
10.4.46	Ukraine.....	132
10.4.47	United Arab Emirates.....	132
10.4.48	United Kingdom.....	133
10.4.49	USA.....	133
10.4.50	Venezuela.....	133
10.5	Province Output.....	133
10.6	Reference Data Copyright Notices.....	139
11	Glossary.....	141
	ISO Country Codes.....	141
	Normalization.....	141
	Parsing.....	141
	Romanization.....	141
	Standardization.....	141
	Tokenization.....	142
	Transformation.....	142
	Transcription / Transliteration.....	142
	Unified Ideographs.....	142
	Validation.....	142
12	Index.....	143

## 1 Document Conventions

This document uses icons in the margin to indicate if an explanation is specific to a certain version of the interface. While the functionality of the API is the same for all interfaces, different syntax may be required.



Applies to the C interface (C wrapper) on all platforms



Applies to the Java interface (Java wrapper) on all platforms

Sample program code may be identified by its fixed space typeface, e.g.:

```
For i = 1 to 5
  j = i + 1
Next
```

## 2 Introduction

AddressDoctor provides a powerful software library with functions to enhance and ensure postal address data quality. With a world population of more than 6.5 billion people and increasingly global trade relationships, more and more people face the challenges of handling addresses from all around the world.

At the same time, taking care of customer relationships is more important than ever - especially in today's rushed world economy. We receive letters generated by computers, talk to computers on the telephone and we check-out in supermarkets by ourselves. Data, once in a computer system, is often considered to be correct. In many cases, it serves as the foundation for numerous business processes. Rarely ever is the data in the system questioned. This can lead to dangerous situations, as we could all see in the movie "The Net" where a young woman loses her identity because of deleted data.

The following example should illustrate the situation:

**Data input by hotel staff**

Sven Schreiber  
 Fursloog Str. 1  
 DE-67134 Berlich

**Sheraton Heathrow HOTEL**

/1 SPG Number:  
 Car Registration:  
 Passport Number: 2167428806  
 Country of Issue:  
 Next Destination:

/PAO  
 1

Sheraton Heathrow Hotel, London, 05.03.0

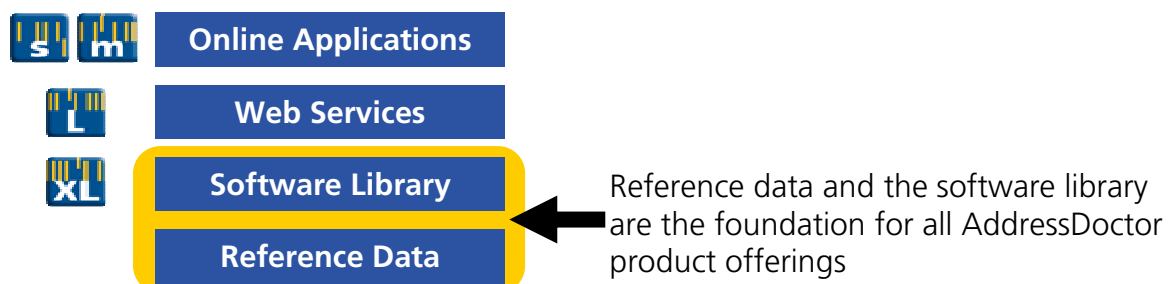
Your attention should be drawn to the Hotel Proprietor's Act of 1965 displayed in the front hall. Safe deposits are available at a  
 Regardless of charge instructions, the signed guest below acknowledges the account as a personal debt and agrees to accept,  
 in the event that the indicated person, company or association fails to pay for any part of, or the full amount of the charges.

**Correct address needed for delivery**

**Sven Schreiber  
 Feuerbergstr. 1  
 67134 Birkenheide  
 Germany**

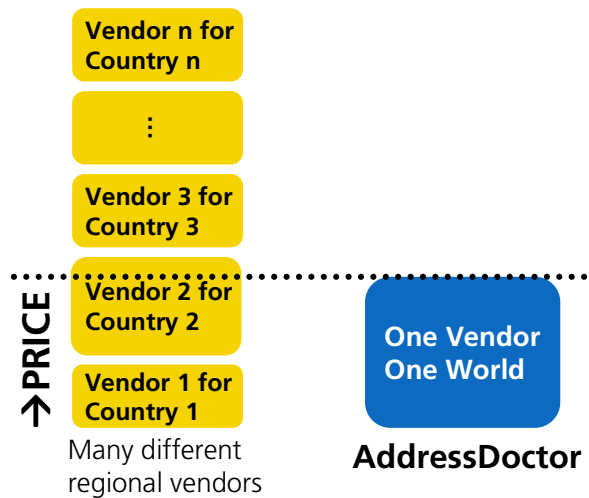
The AddressDoctor product line was introduced in 1994 by Platon Data Technology GmbH (now AddressDoctor GmbH), a German software company that has become the innovation leader in data quality tools for postal addresses. From the very beginning, AddressDoctor has specialized in international addresses.

Here's an overview of the whole AddressDoctor product line:



With the global launch of the AddressDoctor software library, AddressDoctor has set new standards in terms of flexibility, ease of use and processing power. Covering more than 240 countries and territories, the software encompasses knowledge about postal addresses from virtually any place mankind has populated.

The global approach of AddressDoctor is a direct and major benefit to its customers:



- Cost savings in:
  - Contract Management
  - Integration
  - Deployment
  - Licenses
  - Support & Maintenance
  - ...
- Cross-country synergy effects compared to a multi-vendor approach

## 2.1 Functional Overview

The AddressDoctor software library features several stages of address processing, namely Transliteration, Parsing, Validation and Formatting, which interact with each other.

### 2.1.1 Character Set Mapping and Transliteration

AddressDoctor incorporates functionality to handle international strings and their complexities. It uses fully Unicode enabled string processing which enables the transliteration of non-roman characters into the Latin character set and mapping between different character sets.

- Storing data in and mapping between over 30 different character sets including UTF-8, ISO 8859-1, GBK, BIG5, JIS, EBCDIC
- Proper “elimination” of diacritics according to language rules
- Transliteration for various alphabets into Latin Script:
  - Greek (BGN/PCGN 1962, ISO 843 – 1997)
  - Cyrillic (BGN/PCGN 1947, ISO 9 – 1995)
  - Hebrew
  - Japanese Katakana, Hiragana and Kanji
  - Chinese Pinyin (Mandarin, Cantonese)
  - Korean Hangul

For example:

**ΑΘΗΝΑΣ 63**  
**105 52 ΑΘΗΝΑ**  
**GREECE**



**ATHINAS 63**  
**105 52 ATHINA**  
**GREECE**

Data input is in a foreign alphabet

Transliterated data is in Latin script

## 2.1.2 Address Parsing, Formatting and Standardization

Restructuring incorrectly fielded address data is a complex and difficult task especially when done for international addresses. People introduce many ambiguities as they enter address data into computer systems. Among the problems are misplaced elements (such as company or personal names in street address fields) or varying abbreviations that are not only language, but also country specific.

The AddressDoctor Parser component identifies address elements in totally unfielded or partially fielded addresses and assigns them to the proper fields. This is an important precursor to the actual validation. Without restructuring, “no match” situations might result.

Properly identified address elements are also important when addresses have to be truncated or shortened to fit specific field length requirements. With the proper information in the right fields, specific truncation rules can be applied.

- Parses and analyzes free form addresses and identifies individual address elements
- Detects countries (names, ISO codes, big cities, etc.)
- High processing speed
- Ideal pre-processing stage before validation
- Processes over 30 different character sets
- Formats addresses according to the postal rules of the country of destination
- Standardizes address elements (such as Avenue -> Ave, Street -> St or vice versa)
- Identifies “address trash” elements such as telephone numbers and puts them into the proper fields

For example:

**7031 Columbia Gateway Dr, Suite 101**  
**Columbia MD 21046**  
**USA**



**House number:** 7031  
**Street:** Columbia Gateway Dr  
**Sub-Building:** Suite 101  
**City:** Columbia  
**State:** MD  
**ZIP:** 21046  
**Country:** USA

Data is unstructured or in incorrect fields

All elements are stored in proper fields

## 2.1.3 Global Address Validation

Address validation is the correction process where properly fielded address data is compared against reference tables supplied by postal organizations or other data providers. The AddressDoctor engine has to deal with improperly truncated data, incomplete data, missing address elements, ambiguous names and many other challenges.

The AddressDoctor validation engine is designed to provide the best possible matches while minimizing incorrect modifications to address elements. In many cases, it is not possible to fully validate an address. Here AddressDoctor has a unique deliverability assessment feature that classifies addresses according to their probable deliverability.

- Based on the world’s largest reference database of postal data
- Validates individual address elements to check for correctness using sophisticated fuzzy matching technology
- Batch validation mode only checks elements for correctness and changes them if necessary
- Interactive mode will check elements for correctness and improve them if possible. It provides pick lists of alternatives for ambiguous input data records.
- FastCompletion mode automatically completes truncated or incomplete address elements for fast data entry
- Produces standardized and formatted output based on postal standards and user preferences
- Uses an internal performance-optimized data storage mechanism for the reference data. No third party database software is required

For example:

7031 **G**olumbia Gateway Dr.  
 Suite 101  
 Columbia MD 2104**4**  
 USA



Validation

7031 **C**OLUMBIA GATEWAY DR  
**STE** 101  
 COLUMBIA MD 2104**6**  
 USA

Incorrect input address

Corrected output address

## 2.1.4 The AddressDoctor Software Library in a Nutshell

AddressDoctor provides a single, unified library (DLL on Windows or .so/sf shared library on Unix systems, respectively) with both, C function calls ("C API") and a Java API. The library accesses \*.MD database files, which contain the postal reference data. The software consists of a single engine ("AddressDoctor") which, after initialization, processes input addresses contained in AddressObjects, the data structure for storing an input address, parameter settings and the processing result (for details see the "Concepts" Chapter 5 below).

## 2.2 Supported Platforms

The AddressDoctor 5 software library is developed using the C++ programming language. The resulting API is available for C and Java, provided by a single combined AddressDoctor software library. While the AddressDoctor documentation provides only examples for the most common implementation languages reflected by those two API flavours, they may be used to guide implementation in any programming language, like e.g. C++, C#, VB.Net, PHP, Perl, Ruby or Python. Please note that AddressDoctor can only provide support at an API level, not for implementation specific questions.

While the primary development platform is Windows and Microsoft Visual Studio 2005, the library is available for numerous hardware and software platforms. Initially, AddressDoctor library packages will be available for 64 Bit Windows, AIX, Solaris and Linux (RedHat) only, but eventually the following platform packages will become available (some only on request and when covering the full cost of porting, build, test & support):

OS			Compiler	
Type	Architecture	Version	C(++) Version	JDK Version
<b>Windows</b>	<i>x86</i>	2003/XP	MS VS2005	Sun SE 6
	<i>x86-64</i>	2003/XP	MS VS2005	Sun SE 6
<b>Linux (SUSE)</b>	<i>x86</i>	SLES 10	GNU gcc 4.1	Sun SE 6
	<i>x86-64</i>	SLES 10	GNU gcc 4.1	Sun SE 6
<b>Linux (RedHat)</b>	<i>x86</i>	RHEL 5	GNU gcc 4.1	Sun SE 6
	<i>x86-64</i>	RHEL 5	GNU gcc 4.1	Sun SE 6
	<i>System z (64 Bit)</i>	RHEL 5	GNU gcc 4.1	Sun SE 6
<b>AIX</b>	<i>POWER (64 Bit)</i>	5.3	IBM XL C++ 9	IBM SE 6
<b>Solaris</b>	<i>SPARC (64 Bit)</i>	10	Sun Studio 12	Sun SE 6
<b>HP-UX</b>	<i>IA64</i>	11.23	HP aCC A06.23	HP SE 6

Please inquire with AddressDoctor Support (see chapter 9.3) about the individual availability of certain platform package versions.

## 2.3 System Requirements

AddressDoctor has been designed to achieve the best possible performance while being highly efficient in its memory and resource usage. In order to ensure best possible performance, a fast I/O system and sufficient memory is recommended. At the time of writing, the entire worldwide postal reference database requires around 15 to 20 GB of disk space. Additional disk space is needed when US certified databases are used as well. As with most applications, the engine will perform better if more memory and a faster processor are installed. The minimum requirements are 512 MB of memory for validation operations and 128 MB of memory, if only parsing is required.

To optimize performance, the most commonly used databases should reside in memory (see chapter 6.25 for details). Thus it is recommended to have at least 1GB RAM available, up to at least 16 GB needed for loading the full reference database set into memory. As this exceeds the maximum amount

of memory of 3GB a 32 Bit operating system can typically address (see chapter 6.25 again), AddressDoctor strongly recommends using 64 Bit operating systems in production.

## 3 Setup

The AddressDoctor software library has been designed to be independent of other software modules, thus easing the setup process. All components have been linked in such a way that no external dependencies on libraries or DLL files (on Windows) exist, apart from absolutely necessary core system libraries like KERNEL32.DLL (on Windows) or libc.so (on Linux). In some cases, runtime files or software development environments may be required for the demo applications.

### 3.1 General Remarks

The AddressDoctor software library is provided as download in separate ZIP files (packages). There are packages for:

- Software libraries
- Documentation
- Postal reference data (see section 9.1)

#### 3.1.1 Software packages

The file names of the packages for the software contain the release date in the format YYMMDD as well as the release version in the format 5.0.x.yyyy. In this naming convention, x is the major build version and yyyy is the minor build version with a length of three to four digits. In addition, the file names include the platform (PPP) and the architecture used (32 or 64-Bit). File names may contain compiler information as well, in case different compiler versions are available for one platform.

The file names will thus be as follows (all examples are without compiler information):

AD5\_PPP\_32/64\_YYMMDD\_(5.0.x.yyyy).zip

#### **C/Java Packages:**

WIN: Windows

SUS: Linux SUSE

RHT: Linux Red Hat

AIX: AIX

SOS: Solaris SPARC

HPU: HP-UX

As an example a file could be named AD5\_SOS\_32\_090210\_(5.0.11.384).zip, which contains the AddressDoctor library with C and Java API for Solaris Sparc 32 bit. File names containing extra compiler information would look like AD5\_SOS\_32\_090410\_(5.0.11.392)-sun.studio.11.zip. This contains the AddressDoctor library with C and Java API for Solaris Sparc 32 bit, compiled using Sun Studio 11.

### 3.1.2 Documentation package

The documentation for all wrappers and platforms is contained in a ZIP file with a name following this naming convention:

```
AD5_DOC_YYMMDD_(5.0.x.yyyy).zip
```

### 3.1.3 Postal reference data packages

The postal reference data is available in individual ZIP files for each supported country and territory. These files are named as follows:

```
DB5_XXX5ZZ_YYMMDD.zip
```

Once again the YYMMDD stands for the date of the release of the database. The XXX is replaced by the ISO-3 alpha code (ISO 3166) as found in the AddressDoctor country list online ([http://www.addressdoctor.com/en/countries\\_data/countries5.asp](http://www.addressdoctor.com/en/countries_data/countries5.asp)), and ZZ denotes the type of reference data: For now BI for Batch/Interactive, FC for Fast Completion, Cx for CERTIFIED and GC for Geocoding. An example of a file name would be DB5\_DZA5BI\_091210.zip for an Algerian Batch/Interactive database released on December 10<sup>th</sup> 2009.

## 3.2 Installing the Library Files

### 3.2.1 C Installation

The C .lib file requires no special setup. Simply unpack the ZIP file preserving the directory structure. The following directories (folders) will be created:

```
/bin
/etc
/include
/lib
/src
```

The *bin* directory contains executable sample applications like the ConsoleDemo (see chapter 7 for details).

At this time, the sub-directories under *etc* contain XML configuration file examples that must be copied to your working directory for adjusting the default behaviour of the ConsoleDemo application. The *include* directory contains all required header files.

The actual .dll (.so/sl on Unix) file is contained in the *lib* directory and must be copied to your shared library path (`echo %PATH%` on Windows or `echo $LD_LIBRARY_PATH` on Unix, see chapter 7.1 as well). The code of the sample applications (see chapter 7.1) is located in the *src* directory and its sub-directories.

Take extra care to remove any prior versions of the AddressDoctor shared library from your shared library path to avoid confusion and unnecessary support effort. If used, please make sure your configuration XML files are present in the directory your application references them from (i.e. in case of the AddressDoctor ConsoleDemo, the working directory the executable is called from).

On many UNIX platforms, increasing the thread stack size to at least 1MB is required, e.g. using `export AIXTHREAD_STK=1000000` on AIX or `export PTHREAD_DEFAULT_STACK_SIZE=1000000` on HP-UX, furthermore we recommend setting `ulimit -s unlimited`.

### 3.2.2 Java Installation

Using the AddressDoctor software library via the accompanying JAR archive requires a Java Runtime Environment (JRE). It has been developed and thoroughly tested with the Sun JRE version 6 (or rather IBM JRE 6 for AIX, HP JRE 6 for HP-UX, respectively) that can be downloaded from the Java website at <http://java.sun.com/javase/downloads>. While no backward compatibility tests have been conducted with earlier JRE versions, it may be possible to use the AddressDoctor software library with JRE versions before 6. AddressDoctor will not support systems running earlier JRE versions, however. To develop your own applications, the entire Java SDK is required.

The software library ZIP archive contains several files that should be extracted to a directory on your computer preserving the stored folder names (for an explanation of the archive structure see chapter 3.2.1 above). After extraction, files might need to be copied as follows:

#### Windows

Copy AddressDoctor5.jar and AddressDoctor5.dll to the classpath of your Java Runtime. For instance C:\Program Files\Java\jre\lib\ext typically resides in the system-wide classpath, although it is recommended practice to use and explicitly set application specific classpaths using the `-cp` switch, for example after unpacking the ZIP archive to the present working directory (see chapter 7.1 as well):

```
java -Xss2048k -cp bin;lib/AddressDoctor5.jar -Djava.library.path=lib ConsoleDemoJava
```

#### Solaris and other Unix versions

Copy AddressDoctor5.jar and libAddressDoctor5.so (resp. libAddressDoctor5.sl in case of the HP-UX version of the Java wrapper) to the classpath of your Java Runtime. For instance `/usr/j2se/jre/lib/ext` typically resides in the system-wide classpath, although it is recommended practice to use and explicitly set application specific classpaths using the `-cp` switch, for example after unpacking the ZIP archive to the present working directory (see chapter 7.1 as well):

```
java -Xss2048k -cp bin:lib/AddressDoctor5.jar -Djava.library.path=lib ConsoleDemoJava
```

Take extra care to remove any prior versions of the AddressDoctor library from your system wide classpath to avoid confusion and unnecessary support effort. If used, please make sure your configuration XML files are present in the directory your application references them from (i.e. in case of the AddressDoctor ConsoleDemoJava, the working directory the executable is called from).

Please ensure that sufficient memory can be allocated by your application. At this time we recommend 2048k thread stack size if you intend to use the validation functionality, as well as a minimum of 512m of heap space. Assuming the name of the main class of your application is `MyApp` and you are using the AddressDoctor software library on Linux in the `lib` sub-directory, compile and start it as follows:

```
javac -cp ./lib/AddressDoctor5.jar MyApp.java
java -Xss2048k -Xms512m -Xmx2048m -cp ./lib/AddressDoctor5.jar
-Djava.library.path=lib MyApp
```

The Java Virtual Machine may encounter a limit on the amount of heap memory it can assign to an application, typically between 1.5-2,5 GB on 32Bit operating systems. This effectively limits the number of databases that can be preloaded.

On many UNIX platforms, increasing the thread stack size to at least 1MB is required, e.g. using `export AIXTHREAD_STK=1000000` on AIX or `export PTHREAD_DEFAULT_STACK_SIZE=1000000` on HP-UX, furthermore we recommend setting `ulimit -s unlimited`. Additionally, the IBM J9 JVM will require the JVM call `java -Xmso2048k` for increasing the OS stack size.

## 3.3 Installing the Reference Databases

The postal reference databases are named XXX5YY.MD where XXX stands for the ISO-3 alpha code (ISO 3166) of the country and YY for the database type. The postal reference databases are read only and platform independent. The same database files may be used on all supported platforms.

Batch/Interactive	ISO5BI.MD
FastCompletion	ISO5FC.MD
Certified	ISO5Cx.MD with $x=\{1,\dots,n\}$
Geocoding	ISO5GC.MD
Supplementary:	ISO5Ex.MD with $x=\{1,\dots,n\}$

For example the following databases are presently available for Germany:

```
DEU5BI.MD
DEU5FC.MD
DEU5GC.MD
```

### 3.3.1 Installation Notices

All reference database ZIP files are typically unpacked into the same directory, but storing them on different storage devices is also supported (via `SetConfig.xml`, see the respective DTD in Appendix 10.1

and chapter 6.5). Several applications may share a common set of reference databases on a shared read-only drive, although performance might suffer in such a setup (unless all databases are fully pre-loaded to memory, see chapter 6.25). When full pre-loading of all database files you require is not an option, AddressDoctor strongly recommend using SSDs (solid state drives) instead of mechanical HDDs (hard disk drives) for the database files. This will significantly improve performance, especially under multi-threading conditions (see chapter 6.25 for more details).

Postal reference database files carry expiration date information to honour data provider license terms and help customers determine whether they are using current and relevant reference data. This ExpirationDate information is accessible via GetConfig.xml (see the respective DTD in Appendix 10.1). In certain cases a database file will be no longer accessible when its ExpirationDate has elapsed, e.g. when required by data provider license terms.

Depending on the number of database files in use, it might well be necessary to raise the number of file handles or descriptors available to a process, e.g. by virtue of setting `ulimit -n 8192` on UNIX type operating systems. Please note that additional internal limitations of the libc version used might interfere as well, which then calls for an upgrade to the latest patch version or fix pack of the operating system used.

### 3.3.2 Special Remarks for USA CASS Certified Mode

The US CASS certified processing requires additional databases that contain CASS related information such as Carrier Route codes, EWS, ZIPMOVE, LACSLink, DPV, DFS2, SuiteLink etc. This information is contained in files named USA5C1.MD to USA5C22.MD (at the time of writing, subject to additions).

These CASS specific database files must be present in the database directory in order to have their respective information available for the CERTIFIED process mode. While the ZIPMOVE file provides information about past ZIP code changes, the Early Warning System (EWS) file contains information about upcoming changes to ZIP codes and both are required for CASS certification. The regular EWS and ZIPMOVE database updates by USPS are made available through AddressDoctor and need to be placed in the database directory.

For DPV, LACSLink or SuiteLink information the additional files mentioned above are needed, but USPS licensing terms do not allow storing this data outside the US. Therefore, this data is available only to US customers.

Since CASS Cycle L (2007 - 2009), DPV and LACSLink processing are mandatory to achieve a CASS certification, with CASS Cycle M (2009 - 2011) SuiteLink was made mandatory as well. Even with the certified database files missing, the AddressDoctor engine will still be able to add ZIP+4 Codes, as long as USA5BI.MD is available in the database folder (see chapter 6.24 as well)

SuiteLink contains suite numbers for business addresses in selective highrise buildings and targets highrise addresses with high-volume default mail. SuiteLink also improves business addressing information through assignment of suite numbers. The new data provided by the USPS has been

provided to AddressDoctor's end user through an updated USA5C18.MD file. AddressDoctor will also allow records with input suite data that did not match to the ZIP4 file to go through the SuiteLink process ignoring the input suite data. If a match is found during SuiteLink processing the input suite data will be retained in the residue component and output on DAL2 as required by the USPS to retain the extraneous data.

Residential Delivery Indicator (RDI) processing has been added to AddressDoctor for the CASS Cycle N (2011 – 2012) processing and is available with the 5.2.7 release. This component is optional and is not required to get the postal discounts. The new databases needed for RDI processing are USA5C22.MD and USA5C23.MD. The RDI processing is intended for parcel shippers, their agents or analysts. The result of the processing is a single bit of "Y" or "N". The "Y" represents a residential delivery and is determined by the zip9 or zip11 not being found in the database. The "N" represents a business delivery and is determined by the zip9 or zip11 being found in the database.

Customers interested in RDI need to purchase the data from USPS directly and rename the files rts.hs9 to USA5C22.MD and rts.hs11 to USA5C23.MD manually.

### **3.3.3 Special remarks for Canadian SERP Certified Mode**

The Canada SERP processing requires an additional CAN5C1.MD containing the PoCAD (Point of Call Address Data) which has been introduced with the 2011 SERP cycle. The file has to be placed in the database directory specified in the SetConfig.xml file. SERP processing will not be possible without this database.

### **3.3.4 Special Remarks for Australian AMAS Certified Mode**

The Australian AMAS certified processing requires 2 additional databases: AUS5C1.MD and AUS5C2.MD. These database files need to be placed in the database directory specified in SetConfig.xml in Section <DataBase> with Type="CERTIFIED" for ISO="AUS" or "ALL".

Without the additional files, no AMAS processing is possible.

The databases contain Postal Address File (PAF) data which includes Australia Post's Delivery Point Identifiers (DPIDs).

## 4 Quick Start Guide

To install the AddressDoctor software library, simply unpack the ZIP file for the platform in question preserving the directory structure, as described in more detail in section 3. Similarly, unpack the postal reference database files to a destination directory of your choice.

The software library consists of a single engine ("AddressDoctor") which, after initialization, processes input addresses contained in AddressObjects, the data structure for storing an input address, parameter settings and the processing result (for details see the "Concepts" chapter 5 below).

### 4.1 First-Time Use of AddressDoctor 5

The engine needs to be initialized by a specific sequence:

- `AD_Initialize()` must be called to actually initialize the engine. It evaluates the settings and configures the engine accordingly. Only after this function has returned successfully `AD_GetAddressObject()` or any other functions may be called.
- `AD_DeInitialize()` must be called last to de-initialize the engine; the engine is then ready to be initialized again; all AddressObjects must have been released by calling `AD_ReleaseAddressObject()` before calling `AD_DeInitialize()`.

Consequently, include the following minimal C example code (the program flow is similar for Java) for correcting a single address from Singapore in your application (please also refer to the latest API documentation, see Appendix 10.2):



```
AD_AOHandle hAOHandle;
char sResultXML[ 16 * 1024 ];

AD_Initialize(
    "<?xml version='1.0' encoding='iso-8859-1' ?>\n"
    "<!DOCTYPE SetConfig SYSTEM 'SetConfig.dtd'\n"
    "<SetConfig>\n"
    "<General /\n"
    "<UnlockCode>(Enter Code here)</UnlockCode>\n"
    "<DataBase CountryISO3='ALL' Type='BATCH_INTERACTIVE' Path='/ADDB'
    PreloadingType='NONE' /\n"
    "</SetConfig>\n",
    NULL,
    NULL,
    NULL
);
AD_GetAddressObject( &hAOHandle );
AD_SetInputDataXML( hAOHandle,
    "<?xml version='1.0' encoding='ISO-8859-1'?>\n"
    "<!DOCTYPE InputData SYSTEM 'InputData.dtd'\n"
    "<InputData>\n"
```

```

    "<AddressElements>\n"
    "<Country Item='1' Type='NAME'>SGP</Country>\n"
    "<Locality Item='1' Type='COMPLETE'>Singapore</Locality>\n"
    "<PostalCode Item='1' Type='FORMATTED'>048624</PostalCode>\n"
    "<Street Item='1' Type='COMPLETE'>Raffles Place</Street>\n"
    "<Number Item='1' Type='COMPLETE'>80</Number>\n"
    "<Building Item='1' Type='COMPLETE'>#50-01 UOB Plaza 1</Building>\n"
    "<Organization Item='1' Type='NAME'>AddressDoctor GmbH</Organization>\n"
    "</AddressElements>\n"
"</InputData>\n"
);
AD_Process( hAOHandle );
AD_GetResultXML( hAOHandle, sResultXML, sizeof( sResultXML ) );
AD_ReleaseAddressObject( hAOHandle );
AD_DeInitialize();

```

Ensure that the minimal configuration XML (see SetConfig.dtd in chapter 10.1 for configuration setting details) passed upon `AD_Initialize()` contains a valid Unlock Code you received when purchasing the AddressDoctor library and the correct destination path that your reference database files have been unpacked to. See InputData.dtd in chapter 10.1 for more details on the structure of data input as XML using `AD_SetInputDataXML()`. Depending on your requirements, there is also the possibility of using 16 bit input and output (which is the default for Java), see chapter 5.8 for details.

Now compile your application as usual, making sure the AddressDoctor library dependencies are met. How to achieve this varies greatly between platforms and compilers, for example on Linux and using `gcc`, the following command will build the ConsoleDemo C++ example code (see chapters 3.2.1 & 7.1):

```
gcc -Iinclude -Llib -lAddressDoctor5 -lpthread -o bin/ConsoleDemo src/ConsoleDemo.cpp
```

The output of AddressDoctor processing will be provided in `sResultXML` in XML format, see Result.dtd (chapter 10.1) for the structure of the XML result from `AD_GetResultXML()`:

```

<?xml version="1.0" encoding="UTF-16"?>

<Result ProcessStatus="V2"
  ModeUsed="BATCH"
  Count="1"
  CountOverflow="NO"
  CountryISO3="SGP"
  PreferredScript="DATABASE"
  PreferredLanguage="DATABASE">

<ResultData ResultNumber="1"
  MailabilityScore="4"
  ResultPercentage="100.00"
  ElementResultStatus="F0F000F0F000404440E0"

```

```

ElementInputStatus="60600060600020222060"
ElementRelevance="1010001010000000010">
<AddressElements>
  <Country Type="NAME_EN" Item="1">SINGAPORE</Country>
  <Locality Item="1">SINGAPORE</Locality>
  <PostalCode Item="1">048624</PostalCode>
  <Street Item="1">RAFFLES PLACE</Street>
  <Number Item="1">80</Number>
  <Building Item="1">UOB PLAZA 1</Building>
  <SubBuilding Item="1"># 50</SubBuilding>
  <SubBuilding Item="2">01</SubBuilding>
  <Organization Item="1">ADDRESSDOCTOR GMBH</Organization>
</AddressElements>
<AddressLines>
  <RecipientLine Line="1">ADDRESSDOCTOR GMBH</RecipientLine>
  <DeliveryAddressLine Line="1">80 RAFFLES PLACE</DeliveryAddressLine>
  <DeliveryAddressLine Line="2">#50-01 UOB PLAZA 1</DeliveryAddressLine>
  <CountrySpecificLocalityLine Line="1">SINGAPORE
048624</CountrySpecificLocalityLine>
  <FormattedAddressLine Line="1">ADDRESSDOCTOR GMBH</FormattedAddressLine>
  <FormattedAddressLine Line="2">80 RAFFLES PLACE</FormattedAddressLine>
  <FormattedAddressLine Line="3">#50-01 UOB PLAZA 1</FormattedAddressLine>
  <FormattedAddressLine Line="4">SINGAPORE 048624</FormattedAddressLine>
</AddressLines>
<AddressComplete>ADDRESSDOCTOR GMBH
80 RAFFLES PLACE
#50-01 UOB PLAZA 1
SINGAPORE 048624
</AddressComplete>
</ResultData>
</Result>

```

Finally, an example for Java (please note that in comparison to the C example the "Encoding" attribute for the "Input" and "Result" elements has to be explicitly set to UTF-16 or UCS-2 via Parameters.xml here as well as WriteXMLEncoding for both SetConfig.xml and Parameters.xml, as Java defaults to its native 16 Bit string handling, see chapter 5.8):

## Java

```

private static AddressObject m_oAO;

public static void main(String[] args) {
    int iLastError = 0;
    String sResultXML = "";

    try
    {

```

```

AddressDoctor.initialize(
    "<?xml version='1.0' encoding='UTF-16' ?>" +
    "<!DOCTYPE SetConfig SYSTEM 'SetConfig.dtd'" +
    "<SetConfig><General WriteXMLEncoding='UTF-16' />" +
    "    <UnlockCode>(Enter Code here)</UnlockCode>" +
    "    <DataBase CountryISO3='ALL' Type='BATCH_INTERACTIVE'" +
    "        Path='/ADDB' PreloadingType='NONE' />" +
    "</SetConfig>", null,
    "<?xml version='1.0' encoding='UTF-16' ?>" +
    "<!DOCTYPE SetConfig SYSTEM 'Parameters.dtd'" +
    "<Parameters WriteXMLEncoding='UTF-16'" +
    "    <Input Encoding='UTF-16' />" +
    "    <Result Encoding='UTF-16' />" +
    "</Parameters>", null);

iLastError = AddressDoctor.getLastError();
System.out.println("Using AddressDoctor version: " + AddressDoctor.getVersion());
System.out.println("Init returned " + iLastError);
} catch (AddressDoctorException ex)
{
    System.out.println("Exception while initializing " +
        "AddressDoctor: " + ex.toString());
    System.out.println("Further processing not possible, " +
        "application ends!");
    return;
}

try
{
    m_oAO = AddressDoctor.getAddressObject();
} catch (AddressDoctorException ex)
{
    System.out.println("Exception while trying to get an " +
        "AddressObject: " + ex.toString());
    System.out.println("Further processing not possible, " +
        "application ends!");

    try
    {
        AddressDoctor.deinitialize();
    } catch (AddressDoctorException ex2){}
    return;
}

try
{
    m_oAO.setInputDataXML(
        "<?xml version='1.0' encoding='UTF-16'?" +
        "<!DOCTYPE InputData SYSTEM InputData.dtd'" +
        "<InputData>" +
        "<AddressElements>" +
        "    <Key>4711</Key>" +

```

```

" <Country Item='1' Type='NAME'>SGP</Country>" +
" <Locality Item='1' Type='COMPLETE'>Singapore</Locality>" +
" <PostalCode Item='1' Type='FORMATTED'>048624</PostalCode>" +
" <Street Item='1' Type='COMPLETE'>Raffles Place</Street>" +
" <Number Item='1' Type='COMPLETE'>80</Number>" +
" <Building Item='1' Type='COMPLETE'>#50-01 UOB Plaza 1</Building>" +
" <Organization Item='1' Type='NAME'>AddressDoctor GmbH</Organization>" +
"</AddressElements>" +
"</InputData>";
} catch (Exception ex)
{
    System.out.println("Data could not be assigned! Closing " +
        "application: " + ex.toString());
    try
    {
        AddressDoctor.releaseAddressObject(m_oAO);
        AddressDoctor.deinitialize();
    } catch (AddressDoctorException ex2){}
    return;
}

try
{
    AddressDoctor.process(m_oAO);
    iLastError = AddressDoctor.getLastError();
    System.out.println("Process returned " + iLastError);
} catch (AddressDoctorException ex)
{
    System.out.println("Exception during process: " +
        ex.toString());
}

if (iLastError == 0)
{
    try
    {
        sResultXML = m_oAO.getResultXML();
    } catch (AddressDoctorException ex)
    {
        System.out.println("Exception while trying to get " +
            "ResultXML: " + ex.toString());
        return;
    }
    System.out.println(sResultXML);
}

try
{
    AddressDoctor.releaseAddressObject(m_oAO);
    AddressDoctor.deinitialize();
} catch (AddressDoctorException ex)

```

```
{
    System.out.println("Exception while releasing the AO and "+
        "de-initializing AddressDoctor: " + ex.toString());
}
}
```

Please do also refer to the C and Java source code provided for the AddressDoctor 5 ConsoleDemo described in chapter 7.1.

## 4.2 Migration from Version 4

### 4.2.1 New in Version 5

- Thread safe engine, allowing multiple addresses to be processed in parallel by enabling full multi-threading on multiple processor cores
- Improved performance
- Simplified, text and XML based API, including more granular status values
- More predictable and less aggressive parsing
- Built in data enrichment (Geo Coding and more to come)
- Smaller database sizes
- Additional certifications: AMAS and SERP
- Extended CASS processing including RDI

### 4.2.2 Key API Changes

- The C++ and COM API have been discontinued
- The return codes are now signed 32 bit integer values, not enums; the values returned are not part of the API
- The number of AddressObjects is limited; this limit can be set by the engine configuration (MaxAddressObjectCount, see SetConfig.dtd) to control how much of the memory set via MaxMemoryUsageMB is free to be used for preloading country reference databases
- Multiple threads may call `AD_Process()` (with different AddressObjects) in parallel; the maximum number of threads which actually are allowed to process addresses in parallel can be set by the engine configuration
- There are no more UniString objects, either zero-terminated strings are used directly or in- or output is done via XML
- The Preload Object has been eliminated; preloading specifications can be set by the engine configuration
- There are no longer Standardization objects; the corresponding settings can be set by the engine parameter configuration and by the `AD_SetParametersXML()` function for a specific AddressObject
- The engine will never request more memory dynamically from the OS than what has been set by the engine configuration
- Various parameters are passed as zero-terminated strings or via XML; this allows for easy future extensibility of the API
- Apart from 8 bit input, now also 16 bit input is supported (and standard for the Java API)

### 4.2.3 Setting up the AddressDoctor Engine

The AddressDoctor engine processes input addresses (`AD_Process()`) contained in the AddressObjects. As there is only one engine per process, there is no AddressDoctor engine handle.

The engine needs to be initialized by a specific sequence:

- `AD_Initialize()` must be called to actually initialize the engine. It evaluates the settings and configures the engine accordingly. Only after this function has returned successfully `AD_GetAddressObject()` or any other functions may be called.
- `AD_DeInitialize()` must be called last to de-initialize the engine; the engine is then ready to be initialized again; all AddressObjects must have been released by calling `AD_ReleaseAddressObject()` before calling `AD_DeInitialize()`.

The engine stores the following data for its internal use (see `SetConfig.dtd`):

- General engine configuration, i.e., the maximum amount of memory the engine may request from the OS
- The access codes; at least one valid access code must be supplied when calling `AD_Initialize()`
- Optional preloading parameters for the databases

In addition, the engine stores the following parameter data as a default for the AddressObjects (see `Parameters.dtd` in chapter 10.1):

- Process parameters, i.e., the processing mode to be used
- Input parameters, i.e., which input encoding is to be used
- Format specifications for the result, i.e., Casing specifications

This configuration data has default values as specified by the corresponding `SetConfig.dtd`; they can be changed by passing an according config XML as parameter to `AD_Initialize()`:

There is no way to change the configuration after `AD_Initialize()` has been called, this parameter configuration data is used by the AddressObjects by default, when no alternative setting is made for a specific AddressObject.

### 4.2.4 AddressObjects

The AddressObject is a data structure for storing an input address, parameter settings and a result.

AddressObjects store the following (configuration) data:

- Parameter settings (see `Parameters.dtd`)
- An input address (see `InputData.dtd`)
- A result (see `Result.dtd`)
- The last return code

AddressObjects should not be created and destroyed frequently, but rather be reused for performance reasons. Specifically, the parameter settings should be reused to avoid repeated settings overhead.

The AddressDoctor engine manages all AddressObjects: Only a specific number of AddressObjects can be created; this number can be set in the initialization phase of the engine (using the “`MaxAddressObjectCount`” attribute). Recommended setting would be between once up to twice the

number of threads, which must be set using the "MaxThreadCount" attribute and is currently limited to a practical maximum value of 32 threads (please see chapter 5.29 as well)

Please note that the default parameter settings differ between AddressDoctor 4 and 5, for instance the standard settings for script and casing (see chapters 5.12 & 5.14 for details).

## 4.2.5 Direct API

The engine in- and output in AddressDoctor 5 is based on a XML API. See the corresponding DTDs in Appendix 10.1 and the following chapter 5 for details.

To ease the transition from Version 4, the engine partly supports setting and getting some of the XML values and attributes directly, for example

```
AD_SetInputAddressElement( hAOHandle, "PostalCode", 1, "67133" )
```

sets the item 1 postal code to "67133".

To set a street and a dependent street using the direct API in C, the "Item" parameter has to be 1 for the first and 2 for the second:

```
AD_SetInputAddressElement( hAOHandle, "Street", 1, NULL, "Main St 5" );
AD_SetInputAddressElement( hAOHandle, "Street", 2, NULL, "Dependent St 8" );
```

For example, to set 3 formatted address lines, the "Line" parameter has to be set from 1 to 3:

```
AD_SetInputAddressLine( hAOHandle, "FormattedAddressLine", 1, "AddressDoctor GmbH" );
AD_SetInputAddressLine( hAOHandle, "FormattedAddressLine", 2, "Roentgenstr. 9" );
AD_SetInputAddressLine( hAOHandle, "FormattedAddressLine", 3, "D-67133 Maxdorf" );
```

Similarly, setting street and dependent street in Java:

```
m_oAO.setInputAddressElement("Street", 1, "COMPLETE", "Main St 5");
m_oAO.setInputAddressElement("Street", 2, "COMPLETE", "Dependent St 8");
```

And setting 3 formatted address lines in Java:

```
m_oAO.setInputAddressLine("FormattedAddressLine", 1, "AddressDoctor GmbH");
m_oAO.setInputAddressLine("FormattedAddressLine", 2, "Roentgenstr. 9");
m_oAO.setInputAddressLine("FormattedAddressLine", 3, "D-67133 Maxdorf");
```

Both kinds of API functions may be intermixed although this is not recommended; specifically, please note that calling `AD_SetInputDataXML()` clears any possibly existing input data beforehand as input may not be assigned using both, direct and XML API (see the respective return code in chapter 5.25 as well).

Complete direct API examples in C and Java follow (please also refer to the latest API documentation, see Appendix 10.2):



```

AD_AOHandle hAOHandle;
char sCompleteAddress[ 4096 ];
AD_U32 ulNumResults;

AD_Initialize(
    "<?xml version='1.0' encoding='iso-8859-1' ?>\n"
    "<!DOCTYPE SetConfig SYSTEM 'SetConfig.dtd'>\n"
    "<SetConfig>\n"
    "<General /\>\n"
    "<UnlockCode>(Enter Code here)</UnlockCode>\n"
    "<DataBase CountryISO3='ALL' Type='BATCH_INTERACTIVE' Path='/ADDB'
    PreloadingType='NONE'/>\n"
    "</SetConfig>\n",
    NULL,
    NULL,
    NULL
);
AD_GetAddressObject( &hAOHandle );
AD_SetInputAddressElement( hAOHandle, "Country", 1, NULL, "SGP" );
AD_SetInputAddressElement( hAOHandle, "Locality", 1, NULL, "Singapore" );
AD_SetInputAddressElement( hAOHandle, "PostalCode", 1, NULL, "048624" );
AD_SetInputAddressElement( hAOHandle, "Street", 1, NULL, "Raffles Place" );
AD_SetInputAddressElement( hAOHandle, "Number", 1, NULL, "80" );
AD_SetInputAddressElement( hAOHandle, "Building", 1, NULL, "#50-01 UOB Plaza 1" );
AD_SetInputAddressElement( hAOHandle, "Organization", 1, NULL, "AddressDoctor GmbH" );
AD_Process( hAOHandle );
AD_GetResultCount( hAOHandle, &ulNumResults );
if( ulNumResults > 0 )
    AD_GetResultAddressComplete( hAOHandle, 1, sCompleteAddress,
sizeof( sCompleteAddress ) );
AD_ClearData(); // Not necessary here, only if hAOHandle were to be filled with
another input address
AD_ReleaseAddressObject( hAOHandle );
AD_DeInitialize();

```

Or, alternatively:

Java

```
private static AddressObject m_oAO;
public static void main(String[] args) {
    try {
        // Initialize the engine
        AddressDoctor.initialize(
            "<?xml version='1.0' encoding='UTF-16' ?>" +
            "<!DOCTYPE SetConfig SYSTEM 'SetConfig.dtd'" +
            "<SetConfig><General WriteXMLEncoding='UTF-16' />" +
            "    <UnlockCode>(Enter Code here)</UnlockCode>" +
            "    <DataBase CountryISO3='ALL' Type='BATCH_INTERACTIVE'" +
            "        Path='/ADDB' PreloadingType='NONE' />" +
            "</SetConfig>", null,
            "<?xml version='1.0' encoding='UTF-16' ?>" +
            "<!DOCTYPE SetConfig SYSTEM 'Parameters.dtd'" +
            "<Parameters WriteXMLEncoding='UTF-16'" +
            "    <Input Encoding='UTF-16' />" +
            "    <Result Encoding='UTF-16' />" +
            "</Parameters>", null);

        // Get an AddressObject to use
        m_oAO = AddressDoctor.getAddressObject();

        // Set the address elements
        m_oAO.setInputAddressElement("Country", 1, "ISO_3", "SGP");
        m_oAO.setInputAddressElement("Locality", 1, null, "Singapore");
        m_oAO.setInputAddressElement("PostalCode", 1, null, "048624");
        m_oAO.setInputAddressElement("Street", 1, "NAME", "Raffles Place");
        m_oAO.setInputAddressElement("Number", 1, null, "80");
        m_oAO.setInputAddressElement("Building", 1, null, "#50-01 UOB Plaza 1");
        m_oAO.setInputAddressElement("Organization", 1, null, "AddressDoctor GmbH");

        // Process the AddressObject
        AddressDoctor.process(m_oAO);

        // If there is at least one result, print the address on the screen
        if (m_oAO.getResultCount() > 0)
            System.out.println(m_oAO.getResultAddressComplete(1));

        // Clear the AddressObject so that it may be filled with another input address
        m_oAO.clearData();

        // Release the AddressObject, all AddressObjects must be released to deinitialize
        AddressDoctor.releaseAddressObject(m_oAO);

        // Deinitialize the engine
        AddressDoctor.deinitialize();
    }
    catch (AddressDoctorException e) {
```

```
        System.exit(1);  
    }  
}
```

Please take due note of the native XML example shown in chapter 4.1 to understand the differences between direct and XML type API usage for evaluation of which API mode might better suit your needs. Also, the example given above pertains 8 Bit data handling, see chapter 5.8 “Input and Output Encoding” for the differences when handling 16 Bit data (which is the default for Java).

## 4.2.6 Transliteration (formerly UniString Object)

The “Transliteration only” process mode in Version 4 via the UniString object has been superseded: To obtain transliterated address elements without validation, simply set the “Mode” attribute of the “Process” element in Parameters.xml (see DTD in chapter 10.1) to PARSE using `AD_SetParametersXML()` before submitting your AddressObject to `AD_Process()`. For this specific use case, setting an “OptimizationLevel” of NARROW would also be recommended (see chapter 5.26).

## 4.2.7 Unlock Code Mechanism

The AddressDoctor unlock code mechanism has been slightly redesigned, please note that multiple unlock codes are to be passed as separate XML elements (see chapter 6.4) and that information which databases have been unlocked may be queried using `AD_GetConfigSettingsXML()` - see chapter 6.6 for details.

## 5 Concepts

This chapter explains the major functions of the AddressDoctor software library and shows how Transliteration, Formatting and the two major address processing stages Parsing and Correction interact. The entire functionality is implemented through two objects, AddressDoctor (frequently referred to as “Engine”) and AddressObject. For the C and Java interfaces these objects have been mapped to functions.

The following figure may act as a general guideline in understanding the sequence of the different processing stages an address is subjected to by the AddressDoctor engine, a more detailed discussion is given in chapter 5.6:



AddressDoctor supports address parsing and address verification for more than 240 countries and territories through one API. Consequently, AddressDoctor scales easily from a single country setup to multi country or even global scenarios.

### 5.1 Character Set Mapping

In today’s computer environments we encounter numerous character sets. In the early days of computing most systems used either EBCDIC or ASCII character sets. Programmers and system designers used the concept of code pages to cope with the limited characters that were available on these computers.

Several years later Unicode was introduced to address the problems associated with the large number of different character sets that are used around the world. With room for more than 65000 characters it seemed like a sufficient solution at first. Now even this character set has become too small to represent all characters from around the world, thus newer versions of Unicode support well over a million. When data is transferred or transported between different computer systems, character set mapping problems frequently occur.

These problems result from different numerical values that are assigned to the “same” character in different character sets. While the basic ASCII characters of the Latin alphabet like A, B, and C are usually represented with the same numerical values, the problems often start once accented or other non-standard characters are used. These are often encoded differently in each character set.’

The following table shows a comparison of the decimal values for some characters in the Latin and Unicode character set:

Character	Latin	Unicode
A	65	65
B	66	66
Å	143	197
β	225	223
ㄝ	—	12373

While some characters have the same numerical representation in both character sets, others have different values. If a file is created using one character set and then displayed using another character set, mapping problems will occur that lead to an illegible text at best. Taking the text ABÅβ from Latin to Unicode without any mapping will lead to the following output text AB ́ á that is clearly different from the original input. Other characters such as the Japanese cannot even be represented in Latin and would be lost when a Unicode file would be viewed with a Latin interpretation.

AddressDoctor's transliteration stage offers functionality<sup>1</sup> to address these issues. String data is internally stored in the Unicode UCS 2 format. Strings can be assigned in any of the more than 30 supported character sets (and possibly more). If data is retrieved in another character set, a mapping takes place to ensure that the characters are properly represented in the other character set. Provided that each character has a representation in the other character set, no information is lost. Characters that have no representation in a particular character set (such as ㄝ in Latin) will be mapped to a space.

## 5.2 Transliteration

Transcription and Transliteration are processes of changing one character of one character set into other characters of another character set, such as converting from Greek to Latin, or Japanese Katakana to Latin:

A transliteration uses invertible mapping, so that a transliteration can be reversed without information loss. In contrast, a transcription aims to provide non-native speakers with an approximate pronunciation of a word, based on the pronunciation rules of their own language. In practice, transliterations are consistent with transcriptions for many character sets, while no real (i.e. invertible) transliterations exist for most syllable or ideographic languages. Thus for the rest of this document, transliteration is used to denote both, transliteration or transcription.

Transliteration surpasses mere character set mapping, which is limited to the mapping between different numerical representations of a character (see the example in chapter 5.1). A language such as Japanese with the Katakana, Hiragana and Kanji characters has no direct representation in the English

<sup>1</sup>Note for users of the previous version: By virtue of the AD\_UniString object (see chapter 4.2.2)

language. However, each Japanese character has a certain associated sound that can be approximated using phonetic Latin characters.

Numerous transliteration schemes have been introduced for different languages. The following examples show how transliteration works for different languages:

Ä → AE (Latin → Latin)  
ĝ → g (Latin → Latin)  
ヶ → ka (Japanese → Latin)  
Ж → ZH (Cyrillic → Latin)

We can see that even within the Latin alphabet transliteration can be useful when certain extended characters cannot be represented in the target character set.

Most languages use only a subset of the sounds a normal human could produce and of course these subsets differ from language to language. If a sound used by one language cannot be represented correctly in a different script, it must be approximated: This approximation may be quite inadequate if the sounds used in the target language for transliteration differ significantly from the sounds in the original language.

This problem is especially relevant when transliterating languages with very few syllables, such as Japanese (much less so for Chinese). Here are some examples of circular transliteration (i.e. English → kana → English) leading to dramatic changes:

Original: Philippines  
Japanese: フイリピン  
Transliterated: Firipin

Original: Düsseldorf  
Japanese: ギュッセルドルフ  
Transliterated: Dyusserudorufu

Original: Beethoven  
Japanese: ベートーベン  
Transliterated: Betoben

These transliterated words provide challenges when working with transliterated place names for non-Asian countries that were previously represented in an Asian language. Examples using character set mapping and transliteration may be found in chapter 6.15.

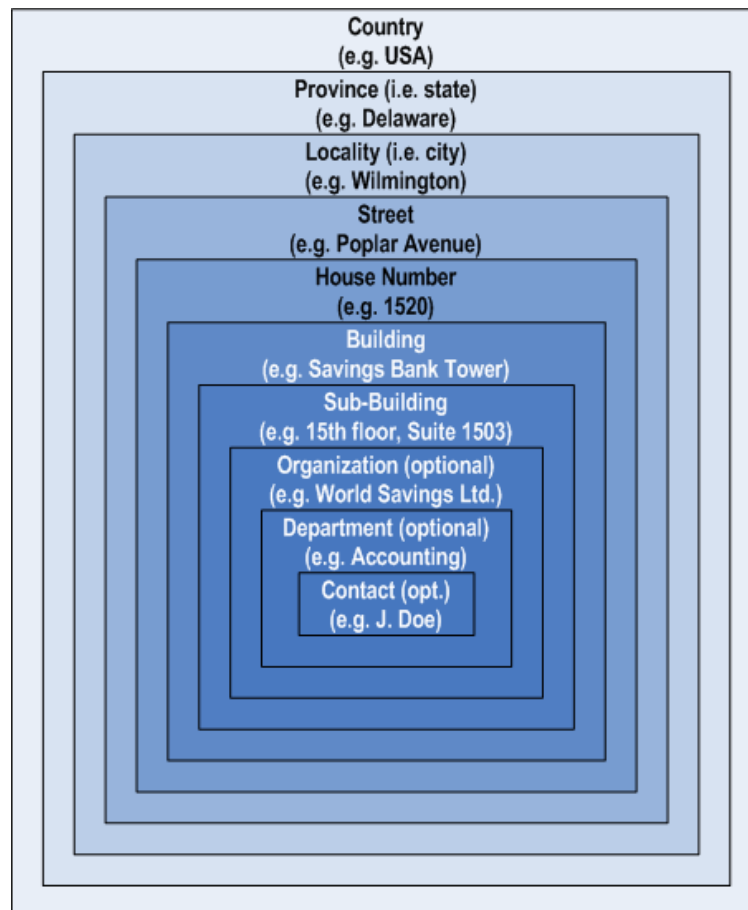
### 5.3 Address Element Abstraction

Addresses have developed differently in countries and cultures around the world. AddressDoctor uses mapping and a hierarchical approach to describe the various address elements. At the foundation of the address “pyramid” is a country. Currently there are 191 United Nations member countries as well as

several dependent and independent territories around the world. AddressDoctor covers a total of over 240 countries and territories in its postal reference databases.

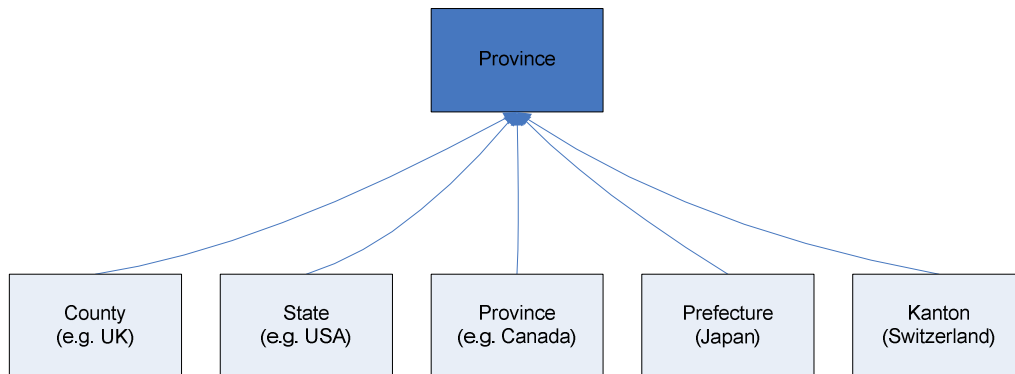
A country is often subdivided into provinces or regions. These regions are not always required in postal addresses, however. Cities or localities belong to a region and buildings are typically assigned to streets. The building themselves can often be subdivided, which is done through the concept of a sub-building. An example for a sub-building would be a floor or suite number. Organizations then reside in sub-buildings and are subdivided into departments that in turn employ people (known as contacts).

The following figure visualizes this abstraction model graphically:



Of course, not all elements are present (or required) in all cases. Non-business addresses for instance will lack the Organization and Department elements. Also, different names may be in use for similar elements: As an example, the territorial subdivision in the USA is called a state. Canada calls it a province, while Switzerland has given the name Kanton to this subdivision. AddressDoctor has defined general terms that allow mapping these concepts globally to the standardized “Item” fields of the AddressObject (see chapters 5.7 and 5.9).

As an example, the AddressObject contains an attribute with the name "Province". Depending on the country, this field may either contain the state (USA), the county (UK), the province (Canada), the prefecture (Japan), the Kanton (Switzerland), the Bundesland (Austria or Germany) and so on. The following figure illustrates this mapping:



## 5.4 Address Parsing

Addresses stored in computer systems were often entered by humans. Frequently, people entering data do not understand the nature of the information that they enter. Quite often the fields for storing the data are not sufficient, because they are either not long enough or because there are too few fields to store addresses from countries all over the world. Please see section 9.4 on page 112 for a recommended database layout to store addresses from all around the world.

Computer programs that validate postal addresses often rely on the information provided by the field names to identify address elements. While this is sufficient in a few cases, most of the time this information can be misleading because information was entered in the wrong fields. It is frequent that consignee information such as names are placed in address fields that are designed to store street information. Postal codes are input in city fields and building numbers are placed in wrong locations. These are just some of the easier challenges found in international addresses where incorrectly fielded data is omnipresent.

Analyzing address elements and assigning them to the proper fields is one of the most difficult challenges of handling postal addresses. AddressDoctor implements a parsing engine that is **independent of postal reference data**.

As a consequence, the parser as implemented in the AddressDoctor engine can be used without any postal reference data present and is especially suited for OEM integration scenarios where the reference data can be added as needed.

The parsing functionality is implemented by the PARSE process mode (and is implicitly included with the other process modes described later in chapter 5.11). It can either work on fielded data that is retrieved from address element fields or from totally unfielded data as it can be found in databases that have just

a line by line layout for address data. While structuring an unfielded address seems more difficult at first, handling the potentially conflicting information in a seemingly fielded address can be an even more difficult challenge. Here the name of the field might indicate that a street should be expected but the software has to decide that this “hint” is possibly nonsense and be bold enough to decide to ignore this information.

Depending on the “OptimizationLevel” attribute set in Parameters.xml (see the DTD in chapter 10.1), the AddressDoctor parser behaves differently in respect to fielded input element assignment (see chapter 5.26 for more detail):

- NARROW: The parser will honor input assignment strictly, with the exception of separation of House Number from Street information.
- STANDARD: The parser will separate address element more actively, e.g.
  - Province will be separated from Locality information
  - PostalCode will be separated from Locality information
  - House Number will be separated from Street information
  - SubBuilding will be separated from Street information
  - DeliveryService will be separated from Street information
  - SubBuilding will be separated from Building information
  - Locality will be separated from PostalCode information
- WIDE: Parser separation will happen similarly to STANDARD, but additionally up to 10 parsing candidates will be passed to validation for processing. Validation will widen its search tree and take additional reference data entries into account for matching.

It is very important to note that, apart from the case of NARROW, such mixed input will naturally result in information to be separated out into different AddressElement Items by the parser. For example, while Street and SubBuilding were jointly assigned as Street on input, that SubBuilding will be located in a SubBuilding Item on output (and not in Street) – provided it could be identified as such.

## 5.5 Address Validation

When validating an address, each component is compared against a postal reference data set that is stored in AddressDoctor’s reference databases (see section 9.1). All elements of an address may be correct, thus resulting in positive validation. On the other hand, each individual element may match against the reference data, but the components do not make sense when looked at in their combination.

Let us regard an example:

City: Wilmington  
ZIP: 90210  
State: CA

In this example each component is correct by itself. However, the components do not match, as the ZIP code does not belong to Wilmington and Wilmington is not in the state of California. Whenever it is possible, AddressDoctor will attempt to correct such errors. To do this without endangering potentially correct data elements and creating “false positives”, great care is taken and very sophisticated algorithms are used to analyze and potentially correct the data.

The algorithms used by AddressDoctor include fuzzy matching and heuristics to predict the best possible correction for an address. It is always AddressDoctor’s intention to correct or improve an address if at all possible. Here it differs from most postal certification schemes such as the Coding Accuracy Support System (CASS) as introduced by the US Postal Service. These certification schemes intend to prevent poorly addressed mail from entering the postal mail stream, thus easing the work of the postal organization. AddressDoctor’s intent, however, is to improve as many addresses as possible (see chapter 5.26 for more detail on “OptimizationLevel”).

## 5.6 The AddressDoctor Engine

The AddressDoctor engine processes input addresses (`AD_Process()`) contained in the AddressObjects. As there is only one engine per process, there is no AddressDoctor engine handle.

The engine needs to be initialized by a specific sequence:

- `AD_Initialize()` must be called to actually initialize the engine. It evaluates the settings and configures the engine accordingly. Only after this function has returned successfully `AD_GetAddressObject()` or any other functions may be called.
- `AD_DeInitialize()` must be called last to de-initialize the engine; the engine is then ready to be initialized again; all AddressObjects must have been released by calling `AD_ReleaseAddressObject()` before calling `AD_DeInitialize()`.

The engine stores the following data for its internal use (see `SetConfig.dtd`):

- General engine configuration, i.e., the maximum amount of memory the engine may request from the OS
- The access codes; at least one valid access code must be supplied when calling `AD_Initialize()`
- Optional preloading parameters for the databases

In addition, the engine stores the following parameter data as a default for the AddressObjects (see `Parameters.dtd` in chapter 10.1):

- Process parameters, i.e., the processing mode to be used
- Input parameters, i.e., which input encoding is to be used
- Format specifications for the result, i.e., Casing specifications

This configuration data has default values as specified by the corresponding SetConfig.dtd; they can be changed by passing an according config XML as parameter to `AD_Initialize()`:

There is no way to change the configuration after `AD_Initialize()` has been called, this parameter configuration data is used by the AddressObjects by default, when no alternative setting is made for a specific AddressObject.

## 5.7 AddressObjects

The AddressObject serves as a container object for a postal address. It has several properties that can store individual components of an address such as postal (ZIP) code, street name and building number, but also company or contact names.

AddressObjects store the following (configuration) data:

- Parameter settings (see Parameters.dtd)
- An input address (see InputData.dtd)
- A result (see Result.dtd)
- The last return code

AddressObjects should not be created and destroyed frequently, but rather be reused for performance reasons. Specifically, the parameter settings should be reused to avoid repeated settings overhead.

The AddressDoctor engine manages all AddressObjects: Only a specific number of AddressObjects can be created; this number can be set in the initialization phase of the engine (using the “MaxAddressObjectCount” attribute). Recommended setting would be between once up to twice the number of threads, which must be set using the “MaxThreadCount” attribute and is currently limited to a practical maximum value of 32 threads (please see chapter 5.29 as well).

For setting the address elements (described in chapter 5.3, examples are Organization, Department, Street, Province, PostalCode and Locality) in an AddressObject, see chapter 6.7, for retrieving them, see chapter 6.11. Retrieving address elements individually is especially useful when data to be processed originates from a database that has individual fields for address elements.

**Alternatively**, the AddressObject may be assigned unfielded FormattedAddressLine data (see chapter 6.7.4), where the address representation is only structured by delimiters such as linefeeds. The FormattedAddressLine representation is also helpful when retrieving processed address data: It will return the processed data according to the country specific formatting rules. When assigning AddressObject values, either the address elements **or** the FormattedAddressLine representation should be used, while both representations are provided on output (see the Result.xml example at the end of chapter 3).

## 5.8 Input and Output Encoding

The XML-encoding is passed within the XML header `<?xml ?>`; if none is explicitly set, UTF-8 or UTF-16 is the default (as defined by the XML standard and depending on the bit width chosen as described below). The different encodings for XML input and output may be specified via attributes (see chapter 6.3 and `SetConfig.dtd/Parameters.dtd` in chapter 10.1). For the direct API, the engine default encoding is ISO-8859-1.

There may be 8 and 16 bit input and result data; to deal with both character sizes, there are two sets of functions for any `Set...()` or `Get...()` functionality:

- The 8 bit versions have no special naming (i.e. `AD_SetInputDataElement()` OR `AD_GetResultDataParameter()`)
- The 16 bit versions end in **W** (for word, i.e. `AD_SetInputDataElementW()` OR `AD_GetResultDataParameterW()`)

When using the 16 bit API functions it is crucial to have set a corresponding 16 bit encoding, otherwise an encoding error code is returned. The 16 bit input functions `Set...W()` also support an additional parameter for the string length, thereby making it possible to pass non-zero-terminated strings. To enable passing zero-terminated strings of unknown length, the special value `AD_AUTOLEN` can be passed as string length; the engine then automatically determines the length.

The currently active encoding (see `Parameters.dtd`) must match the used function: When a 16 bit function is called, the encoding must also be 16 bit (i.e. UTF-16), consequently.

This is specifically the case for the Java API (see the example in chapter 4.1), which does support 16 bit input and output only, in line with internal Java string handling.

## 5.9 AddressElement Items and AddressLines

Many of the direct (legacy) API functions have an item or line parameter. The same applies to the XML API, where XML element attributes are used for that purpose. These parameter numbers refer to the index or hierarchical level of an address element or line: Items and lines are counted from 1 on, the default for XML is 1 (see the DTDs in 10.1).

To set a street and a dependent street using the direct API in C, the “Item” parameter has to be 1 for the first and 2 for the second:



```
AD_SetInputAddressElement( hAOHandle, "Street", 1, NULL, "Main St 5" );
AD_SetInputAddressElement( hAOHandle, "Street", 2, NULL, "Dependent St 8" );
```

For example, to set 3 formatted address lines, the “Line” parameter has to be set from 1 to 3:

```
AD_SetInputAddressLine( hAOHandle, "FormattedAddressLine", 1, "AddressDoctor GmbH" );
AD_SetInputAddressLine( hAOHandle, "FormattedAddressLine", 2, "Roentgenstr. 9" );
AD_SetInputAddressLine( hAOHandle, "FormattedAddressLine", 3, "D-67133 Maxdorf" );
```

Similarly, setting street and dependent street in Java:



```
m_oAO.setInputAddressElement("Street", 1, "COMPLETE", "Main St 5");
```

```
m_oAO.setInputAddressElement("Street", 2, "COMPLETE", "Dependent St 8");
```

And setting 3 formatted address lines in Java:

```
m_oAO.setInputAddressLine("FormattedAddressLine", 1, "AddressDoctor GmbH");
m_oAO.setInputAddressLine("FormattedAddressLine", 2, "Roentgenstr. 9");
m_oAO.setInputAddressLine("FormattedAddressLine", 3, "D-67133 Maxdorf");
```

Please refer to chapter 6.7 for understanding the valid combinations of AddressElement Items and AddressLines for address data input.

In the XML API case, an example for InputData.xml with two items assigned for sub-elements of the PostalCode (known as ZIP+4, see chapter 6.24) would be:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE InputData SYSTEM "InputData.dtd">

<InputData>
  <AddressElements>
    <Country Item="1" Type="NAME">USA</Country>
    <Locality Item="1" Type="COMPLETE">Raleigh</Locality>
    <PostalCode Item="1" Type="UNFORMATTED">27601</PostalCode>
    <PostalCode Item="2" Type="UNFORMATTED">1356</PostalCode>
    <Province Item="1" Type="COUNTRY_STANDARD">NC</Province>
    <Street Item="1" Type="COMPLETE">Fayetteville Street</Street>
    <Number Item="1" Type="COMPLETE">133</Number>
    <SubBuilding Item="1" Type="COMPLETE">Suite 201</SubBuilding>
    <Organization Item="1" Type="COMPLETE">AddressDoctor</Organization>
  </AddressElements>
</InputData>
```

Check chapter 6.7 for details on how to process XML input using `AD_SetInputDataXML()`. In this XML example all "Type" attributes (see chapter 5.10) correspond to their default values, so omitting them would yield the same processing results.

The following table gives examples what certain AddressElement Items would typically contain:

AddressElement	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService	PO Box	Bag Nr. / Postoffice				
SubBuilding	Entrance	Floor	Room			
Building	Estate/Block Name	Building Name				
Street	Primary Street	Secondary Street / RR	reserved for TWN	reserved for TWN		
Number	HNO	RR Box Nr.	reserved for TWN	reserved for TWN		
Province	Province	SubProvince	Administrative district			
PostalCode	PostalCode					
Locality	City	District	Suburb	reserved for TWN	reserved for TWN	
Country	Mother Country	Territory				
Key	Key1	Key2	Key3			

Legend:

- Typically needed for correct global address representation
- Typically not populated for a correct address\*
- Not available through the AddressDoctor 5 API
- Will be needed for future country support
- Not address relevant (will be copied over to output)

\*but may contain certain input elements copied over to output

The sequence of AddressElement Items is hierarchical, as defined by reference data: Depending on reference data detail, some empty Items may thus be followed by ones filled again. Consequently, the item hierarchy on output is only really meaningful for AddressElements for which reference data is available (typically the ones that are postally relevant, like locality). Please make sure to check the "ElementResultsStatus" described in chapter 5.23.2 and "ElementRelevance" in chapter 5.23.3 to decide whether the hierarchical sequence of the output has been retained from input (parsing only, see chapter 5.4) or was adjusted based on reference data (parsing and validation, see chapter 5.5).

A few AddressElement Item output examples for countries with remarkable AddressElement names can be found in Appendix 10.4. For a more complete introduction to international addresses and their address elements see the "The Global Source Book for Name and Address Data Management" by Graham Rhind: <http://www.grcdi.nl/book2.htm>

## 5.10 Address Item Types

Normally, each AddressElement Item number should only occur once, although some address elements may contain several logically separate sub-elements called Item Types: For example items of the type TITLE, FIRST\_NAME, MIDDLE\_NAME, LAST\_NAME and FUNCTION may be assigned to the sub-elements of each "Contact" address element at the same time, while there is little sense in assigning both, the FORMATTED and UNFORMATTED type, for a "PostalCode" address element – also see the InputData.xml examples in chapter 5.9 and chapter 6.7.2.

For the majority of address elements the default item type is COMPLETE, which may be conveniently used whenever no separation of address elements into logically separate sub-elements is available for

the input data (see chapter 6.7.2 on fielded assignment of address elements and the InputData.xml DTD in chapter 10.1 for valid item types): For instance you may assign "Paris Cedex 11" either in one piece, as Locality item 1, Type "COMPLETE" or separately, with "Paris" as Locality Item 1, Type "NAME" and "Cedex 11" as Locality Item 1, Type "SORTING\_CODE".

The following is a list of valid Item/Type input combinations when the respective default type is used:

Key:	RECORD_ID & TRANSACTION_KEY may be set concurrently
Organization:	COMPLETE & DEPARTMENT may be set concurrently
Contact:	COMPLETE & FUNCTION & GENDER may be set concurrently (when NAME is used instead, FIRST_NAME, MIDDLE_NAME and LAST_NAME may not be set - see chapter 6.7.2 for more detail)
Province:	COUNTRY_STANDARD, ABBREVIATION & EXTENDED may be set concurrently
PostalCode:	FORMATTED & UNFORMATTED may be set concurrently
Street:	COMPLETE & ADD_INFO* may be set concurrently
Locality:	COMPLETE only
Number:	COMPLETE & ADD_INFO* may be set concurrently
Building:	COMPLETE only
SubBuilding:	COMPLETE only
DeliveryService:	COMPLETE & ADD_INFO* may be set concurrently

It is recommended to refrain from setting "Type" attributes explicitly on input, apart from the examples given above and in chapter 6.7.2: Omitting them corresponds to their default values, which yields decent processing results in most practical situations. Under special circumstances, input item types might need to be adjusted under direction from AddressDoctor support (see chapter 9.3). Please note that the majority of types documented in the InputData.xml DTD (see chapter 10.1) is only listed for reasons of symmetry with the Result.xml DTD and not really intended for actual use on input.

For an overview and explanation of what item types are available on output, please refer to the Result.xml DTD (see chapter 10.1).

## 5.11 Process Modes

The AddressDoctor engine supports several validation types. Most of them are country independent and work for all supported countries. An exception is the CERTIFIED validation type that offers country specific logic and does not work for all countries. Each validation type was designed for a specific task. The four validation process modes are:

---

\* While ADD\_INFO could in principle be used to provide additional information on AddressElement input that is supposed to be passed through validation without change, it is really intended to be filled on output, containing portions (provided such a split-off could be determined) of not postally relevant AddressElement input that could not be validated against reference data.

- Correction Only (BATCH)
- Suggestions (INTERACTIVE)
- Fast Completion (FASTCOMPLETION)
- Certified (CERTIFIED)

The process mode is a parameter to the `AD_Process()` function of the AddressDoctor. When calling that function to process an `AddressObject`, the validation type that was supplied (see chapter 6.3) via `AD_SetParametersXML()` before the call determines the processing that takes place. Each `AddressObject` and thus each individual call of the `AD_Process()` function may use a different validation type. Additionally, two more process modes bypassing validation are available for special pre-processing purposes:

- Obtain separate tokens (possibly including transliteration) from parsed input data without corrections (PARSE)
- Identification of records missing country information, including correction where possible (COUNTRYRECOGNITION)

See the figure in Appendix 10.3 for more details on the AddressDoctor processing flow. Please note that process modes might fall back to others as described below, so it is recommended practice to check that the process mode used was the one intended, both, by interpreting the process status value (see chapter 5.15) and checking directly (see chapter 6.10).

### 5.11.1 Batch

The Correction Only (also known as “BATCH”) type is intended to be used in batch processing environments when no human input or selection is possible. It is optimized for speed and will terminate its attempts to correct an address when ambiguous data is encountered that cannot be corrected automatically. The Batch processing mode will fall back to Parse Only (see chapter 5.11.5), when the respective database is missing for a specific country.

### 5.11.2 Interactive

When working in interactive environments, it is often useful to generate suggestions when an address input is ambiguous. This can be achieved by using a suggestions validation type, that is known as “INTERACTIVE”. This validation type is especially useful in Web based data entry environments when capturing data from customers or prospects. It requires the input of an almost complete address and will attempt to validate or correct the data provided. If ambiguities are detected, this validation type will generate up to 20 suggestions that can be used for pick lists. The Interactive processing mode will fall back to Parse Only (see chapter 5.11.5), when the respective database is missing for a specific country.

### 5.11.3 Fast Completion

The Fast Completion validation type is used in quick address entry applications. It allows input of truncated data in several address fields and will generate suggestions for this input. Due to its fast response time, the engine can also be used to create suggestions while users type. The Fast Completion type is best suited when users are aware that they can purposely truncate input data. However, the

“FASTCOMPLETION” validation type does not support extended parsing and thus can only be used when assigning fielded input data (see chapter 6.7) using the “AddressElement” Items (see chapter 5.9) – as opposed to AddressLine input using “FormattedAddressLine” or “DeliveryAddressLine”. The Fast Completion processing mode will fall back to Parse Only (see chapter 5.11.5), when the respective database is missing for a specific country.

## 5.11.4 Certified

A number of countries have special requirements for the processing of addresses from their countries. An example of such a special processing requirement is the CASS certification of the United States Postal Service (USPS). In order to process addresses compliant with a certification scheme, the validation type “CERTIFIED” is available. At this time, CASS certification for the USA, SERP certification for CAN and AMAS certification for AUS are supported. The Certified processing mode will fall back to Batch if it is not supported for a specific country. Please note that extended parsing of unfielded data is not supported in the “CERTIFIED” processing mode (see chapters 5.5 and 6.24 for the differences between certified and normal processing).

## 5.11.5 Parse Only

For separating address input into tokens for subsequent processing in other systems, bypassing AddressDoctor validation, the special process mode “PARSE” can be used (see chapter 5.4 for a general introduction on parsing and chapter 6.9 for an example of address parsing).

A typical use case scenario for this mode might be that address data of already high quality simply needs to be tokenized quickly for export to an external system, possibly including transliteration (see the “PreferredScript” parameter in chapter 5.12.1), formatting (see chapter 5.13) and standardization (see chapter 5.14) of the output.

## 5.11.6 Country Recognition

Sometimes input data lacks country information, which is crucial for successful AddressDoctor processing. To identify such problematic records quickly, without having to run the data set through full validation, a special process mode “COUNTRYRECOGNITION” is provided. This functionality is the first step of the AddressDoctor processing flow and thus part of all process modes (see Appendix 10.3). Please refer to chapter 5.12.3 for means of providing missing country information.

This process mode will also attempt to amend missing country information where possible, based on characteristic information like major locality or territory names (see chapter 5.15 for the possible Rx process status values). Please note that such attempts at adding country information can only succeed where the information identified is absolutely unambiguous: For example, there is a Berlin in Germany as well as in numerous US states or South Africa, Columbia and El Salvador. Also, “MA” might refer to the ISO2 code for Morocco or the US state of Massachusetts.

## 5.12 Process Parameters

Numerous parameters pertaining to processing may be specified through Parameters.xml, see the DTD definition in the Appendix (chapter 10.1). These parameters may usually be defined with a global AddressDoctor scope or a per AddressObject scope, an example is given in chapter 6.3.

### 5.12.1 The PreferredScript Parameter

The “PreferredScript” attribute of the “Result” element is used to specify in which alphabet the output should be returned (see the Character Set Mapping and Transliteration chapters 5.1 & 5.2):

DATABASE	Latin I or ASCII characters (as per reference database standard)
POSTAL_ADMIN_PREF	Latin I or ASCII characters (as preferred by local postal administration)
POSTAL_ADMIN_ALT	Latin I or ASCII characters (local postal administration alternative)
ASCII_SIMPLIFIED	ASCII characters
ASCII_EXTENDED	ASCII characters with expansion of special characters (e.g. Ö = OE)
LATIN	Latin I characters
LATIN_ALT	Latin I characters (alternative transliteration)

The default setting for the “PreferredScript” attribute is “DATABASE”. The alphabet in which the data is returned differs from country to country. For most countries the output will be Latin I regardless of the selected preferred language.

For countries that use an alphabet other than Latin I, the returned alphabet differs from country to country. The following table shows how the output is returned for specific countries:

Country	DATABASE	POSTAL_ADMIN_PREF	POSTAL_ADMIN_ALT	LATIN	LATIN_ALT	ASCII_SIMPLIFIED	ASCII_EXTENDED
RUS	Cyrillic	Cyrillic	Cyrillic	CYRILLIC_ISO	CYRILLIC_BGN	CYRILLIC_ISO + LATIN_SIMPLE	CYRILLIC_ISO + LATIN
JPN	Kanji	Kanji	Kana	JAPANESE	JAPANESE	JAPANESE + LATIN_SIMPLE	JAPANESE + LATIN
CHN	Hanzi	Hanzi	Hanzi	CHINESE_MANDARIN	CHINESE_CANTONESE	CHINESE_MANDARIN + LATIN_SIMPLE	CHINESE_MANDARIN + LATIN
HKG	Hanzi	Hanzi	Hanzi	CHINESE_CANTONESE	CHINESE_MANDARIN	CHINESE_CANTONESE + LATIN_SIMPLE	CHINESE_CANTONESE + LATIN
TWN	Hanzi	Hanzi	Hanzi	CHINESE_CANTONESE	CHINESE_MANDARIN	CHINESE_CANTONESE + LATIN_SIMPLE	CHINESE_CANTONESE + LATIN
GRC	Greek	Greek	Greek	GREEK_ISO	GREEK_BGN	GREEK_ISO + LATIN_SIMPLE	GREEK_ISO + LATIN
KOR	Latin	Hangul	Hanja	KOREAN	KOREAN	KOREAN + LATIN_SIMPLE	KOREAN + LATIN
ISR	Latin	Hebrew	Hebrew	HEBREW	HEBREW	HEBREW + LATIN_SIMPLE	HEBREW + LATIN
ROM	Latin-3	Latin-3	Latin-3	Latin-3	Latin-3	LATIN_SIMPLE	LATIN
POL	Latin-2	Latin-2	Latin-2	Latin-2	Latin-2	LATIN_SIMPLE	LATIN
CZE	Latin-2	Latin-2	Latin-2	Latin-2	Latin-2	LATIN_SIMPLE	LATIN
CRI	Latin-2	Latin-2	Latin-2	Latin-2	Latin-2	LATIN_SIMPLE	LATIN
HUN	Latin-2	Latin-2	Latin-2	Latin-2	Latin-2	LATIN_SIMPLE	LATIN
MDA	Latin-2	Latin-2	Latin-2	Latin-2	Latin-2	LATIN_SIMPLE	LATIN
SVK	Latin-2	Latin-2	Latin-2	Latin-2	Latin-2	LATIN_SIMPLE	LATIN
LAT	Latin-7	Latin-7	Latin-7	Latin-7	Latin-7	LATIN_SIMPLE	LATIN

Countries not listed in the table use the default output setting described previously. Examples using different scripts can be found in chapter 6.15.

### 5.12.2 The PreferredLanguage Parameter

The “PreferredLanguage” attribute of the “Result” element is used to specify in which language the output should be returned. The default setting for “PreferredLanguage” is “DATABASE”.

The alphabet in which the data is returned differs from country to country (see 5.12.1), but for most countries the output will be Latin, regardless of the selected preferred language:

DATABASE	Language derived from reference data for each address.
ENGLISH	English Locality and Province name output, if available.
ALTERNATIVE_1,2,3	Reserved for later use, e.g. for multi-language countries like Canada, Belgium or Switzerland. If no alternative is provided as part of the postal reference data, this setting will revert to the default "DATABASE".

### 5.12.3 The ForceCountryISO3 and DefaultCountryISO3 Parameters

The "ForceCountryISO3" and "DefaultCountryISO3" attributes of the "Input" element allow a certain degree of influence on country recognition. While "ForceCountryISO3" will cause address records to be always treated as originating from the country set here (thus overriding any explicitly assigned country element), "DefaultCountryISO3" will only apply to records lacking such explicit country information.

### 5.12.4 The CountryType and CountryofOriginISO3 Parameters

The "CountryofOriginISO3" and "CountryType" attributes of the "Result" element are used to control country information output. While "CountryOfOriginISO3" will cause country information output to be suppressed for address records originating from the country set here, "CountryType" will determine in which format country information will be output.

Some possible values for "CountryType" are (see the DTD in chapter 10.1 for a complete list, the default is "NAME\_EN"):

ABBREVIATION	ISO_2
ISO_3	ISO_NUMBER
NAME_CN	NAME_DA
NAME_DE	NAME_EN
NAME_ES	NAME_FI
NAME_FR	NAME_GR
NAME_HU	NAME_IT
NAME_JP	NAME_KR
NAME_NL	NAME_PL
NAME_PT	NAME_RU
NAME_SA	NAME_SE

### 5.12.5 The MatchingAlternatives and MatchingScope Parameters

The "MatchingAlternatives" and "MatchingScope" attributes of the "Process" element are used to influence the matching of address elements during validation.

While “MatchingAlternatives” allows suppressing the use of historical and synonym (or, more precisely, exonym - see <http://wikipedia.org/wiki/Exonym>) data for matching address elements (NONE, SYNONYM\_ONLY, ARCHIVE\_ONLY, with a default of ALL), setting “MatchingScope” other than the default “ALL” will reduce the granularity of address elements (see chapter 5.3) for which matching must succeed, i.e. “LOCALITY\_LEVEL” will only consider matches on province, locality and postcode level, while “STREET\_LEVEL” extends matching to streets and “DELIVERYPOINT\_LEVEL” finally adds house number and sub building matching.

Please refer to the DTD in chapter 10.1 for a complete list of valid attribute values, these attribute settings may not have an effect for countries lacking the necessary level of detail in the postal reference data.

## 5.13 Output Formatting

The AddressComplete (see chapter 6.7.4 as well) multi-line output format (for a global reference of postal address formats see [http://www.addressdoctor.com/en/countries\\_data/addressformats.asp](http://www.addressdoctor.com/en/countries_data/addressformats.asp)) generated by AddressDoctor when processing may be modified and adjusted from the standard behavior by setting parameters provided in the “Result” element of Parameters.xml (see the DTD in Appendix 10.1) passed via `AD_SetParametersXML()`.

The “Result” element allows formatting control over:

- The type of format through “FormatType”
- The delimiter used in output formatting through “FormatDelimiter”
- Choosing whether the country is included through “FormatWithCountry”
- Number of lines through “FormatMaxLines”

Valid settings for “FormatType” are (default is “ALL”): ALL, ADDRESS\_ONLY, WITH\_ORGANIZATION, WITH\_CONTACT, WITH\_ORGANIZATION\_CONTACT or WITH\_ORGANIZATION\_DEPARTMENT

“FormatDelimiter” is from a choice of (default is “CRLF”): CRLF, LF, CR, SEMICOLON, COMMA, TAB, PIPE or SPACE.

“FormatMaxLines” determines the maximum number of overall address lines returned in a range of 1-19 (the default is 19) and “FormatWithCountry” may be switched “ON”, from the default “OFF”.

These formatting parameters are available both, as attributes of the Input element (if the input is provided in multi-line fashion), as well as attributes of the “Result” element, that are applied unless the “AddressComplete” attribute of the “Result” element is set to “OFF” (from the default “ON”).

## 5.14 Output Standardization

The output generated by AddressDoctor when processing addresses follows the rules of the postal administrations and the Universal Postal Union (UPU).

It is possible to modify and adjust the standard output behavior by setting attributes provided in the "Result" element of Parameters.xml (see the DTD in Appendix 10.1) passed via `AD_SetParametersXML()`.

The "Result" element allows standardization control over:

- Element length (by means of abbreviation) through "GlobalMaxLength"
- Casing through "GlobalCasing"
- Abbreviation through "ElementAbbreviation"

While the "GlobalMaxLength" attribute determines the default maximum number of characters per line for all address elements, "FormatMaxLines" (see the previous chapter 5.13) determines the maximum number of overall address lines returned in the case of multi-line "AddressComplete" output. Please note that the default value for "GlobalMaxLength" is 1024.

The "GlobalCasing" attribute can be used to influence the casing of the output. The five possible options are native casing as per reference database standard (NATIVE), upper casing (UPPER), lower casing (LOWER), mixed casing (MIXED) and unchanged (NOCHANGE).

While upper casing and lower casing will create an output independent of the country the data is in, mixed casing will consider country specific rules, while the default native casing will be based on the reference database content. Setting the casing to be unchanged will return the data the way it was entered for the output of PARSE process mode, while validated results will be provided as found in the reference data and according to postal rules. Result address elements that could not be checked against the reference data will retain their input casing when NOCHANGE is set.

Additionally, standardization may also be defined per address element via the "MaxLength" and "Casing" attributes of the "AddressElementStandardize" element, thus overriding the global settings: "MaxLength" is then used to set the maximum characters per line for each address element. Setting "MaxLength" to 0 will inherit the length configured globally.

Each address element has a sensible allowed minimum length. Valid minimum values for "MaxLength" are as follows:

Organization:	25	Street:	20
Department:	25	Number:	5
Contact:	25	DeliveryService:	25
First Name:	20	Locality:	20
Middle Name:	20	PostalCode:	5
Last Name:	20	Province:	2
Title:	20	Country:	2
Function:	20	CountrySpecificLocalityLine:	25
Salutation:	20	DeliveryAddressLine:	25

Gender:	1	RecipientLine:	25
Building:	25	FormattedAddressLine:	25
Sub-Building:	25	AddressComplete:	25

Please note that depending on the data and the country selected return values might still exceed the selected minimum value. This happens if there is no useful way to abbreviate the values further. An example would be to abbreviate the following postal code from United Kingdom "AB123AD" to 5 characters. The return value will still be contains 7 digits: "AB123AD".

Release 5.2.7 introduces the new Parameter "ElementAbbreviation". At the moment this parameter will only influence data from the USA.

In CERTIFIED mode, setting the parameter to "ON" while having all CASS relevant databases available will abbreviate street and locality if the output for a validated or corrected address would be longer than the defined length by USPS. Setting it to "OFF" will return the values based on input, field length setting and database entries.

In BATCH and INTERACTIVE mode, setting the parameter to "ON" will only abbreviate locality if the output would be longer than the defined length by USPS.

## 5.15 Process Status Values

The process status values returned by `AD_GetResultXML()` Or `AD_GetResultParameter()` describe the result output quality of a `Process()` call to AddressDoctor in an overview fashion (for more detailed information on results consult the "ElementResultStatus" as well, see chapter 5.23.2):

- V4 Verified – Input data correct - all (postally relevant) elements were checked and input matched perfectly
- V3 Verified – Input data correct on input but some or all elements were standardised or input contains outdated names or exonyms
- V2 Verified – Input data correct but some elements could not be verified because of incomplete reference data
- V1 Verified – Input data correct but the user standardisation has deteriorated deliverability (wrong element user standardisation – for example, postcode length chosen is too short). Not set by validation.
- C4 Corrected – all (postally relevant) elements have been checked
- C3 Corrected – but some elements could not be checked
- C2 Corrected – but delivery status unclear (lack of reference data)
- C1 Corrected – but delivery status unclear because user standardisation was wrong. Not set by validation.
- I4 Data could not be corrected completely, but is very likely to be deliverable – single match (e.g. HNO is wrong but only 1 HNO is found in reference data)
- I3 Data could not be corrected completely, but is very likely to be deliverable – multiple matches (e.g. HNO is wrong but more than 1 HNO is found in reference data)
- I2 Data could not be corrected, but there is a slim chance that the address is deliverable

- I1 Data could not be corrected and is pretty unlikely to be delivered.
- N1 Validation Error: No validation performed because country was not recognized
- N2 Validation Error: No validation performed because required reference database is not available
- N3 Validation Error: No validation performed because country could not be unlocked
- N4 Validation Error: No validation performed because reference database is corrupt or in wrong format
- N5 Validation Error: No validation performed because reference database is too old – please contact AddressDoctor to obtain updated reference data
- Q3 FastCompletion Status – Suggestions are available – complete address
- Q2 FastCompletion Status – Suggested address is complete but combined with elements from the input (added or deleted)
- Q1 FastCompletion Status – Suggested address is not complete (enter more information)
- Q0 FastCompletion Status – Insufficient information provided to generate suggestions
- S4 Parsed perfectly
- S3 Parsed with multiple results
- S2 Parsed with Errors – Elements change position
- S1 Parse Error – Input Format Mismatch
- RB Country recognized from Abbreviation
- RA Country recognized from ForceCountryISO3 Setting
- R9 Country recognized from DefaultCountryISO3 Setting
- R8 Country recognized from name without errors
- R7 Country recognized from name with errors
- R6 Country recognized from territory
- R5 Country recognized from province
- R4 Country recognized from major town
- R3 Country recognized from format
- R2 Country recognized from script
- R1 Country not recognized - multiple matches
- R0 Country not recognized

The Vx,Cx and Ix Process Status values may be returned “BATCH”, “INTERACTIVE” or “CERTIFIED” `Process()` calls, while Qx is only returned for “FASTCOMPLETION”, Sx for “PARSE” and Rx for “COUNTRYRECOGNITION” (see chapter 5.11 for details on the different Process Modes). Nx Process Status values may be returned for any `Process()` call.

Please note that for BATCH processing it is strictly recommended to only accept records with Vx or Cx status for automated data updates. Ix records need to be reviewed manually before using these results for any data update whatsoever.

## 5.16 Mailability Scores

AddressDoctor provides an estimate of how likely successful delivery of mail to an address might be. This is a simplification of the process status values (see chapter 5.15) and gives a measure to determine whether an address should be bothered with for mailing in a specific usage scenario:

- 5: Completely Confident
- 4: Almost Certain
- 3: Should Be Fine
- 2: Fair Chance
- 1: Risky
- 0: Undeliverable

Addresses with a mailability of 5 and 4 may always be considered for sending mail, while 0 or 1 should not be used independent of the scenario. Addresses marked with 2 or 3 may be used, but should be treated with caution: 2 indicates that the results are not corrected and therefore may still contain an incorrect address component. 3 indicates a correction which may require a review before sending the mail piece.

If there is a requirement to understand exactly what was validated or corrected in the address, the `ProcessStatusValue`, `ElementInputStatus` and `ElementResultStatus` fields should be used instead of the Mailability Score.

### 5: Completely Confident

All relevant elements of the address that have been entered were checked in the processing and have been verified in the process.

### 4: Almost Certain

An address is considered to be *Almost Certain* when one of the following two scenarios is present. Scenario 1: Some of the relevant elements of the address could not be checked due to reference data and the rest of the address have been verified in the process. Scenario 2: All relevant elements have been entered and some of the relevant elements of the address have been corrected in the process with a very high confidence. This only happens if the match was unique and the number of discrepancies was very low.

### 3: Should Be Fine

Some of the relevant elements of the address have been corrected in the process. A correction only happens if the match was unique and the number of discrepancies was acceptable.

### 2: Fair Chance

The address could not be corrected or validated in the process based on two scenarios. Scenario 1: A candidate match could not be made that had sufficient confidence. Scenario 2: The address matching ended with multiple candidates with similar confidence levels (multi-match situation). The input address, therefore, has a *Fair Chance* to be mailable as the relevant elements exist.

1: Risky

The address entered could only generate a partial match.

0: Undeliverable

The address entered is either missing too many components or a majority of the components could not be verified as they generate no matches against the reference data.

## 5.17 Geocoding Status Values

AddressDoctor 5 introduces Geocoding for selected countries: This means the Version 5 API will provide the option to enrich a validated address by the respective geo-coordinates in WGS84 (<http://wikipedia.org/wiki/WGS84>) format.

The quality of coverage will naturally vary from country to country and while AddressDoctor shall strive to provide geo-coordinates on house number or building level, depending on data availability, only street or even locality level geo-coordinates might be available.

The corresponding status values returned with the processing result via `AD_GetResultXML()` or `AD_GetResultParameter()` are:

EGCU:	Geocoding database not unlocked
EGCN:	Geocoding database not found
EGCC:	Geocoding database corrupt
EGC0:	No Geocode available
EGC1..3:	Reserved for future use
EGC4:	Geocode with partial postal code level accuracy (e.g. 795xx)
EGC5:	Geocode with postal code level accuracy
EGC6:	Geocode with locality level accuracy
EGC7:	Geocode with street level accuracy
EGC8:	Geocode with house number level accuracy (interpolated approximation)
EGC9:	Reserved for future use

## 5.18 CASS Status Values

AddressDoctor 5 provides the output required by the USPS CASS Standard, see chapter 6.24 for details on the actual output available. The corresponding status values returned with the processing result via `AD_GetResultXML()` Or `AD_GetResultParameter()` are:

ECA0:	CASS output not available (for this address)
ECA1:	CASS attributes only partially provided (some databases are missing)
ECA2..4:	Reserved for future use
ECA5:	CASS attributes provided

## 5.19 SERP Status Values

AddressDoctor 5 provides the output required by the Canada Post SERP Standard, see chapter 6.24 for details on the actual output available. The corresponding status values returned with the processing result via `AD_GetResultXML()` Or `AD_GetResultParameter()` are:

ESE0:	SERP output not available (for this address)
ESE1:	SERP attributes provided

If the Validation type is CERTIFIED and the SERP Enrichment Status is ON, two enrichments are provided: CATEGORY and EXCLUDED\_FLAG. For details, see chapter 6.24.2.

## 5.20 SNA Status Values

AddressDoctor 5 provides the output required by the La Poste SNA Standard, see chapter 6.24 for details on the actual output available. The corresponding status values returned with the processing result via `AD_GetResultXML()` Or `AD_GetResultParameter()` are:

ESN0:	SNA output not available (for this address)
ESN1:	SNA attributes provided

## 5.21 AMAS Status Values

AddressDoctor 5 provides the output required by the Australia Post AMAS Standard, see chapter 6.24 for details on the actual output available. The corresponding status values returned with the processing result via `AD_GetResultXML()` Or `AD_GetResultParameter()` are:

EAM0:	AMAS output not available (for this address)
EAM1:	AMAS output is provided – Address is corrected or validated and DPID is delivered
EAM2:	AMAS output is not provided – No correction or validation is possible – No DPID can be returned

## 5.22 Country Specific Enrichment

For selected countries, AddressDoctor 5 provides additional enrichment output required by the local markets, see [chapter 5.22.2](#) and chapter 6.13 for details on the actual output available.

### 5.22.1 Country Specific Enrichment Status Values

The corresponding status values returned with the processing result via `AD_GetResultXML()` or `AD_GetResultParameter()` are:

For USSupplementary:

EUS0: US country specific output not available (for this address)  
EUS1: US country specific attributes provided (not necessarily all attributes are populated)

For GBSupplementary:

EGB0: GB country specific output not available (for this address)  
EGB1: GB country specific attributes provided (not necessarily all attributes are populated)

### 5.22.2 Country Specific Enrichment Output fields

The following output fields are currently supported:

For USSupplementary:

**COUNTY\_FIPS\_CODE:**  
3 digit number identifying a county in the United States. The United States Federal Information Processing Standard (FIPS) maintains a set of codes that identify states, counties, and other territorial possessions. The two-digit state code identifies each state. The three-digit county code identifies a county within a state. The five digits of the state and county code can uniquely identify any county

**STATE\_FIPS\_CODE:**

2 digit number identifying states in the United States. The Federal Information Processing Standard (FIPS) controls the numerical and alphabetical codes that identify states and other territories of the United States.

**MSA\_ID:**

The Metropolitan Statistical Area identification number (MSAID) is a 4 digit number that identifies an urban area with a population greater than 50,000.

**CBSA\_ID:**

Represents a Core-Based Statistical Area (CBSA) identification number. A CBSA identifies an urban area with a population greater than 10,000. A CBSA can be a Metropolitan Statistical Area or Micropolitan Statistical Area. A Metropolitan Statistical Area has over 50,000 inhabitants. A Micropolitan Statistical Area has between 10,000 and 50,000 inhabitants. A CBSA\_ID is a 5 digit number.

**FINANCE\_NUMBER:**

A finance number has six digits. The output is a code assigned to United States post offices and other postal facilities to enable collection of cost and statistical data. The first two digits of the finance number identify the state. The final four digits identify the USPS post office or postal facility.

**RECORD\_TYPE:**

A single-character code that describes the type of a mailbox or delivery. For example, the code can indicate if the address is in a high-rise building (value H) or a post office box (value P).

**CMSA\_ID:**

Represents a Consolidated Metropolitan Statistical Area (CMSA) identification number. A PMSA becomes a CMSA if local opinion favors the designation. The CMSA\_ID is a 4 digit unique number.

**TIME\_ZONE\_CODE:**

1 to 3 characters numerical value identifying the difference to GMT. Example would be "-5" for Eastern Standard Time.

**TIME\_ZONE\_NAME:**

3 Characters identifying the time zone the address is in like "EST" Eastern Standard Time

**CENSUS\_TRACT\_NO:**

Census Tract is a statistical subdivision of a county. A 6 digit number identifies the CENSUS\_TRACT\_NO.

**CENSUS\_BLOCK\_NO:**

Census Block is the smallest entity for which the Census bureau collects census information. The CENSUS\_BLOCK\_NO is a 4 digit number.

**CENSUS\_BLOCK\_GROUP:**

A Census Block Group is a group of Census blocks sharing the same first digit.

**PMSA\_ID:**

Represents a Primary Metropolitan Statistical Area (PMSA) identification number. Two or more PMSA are created if a MSA reaches a size of 1 million or more people. The PMSA\_ID is a 4 digit unique number.

**MCD\_ID:**

Represents a Minor Civic Division which is a primary legal subdivision of a county defined by the Government. The MCD\_ID is a 5 digit number.

**PLACE\_FIPS\_CODE:**

5 digit number identifying localities in the United States. The Federal Information Processing Standard (FIPS) controls the numerical codes that identify localities in the United States.

For GBSupplementary:

**DELIVERY\_POINT\_SUFFIX:**

The Royal Mail assigns a two-character suffix to every mailbox in a UK post code area. It uses the post code and delivery point suffix to identify every mailbox.

The delivery point suffix format is a digit followed by a letter.

## 5.23 Element Status and Relevance Values

Element status values give a detailed explanation of the outcome of the validation operation. They are only meaningful after a validation operation has been performed, even though some information is available after a parsing operation for the “ElementInputStatus” value.

In AddressDoctor 5 now 20 address elements are covered in both, “ElementInputStatus” and “ElementResultStatus”. The former provides per element information on the matching of input elements to reference data, while the latter categorizes the result in more detail than the overview process status values described in section 5.15 (by indicating if and how the output fields have been changed from the input fields).

The element positions (from left to right) are, where level 0 pertains to the Item 1 status information, while level 1 summarizes the status information on Items 2-6 (see chapter 5.9 on address element items):

- 1: PostalCode level 0
- 2: PostalCode level 1 (e.g. ZIP+4 – Plus 4 addition)
- 3: Locality level 0
- 4: Locality level 1 (e.g. Urbanisation, Dependent Locality)
- 5: Province level 0
- 6: Province level 1 (e.g. Sub Province)
- 7: Street level 0
- 8: Street level 1 (e.g. Dependent street)
- 9: Number level 0
- 10: Number level 1
- 11: Delivery service level 0 (e.g. PO Box, GPO, Packstation, Private Bags)
- 12: Delivery service level 1
- 13: Building level 0
- 14: Building level 1
- 15: SubBuilding level 0
- 16: SubBuilding level 1
- 17: Organisation level 0
- 18: Organisation level 1
- 19: Country level 0 (Mother country)
- 20: Country level 1 (e.g. Territory)

Please see chapter 5.3 for more in-depth information on address elements.

### 5.23.1 ElementInputStatus

The possible values for validation are:

- 0: empty
- 1: not found
- 2: not checked (no reference data)
- 3: wrong - Set by validation only: The reference database suggests that either Number or DeliveryService is out of valid number range. Input is copied, not corrected for batch mode, for interactive mode and fast completion suggestions are provided.
- 4: matched with errors in this element
- 5: matched with changes (inserts or deletes)
  - For example:
    - Parsing: Splitting of house number for “MainSt 1”
    - Validation: Replacing input that is an exonym or dropping superfluous fielded input that is invalid according to the country reference database
- 6: matched without errors

For parsing, the following values are possible:

- 0: empty
- 1: element had to be relocated
- 2: matched but needed to be normalized
- 3: matched and OK

### 5.23.2 ElementResultStatus

Is only set after validation as an indication whether verification (“verified”) or correction (“changed”) were possible or not, the potential values are (for all address elements apart from country):

- 0: empty
- 1: not validated and not changed. Original is copied.
- 2: not validated but standardized.
- 3: validated but not changed due to invalid input, database suggests that number is out of valid ranges. Input is copied, not corrected – this status value is only set in batch mode.
- 4: validated but not changed due to lack of reference data.
- 5: validated but not changed due to multiple matches. Only set in batch mode, otherwise multiple suggestions that replace the input are marked as corrected (status value 7).
- 6: validated and changed by eliminating the input value
- 7: validated and changed due to correction based on reference data
- 8: validated and changed by adding value based on reference data

- 9: validated, not changed, but delivery status not clear (e.g. DPV value wrong; given number ranges that only partially match reference data).
- C: validated, verified but changed due to **outdated** name
- D: validated, verified but changed from exonym to official name
- E: validated, verified but changed due to standardization based on casing or language. Validation only sets this status if input fully matches a language alternative.
- F: validated, verified and not changed due to perfect match

For Country (position 19 & 20), the following values are possible (which specifically apply to the process mode COUNTRYRECOGNITION as well, see chapters 5.11.6 and 5.12.3):

- 0: empty
- 1: Country not recognized
- 4: Country recognized from DefaultCountryISO3 setting
- 5: Country not recognized - multiple matches
- 6: Country recognized from script
- 7: Country recognized from format
- 8: Country recognized from major town
- 9: Country recognized from province
- C: Country recognized from territory
- D: Country recognized from name with errors
- E: Country recognized from name without errors
- F: Country recognized from ForceCountryISO3 setting

### 5.23.3 ElementRelevance

In addition to the element status values described previously, information is available on which of the address elements of the address processed are actually relevant from the local postal operator's point of view. The possible values for each address element are "1" for relevant and "0" otherwise. For any given address, all address elements with a value of "1" must be present for an output address to be deemed valid by the local postal authority. "ElementRelevance" may well vary from address to address for countries with different address types, e.g. rural versus metropolitan addressing. Furthermore, AddressElements that have actually been validated against reference data (i.e. with ElementResultStatus 7 and higher) may override the default ElementRelevance value defined for that AddressElement.

Please note that "ElementRelevance" is really only meaningful for a "ProcessStatus" value of Cx or Vx (and possibly I3 & I4 for Process Mode INTERACTIVE, see chapter 5.15 for details on "ProcessStatus").

### 5.24 ResultPercentage Values

The "ResultPercentage" value gives an indication how similar a result is to the parsed input, values close to 100% imply high similarity. They are mainly provided to allow for filtering out too extensive

corrections in records with Cx BATCH “ProcessStatus” value (see chapter 5.15) in master data management environments with very stringent data quality requirements.

Also, “ResultPercentage” may be used to determine which INTERACTIVE results show the least deviation from input. AddressDoctor discourages using “ResultPercentage” values for any other use case scenarios than the two described above.

## 5.25 Return Codes

The use of the AddressDoctor engine may result in error conditions signalled via return codes.

All API functions return an AD\_I32 (32 bit signed integer) return code value:

- A value of 0 (zero) indicates success.
- A negative value of -10000 or below indicates a very critical error, further processing is usually impossible; it is strongly advised to shut down the whole process, as it may be in an instable state.
- Negative values between -1 and -9999 indicate critical errors, further processing may be impossible.
- A positive value of 1000 and above indicates non-critical errors, further processing is possible. Return code values between 1 and 999 have been assigned to warnings, indicating possible issues with configuration settings, address input or output.

The return value must always be checked for by the calling logic. While it informs about fundamental errors, the actual validation results are returned via separate API functions (see chapter 6.11).

Following are the most common error return codes, including an explanation (see the API documentation in chapter 10.2 for a complete and up-to-date list):

0        **OK**, no error

**Warnings**, the operation was completed, but maybe with an unexpected result:

- 1        The SetConfig.xml contained at least one corrupt unlock code
- 2        The SetConfig.xml contained at least one expired unlock code
- 3        The SetConfig.xml listed at least one database file which was not found
- 4        The SetConfig.xml listed at least one corrupt database file
- 5        The SetConfig.xml listed at least one database with a not supported version
- 6        The SetConfig.xml listed at least one database which is not supported (i.e. DEU CERTIFIED)
- 7        No valid unlock code for a database file
- 8        The SetConfig.xml listed at least one database at least two times
- 9        The MaxMemoryUsageMB setting in SetConfig.xml was too small to fulfill all preloading settings and/or the CacheSize setting
- 100     An input element or line which already had content was overwritten
- 101     The AddressComplete input has too many lines, extra lines will be ignored for further processing
- 200     The output buffer is too small, the output was written, but truncated

- 201 At least one character of the output could not be encoded in the chosen encoding, these characters were replaced by an underscore ('\_')
- 300 The engine usage period has expired
- 301 The unlock code for a database file has expired
- 400 Address lines and/or Address Complete were given on input; this part of the input was ignored
- 401 More than 10 lines were given via FormattedAddressLines or AddressComplete as input; the lines beyond 11 were ignored
- 900 No database at all was found, probably because the path was wrong
- 901 No database at all was opened, probably because the path was wrong and/or no valid unlock code was given
- 902 Error while attempting to open at least one of the extra CASS DBs

**Errors**, the requested operation was not executed:

- 1000 A pointer parameter was NULL
- 1001 A function parameter was 0
- 1002 A NULL pointer to an object was used (not relevant for C API)
- 1003 Two XMLs were given (as string and within a file), only one must be given
- 1004 The output buffer size is not valid (i.e. 0)
- 1005 Buffer misalignment, an `AD_WCHAR*` points to an odd address
- 1100 A parameter is out of range or illegal
- 1101 An XML string is invalid
- 1200 The character sequence of a string is not valid (i.e. contains control codes or does violate some constraint)
- 1201 The encoding parameter did not match the character size of the API call, i.e. UCS2 (16 bit) vs. char (8 bit)
- 1300 No SetConfig.xml was given as parameter for `AD_Initialize`
- 1301 The engine has already been initialized
- 1302 `AD_DeInitialize()` failed because not all AddressObjects have been released
- 1400 No AddressObject is available (all AddressObject handles have already been obtained via `AD_GetAddressObject()`)
- 1401 The passed AddressObject handle is not valid
- 1500 A database file has not been found
- 1501 A database file is invalid/corrupt
- 1502 No valid unlock code for a database file
- 1503 A database file has a non-supported version.
- 1700 The country could not be identified
- 1800 Results are available, for this reason no AO modification is allowed
- 1801 XML and direct API calls were used intermixed when setting the input data of an AddressObject
- 1802 `AD_Process()` has not been called successfully, no result is available
- 1803 The attempted operation was invalid, i.e. trying to set incompatible address elements
- 1900 The result index parameter is out of range (must be  $\geq 1$ )

1901 The output buffer is too small to hold the result, no output was written

**Critical Errors**, no further calls, except possibly `AD_Initialize()` or `AD_DeInitialize()` should be made to the engine:

- 1300 The engine has not yet been initialized, need to call `AD_Initialize()`
- 1600 No valid unlock code was given
- 1601 The engine usage period has expired
- 1602 A clock inconsistency has been detected
- 9900 A memory allocation request failed
- 9901 A file operation failed

**Very Critical Errors**, should only occur under highly adverse circumstances. No further calls, except possibly `AD_DeInitialize()` should be made to the engine - please report that you actually encountered one of these errors:

- 10000 Some unknown exception has been thrown; this event should never occur
- 10001 Some internal assertion has failed; this event should never occur
- 10002 Some internal error has been encountered; this event should never occur

## 5.26 OptimizationLevel

AddressDoctor processing allows setting the “OptimizationLevel” attribute in Parameters.xml (see the DTD in chapter 10.1) upon `AD_Initialize()` for controlling the trade-off between processing speed and quality:

- NARROW: The parser will honor input assignment strictly, with the exception of separation of House Number from Street information.
- STANDARD: The parser will separate address element more actively, e.g.
  - Province will be separated from Locality information
  - PostalCode will be separated from Locality information
  - House Number will be separated from Street information
  - SubBuilding will be separated from Street information
  - DeliveryService will be separated from Street information
  - SubBuilding will be separated from Building information
  - Locality will be separated from PostalCode information
- WIDE: Parser separation will happen similarly to STANDARD, but additionally up to 10 parsing candidates will be passed to validation for processing. Validation will widen its search tree and take additional reference data entries into account for matching.

Please note that adjusting “OptimizationLevel” might have no effect for countries that lack the postal reference data information required for the kind of separation described above.

Obviously, increasing separation granularity from NARROW to DEFAULT already consumes some processing power, but the major impact on processing speed here is from AddressDoctor validation

processing a larger search tree, thus increasing the number of data accesses and comparisons for the "OptimizationLevel" WIDE, in an attempt to make the most out of the input data given.

Thus a recommended batch usage pattern for AddressDoctor 5 would be (assuming rather low levels of address quality):

- Run a quick sweep through your data using the COUNTRYRECOGNITION process mode (see chapter 5.11.6) for separating out those records lacking country information, which might have to be amended manually before further processing them.
- Do a fast check of overall record quality using "OptimizationLevel" NARROW to identify the valid or correctable records and separate out all records that have not resulted in a V or C "ProcessStatus" value (see chapter 5.15).
- Feed those problematic records back into the AddressDoctor engine, processing them with "OptimizationLevel" WIDE to see what might be salvaged, indicated by a V, C or I4 "ProcessStatus" value.

## 5.27 Preloading

Performance is often critical when deploying AddressDoctor with large databases. Typically, the I/O subsystem is the slowest component in a system. As memory prices have fallen sharply, users can now afford machines with a lot of installed memory.

To utilize the available memory for performance optimization, the AddressDoctor engine offers the "PreloadingType" attribute for each DataBase element. It allows loading AddressDoctor reference databases (.md files) into the main memory of the computer.

The following preloading types are available

- No preloading (PreloadingType="NONE" - the default)
- Partial preloading (PreloadingType="PARTIAL")
- Full preloading (PreloadingType="FULL")

Partial preloading will load the metadata and indexing structures into memory. The reference data itself will remain on the hard drive. Partial preloading offers some performance enhancements and is an alternative when not enough memory is available to fully load the desired databases.

Full preloading will move the entire reference database into memory. This may need a significant amount of memory for countries with large databases such as the USA or the United Kingdom, but it will increase the processing speed significantly.

However, there are conditions where full preloading can have a negative impact on speed. Please see chapter 6.25 for details on this topic. Please note that the AddressDoctor engine itself requires additional memory (see chapter 2.3) in addition to the memory used for preloading.

The "PreloadingType" attribute can be set per database as a configuration parameter of the `AD_Initialize()` call of the AddressDoctor object. If no preloading type is explicitly set, the default preloading ("NONE") will be used.

With version 5.1.4, Memory Mapped Files have been introduced as the new default preloading mechanism (`PreloadingMethod="MAP"` in `SetConfig.xml`, see Appendix 10.1 for the DTD). The former preloading mechanism (`PreloadingMethod="LOAD"`) is still available for legacy and fallback use, but should not be considered for new deployments.

When using the default "MAP" method, the engine uses the file mapping mechanism of the operating system. To actually force the file contents into memory, the data is touched (read) once upon `AD_Initialize`. The "LOAD" mechanism on the other hand uses a memory allocation call and then reads the .md file data into the allocated memory block (see chapter 5.30 as well).

So in case enough physical memory is actually present, the behavior, incl. speed, is absolutely identical (although not completely: The OS will typically write-protect mapped data, thereby possibly masking certain bugs...). Specifically, in low memory conditions, the OS will either start discarding the mapped data or swap the loaded data out to disc.

Memory Mapping has two advantages over the preloading mechanism used previously:

1. In multi-process conditions (multiple processes running AddressDoctor using a common set of .md files) the operating system will load the data into main memory only once, thus sharing preloaded reference databases between separate processes.
2. The operating system will never write reference data contents to the paging file, in case of low memory conditions (but they might get dropped from the file system cache; if the data is needed later on, it is simply read from disk again).

However, due to larger alignment requirements of the OS, "MAP" will use up more virtual memory space (2-3% more for all files). As "MAP" is the default, "PreloadingMethod" may be omitted if enabling "MAP" is intended.

Since large amounts of memory may be allocated during preloading, with significant data amounts moved into memory, it might take some time to load the databases into memory. Databases will be preloaded in the order they are passed via `SetConfig.xml` (see the respective DTD in Appendix 10.1) on `AD_Initialize()`.

The following information is available through `AD_GetConfigSettingsXML()` to check which databases have been successfully preloaded after the `AD_Initialize()` call:

```
CountryISO3  
Type (BATCH_INTERACTIVE | FASTCOMPLETION | CERTIFIED | GEOCODING)
```

Path  
 Status  
 Size  
 Version  
 StartDate  
 ExpirationDate  
 UnlockStartDate  
 UnlockExpirationDate  
 ReleaseDate  
 DataDate  
 Encoding  
 PreloadingType (FULL | PARTIAL | NONE)  
 PreloadingSize

If a database could not be found or pre-loaded for some reason, the corresponding database ISO code does not have such a Database section. For all pre-loaded databases there will be a Database section which contains a "PreloadingType" attribute specifying the actual preloading type.

Resetting the preloading parameters after the `AD_Initialize()` function has been called is only possible by issuing `AD_DeInitialize()` first (preceded by `AD_ReleaseAllAddressObjects()` for releasing all AddressObjects, see chapter 6.1).

## 5.28 Caching

Caching reserves a certain portion of "MaxMemoryUsageMB" (see the SetConfig.xml DTD in Appendix 10.1) for speeding up file system lookups in reference data that has not been preloaded.

Using the "CacheSize" attribute in SetConfig.xml (passed upon `AD_Initialize()`) the amount of memory reserved in such a way may be controlled - valid settings are NONE, SMALL, LARGE. Using the standard setting of "LARGE" is always recommended, unless all reference data needed is preloaded (so that "NONE" may be used) or the memory footprint needs to be reduced via the "SMALL" or "NONE" setting. However, "NONE" should be avoided, unless memory is really extremely scarce.

The size of the cache may be determined through `AD_GetConfigSettingsXML()`: The actual size of the cache may be less than requested, if not enough memory is available (i.e. "SMALL", although "LARGE" was requested).

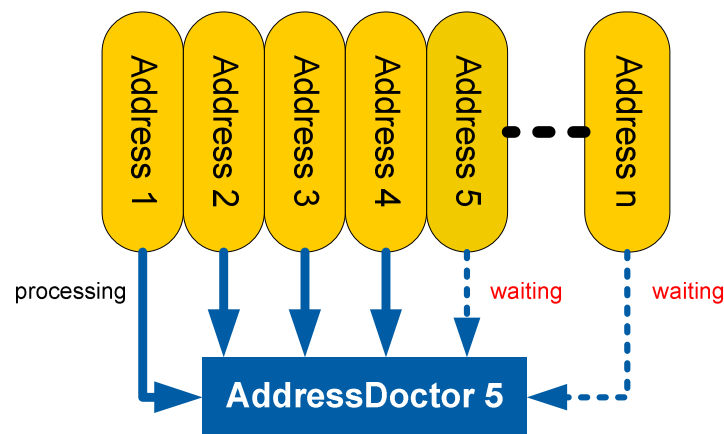
## 5.29 Multithreading

The AddressDoctor API is multi-threading-safe, any number of threads may call any of the API functions at any time without having to fear a crash due to data corruption. However, it is strictly to be avoided to call multiple API functions from different threads at the same time using the same AddressObject; such a call sequence is typically a programming error!

While AddressDoctor enables benefitting from multi-core processor architectures, the actual thread handling is strictly in the domain of the calling application: No threads are actually created or destroyed by the engine and there are no API functions to process more than one address.

The number of threads which the engine actually allows to process addresses in parallel (by calling `AD_Process()` from a separate thread per address) is configurable (the default is 1); if more threads than configured using "MaxThreadCount" (see `SetConfig.dtd`) call `AD_Process()` at the same time, the additional threads are blocked until other threads currently calling `AD_Process()` return. Please note that such blocking only influences the timing and sequence of the `AD_Process()` calls, but not the data processing and its outcome.

The figure below illustrates parallel address processing in a multi-threaded environment with  $n$  calling threads, but "MaxThreadCount" set to 4:



Currently there is no similar limitation on calls to any other API functions which have an `AddressObject` handle as parameter; however, a limit like for `AD_Process()` may be imposed in the future. However, this would be totally transparent to the calling thread(s).

"MaxThreadCount" should normally not be set to a value larger than the number of available cores/CPU's, possibly minus one (to allow for operating system overhead), as this is unlikely to increase performance. For the moment, a practical maximum value for "MaxThreadCount" of 1024 is enforced in `SetConfig.xml` (see the DTD in chapter 10.1 for reference). If the maximum number of `AddressObjects` as set by "MaxAddressObjectCount" is smaller than "MaxThreadCount", "MaxThreadCount" is internally reduced to the number specified by "MaxAddressObjectCount" as no more parallel calls to `AD_Process()` could be made anyway.

The actual value of "MaxThreadCount" can be determined by calling `AD_GetConfigSettingsXML()`. It is recommended to set "MaxAddressObjectCount" to the number of threads set with "MaxThreadCount". However, depending on the implementation, 2 `AddressObjects` per thread are necessary if a double-buffering mechanism is employed.

The largest performance gains (the best scalability) will be achieved in a multi-core environment with full preloading for all accessed databases, as otherwise the multiple threads will be blocked frequently by calls to the file system. In fact, this effect may become so dominant, that the scalability in most relevant cases will be significantly reduced, if the accessed databases are not preloaded.

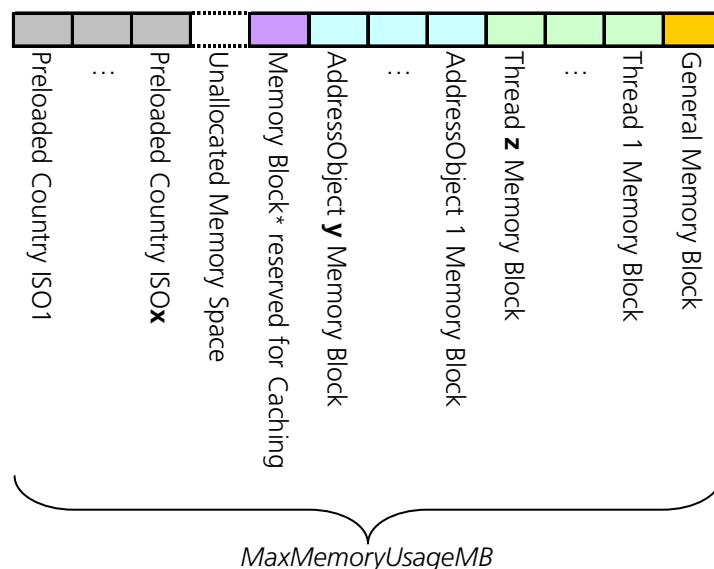
Note: The term scalability refers to the speedup factor which is achieved when trying to utilize additional cores/CPUs. Obviously the best possible speedup factor for N cores/CPUs as opposed to using only one core would be N, that is, N-times more addresses could be processed per hour when utilizing N cores/CPUs instead of one. In reality, such a perfect scaling is almost never achieved, either because only parts of the called functions can operate in parallel or there is a contention for some system resource(s) such as the front side bus, file system or memory allocation functions.

The internal design of the AddressDoctor engine will allow a good or even very good scaling if the computer system itself is designed appropriately (i.e. big large local caches for each core, fast memory buses) and blocking due to file system contention is avoided.

## 5.30 Memory Management

As described in the preceding chapters, the AddressDoctor engine handles many different types of objects in memory, like AddressObjects, pre-loaded postal reference databases, thread locks or caches.

The following figure gives a schematic overview of the memory layout used for those different object types:



The overall memory consumption may be controlled with the “MaxMemoryUsageMB” attribute, while the number of memory blocks allocated for threads  $z$  is controlled via “MaxThreadCount” (see chapter 5.29). The number of AddressObject blocks  $y$  is determined via “MaxAddressObjectCount” (see chapter 5.29) and the size of the memory block used for caching is set through the “CacheSize” attribute (see chapter 5.28). Finally, there is the number of country postal reference databases configured for pre-loading using the “PreLoadingType” attribute of the Database element (see chapter 5.27).

Please note that some memory may remain unallocated as shown in the figure above due to not pre-loading enough databases to fill up “MaxMemoryUsageMB” or a database designated for pre-loading not fitting into the remaining space. All of the attributes listed in the last paragraph may be set on initialization using SetConfig.xml (see chapter 6.1).

The general memory block size is 4 Mbytes, the size of an AddressObject memory block is about 3 Mbytes, the size of a processing memory block is 38 Mbytes for 32 bit systems and 48 Mbytes for 64 bit system plus the size of the cache block, which is 0 for CacheSize=’NONE’, 1,5 Mbytes for CacheSize=’SMALL’ and 6 Mbytes for CacheSize=’LARGE’.

However, please note that these values may change in the future.

Examples:

MaxThreadCount=’4’ MaxAddressObjectCount=’8’ CacheSize=’SMALL’ on a 32 bit system:  
Dynamic memory usage =  $4 + 8 * 3 + 4 * (38 + 1,5) = 186$  Mbytes

MaxThreadCount=’6’ MaxAddressObjectCount=’6’ CacheSize=’LARGE’ on a 64 bit system:  
Dynamic memory usage =  $4 + 6 * 3 + 6 * (48 + 6) = 346$  Mbytes

## 6 How do I...

### 6.1 ...initialize AddressDoctor?

`AD_Initialize()` Or `AD_InitializeW()` must be called to actually initialize the engine: It evaluates the settings and configures the engine accordingly (see chapter 5.6 for an overview). Only after this function has returned successfully may `AD_GetAddressObject()` or any other functions be called. If the engine was not initialized properly, all AddressDoctor API functions will produce a return code of -1300 (see chapter 5.25 for reference).

For example:



```
AD_Initialize(
    "<?xml version='1.0' encoding='iso-8859-1' ?>\n"
    "<!DOCTYPE SetConfig SYSTEM 'SetConfig.dtd'>\n"
    "<SetConfig>\n"
    "<General /\>\n"
    "<UnlockCode>(Enter Code here)</UnlockCode>\n"
    "<DataBase CountryISO3='ALL' Type='BATCH_INTERACTIVE' Path='/ADDB'
    PreloadingType='NONE'/>\n"
    "</SetConfig>\n",
    NULL,
    NULL,
    NULL
);
```

Or in Java:



```
// Initialize the Engine using the 'Direct' API
AddressDoctor.initialize(
    "<?xml version='1.0' encoding='UTF-16LE'?>\n" +
    "<!DOCTYPE SetConfig SYSTEM 'SetConfig.dtd'>\n" +
    "<SetConfig>\n" +
    "<General WriteXMLEncoding='UTF-16LE' /\>\n" +
    "<UnlockCode>(Enter Code here)</UnlockCode>\n" +
    "<DataBase CountryISO3='ALL' Type='BATCH_INTERACTIVE' Path='/ADDB'
    PreloadingType='NONE'/>\n" +
    "</SetConfig>",
    null,
    null,
    null
);
```

Alternatively, the SetConfig XML string can be stored in an external file. In this case, the initialize call looks like this:



```
AD_Initialize(
    NULL,
    "SetConfig.xml",
    NULL,
    NULL
);
```

Or in Java:



```
// Initialize the Engine using the 'XML' API
AddressDoctor.initialize(
    null,
    "SetConfig.xml",
    null,
    null
);
```

The following return codes (see chapter 5.25 for an explanation of return codes) are typical warnings and errors returned by `AD_Initialize()`:

- AD\_SC\_WRN\_INIT\_UNLOCKCODE\_CORRUPT (1)
- AD\_SC\_WRN\_INIT\_UNLOCKCODE\_EXPIRED (2)
- AD\_SC\_WRN\_INIT\_DB\_NOT\_FOUND (3)
- AD\_SC\_WRN\_INIT\_DB\_CORRUPT (4)
- AD\_SC\_WRN\_INIT\_DB\_UNSUPPORTED\_VERSION (5)
- AD\_SC\_WRN\_INIT\_DB\_NOT\_SUPPORTED (6)
- AD\_SC\_WRN\_INIT\_DB\_NOT\_UNLOCKED (7)
- AD\_SC\_WRN\_INIT\_MULTIPLE\_DB\_ENTRIES (8)
- AD\_SC\_WRN\_MAXMEMORYUSAGE\_TOO\_SMALL (9)
- AD\_SC\_ERR\_INIT\_NO\_DB\_FOUND (900)
- AD\_SC\_ERR\_INIT\_NO\_DB\_OPENED (901)
- AD\_SC\_ERR\_EXTRA\_CASS\_DBS\_ERROR (902)

If one of these codes is returned, the engine is initialized - however, some potential problem occurred which needs to be investigated: For that purpose, retrieving `GetConfig.xml` via `AD_GetConfigSettingsXML()` is strongly advised, as its contents provide additional information about problems with unlock codes and / or database files.

`AD_DeInitialize()` must be called last to de-initialize the engine; the engine is then ready to be initialized again: All `AddressObjects` must have been released by calling `AD_ReleaseAddressObject()` or `AD_ReleaseAllAddressObjects()` before calling `AD_DeInitialize()`, see chapter 4.1 for a full example (including Java).

## 6.2 ...determine the AddressDoctor version?

`AD_GetVersion()` can be called at any time, even before calling `AD_Initialize()` / `AD_InitializeW()`, to retrieve the zero-terminated engine version string in the format `x.x.x.x`, i.e. "5.0.0.251".

## 6.3 ...specify processing or input parameters and a result format?

An `AddressObject` has a result format configuration, for possible attributes see `Parameters.dtd` in chapter 10.1 (and the most common parameters are described starting with chapter 5.12). These parameter attributes can be set for each `AddressObject` individually by calling `AD_SetParametersXML()`.

For example:



```
AD_SetParametersXML(hAOHandle,
    "<?xml version='1.0' encoding='iso-8859-1' ?>\n"
    "<!DOCTYPE Parameters SYSTEM 'Parameters.dtd'>\n"
    "<Parameters>\n"
    "<Process Mode='BATCH' />\n"
    "<AddressElementStandardize>\n"
    "<Country Casing='UPPER' />\n"
    "</AddressElementStandardize>\n"
    "</Parameters>\n",
    NULL,
    NULL
);
```

Or in Java:



```
// This code assumes you've already acquired m_oAO as the active AddressObject
m_oAO.setParametersXML(
    "<?xml version='1.0' encoding='UTF-16LE' ?>\n" +
    "<Parameters>\n" +
    "<Process Mode='BATCH' />\n" +
    // Java uses UTF-16LE as default encoding for its String method
    "<Input Encoding='UTF-16LE' />" +
    "<Result Encoding='UTF-16LE' />" +
    "<AddressElementStandardize> \n" +
    "<Country Casing='UPPER' />\n" +
    "</AddressElementStandardize> \n" +
    "</Parameters>",
    null
);
```

As shown for SetConfig.xml in chapter 6.1, alternatively a file name may be provided:

```
C AD_SetParametersXML( hAOHandle,
    NULL,
    NULL,
    "Parameters.xml"
);
```

Or in Java:

```
Java // This code assumes you've already acquired m_oAO as the active AddressObject
m_oAO.setParametersXML(
    null,
    "Parameters.xml"
);
```

Instead of setting attributes for each AddressObject individually, Parameters.xml may already be passed on `AD_Initialize()` (refer to the API documentation in chapter 10.2 for details), thus applying global defaults to all AddressObjects that do not have individual parameters set via the method described above.

For example:

```
C AD_Initialize(
    "<?xml version='1.0' encoding='iso-8859-1' ?>\n"
    "<!DOCTYPE SetConfig SYSTEM 'SetConfig.dtd'>\n"
    "<SetConfig>\n"
    "<General />\n"
    "<UnlockCode>(Enter Code here)</UnlockCode>\n"
    "<DataBase CountryISO3='ALL' Type='BATCH_INTERACTIVE' Path='/ADDB'
    PreloadingType='NONE' />\n"
    "</SetConfig>\n",
    NULL,
    "<?xml version='1.0' encoding='iso-8859-1' ?>\n"
    "<!DOCTYPE Parameters SYSTEM 'Parameters.dtd'>\n"
    "<Parameters>\n"
    "<Process Mode='BATCH' />\n"
    "<AddressElementStandardize>\n"
    "<Country Casing='UPPER' />\n"
    "</AddressElementStandardize>\n"
    "</Parameters>\n",
    NULL
);
```

Or in Java:

## Java

```
AddressDoctor.initialize(
    "<?xml version='1.0' encoding='UTF-16' ?>" +
    "<!DOCTYPE SetConfig SYSTEM 'SetConfig.dtd'" +
    "<SetConfig><General WriteXMLEncoding='UTF-16' />" +
    "    <UnlockCode>(Enter Code here)</UnlockCode>" +
    "    <DataBase CountryISO3='ALL' Type='BATCH_INTERACTIVE'" +
    "        Path='/ADDB' PreloadingType='NONE' />" +
    "</SetConfig>", null,
    "<?xml version='1.0' encoding='UTF-16' ?>" +
    "<!DOCTYPE SetConfig SYSTEM 'Parameters.dtd'" +
    "<Parameters WriteXMLEncoding='UTF-16'" +
    "    <Input Encoding='UTF-16' />" +
    "    <Result Encoding='UTF-16' />" +
    "</Parameters>", null);
```

Again, the Parameters XML string can be stored in an external file (as is the case for SetConfig, see above). Then the `AD_Initialize()` call would look like the following:

## C

```
AD_Initialize(
    NULL,
    "SetConfig.xml",
    NULL,
    "Parameters.xml"
);
```

Or in Java:

## Java

```
AddressDoctor.initialize(
    "<?xml version='1.0' encoding='UTF-16' ?>" +
    "<!DOCTYPE SetConfig SYSTEM 'SetConfig.dtd'" +
    "<SetConfig><General WriteXMLEncoding='UTF-16' />" +
    "    <UnlockCode>(Enter Code here)</UnlockCode>" +
    "    <DataBase CountryISO3='ALL' Type='BATCH_INTERACTIVE'" +
    "        Path='/ADDB' PreloadingType='NONE' />" +
    "</SetConfig>", null,
    "<?xml version='1.0' encoding='UTF-16' ?>" +
    "<!DOCTYPE SetConfig SYSTEM 'Parameters.dtd'" +
    "<Parameters WriteXMLEncoding='UTF-16'" +
    "    <Input Encoding='UTF-16' />" +
    "    <Result Encoding='UTF-16' />" +
    "</Parameters>", null);
```

Please note that adjusting parameters might have no effect for countries that lack the postal reference data information required for their making a difference, examples would be “OptimizationLevel” (chapter 5.26), “PreferredLanguage” (chapter 5.12.2) or “MatchingScope” (chapter 5.12.5). For a reference on country coverage see: [http://www.addressdoctor.com/en/countries\\_data/countries5.asp](http://www.addressdoctor.com/en/countries_data/countries5.asp)

## 6.4 ...handle unlock codes?

To validate addresses from a country an unlock code is required. Each code unlocks a number of countries for a specified time. The unlock code is passed to the initialize function (see chapter 6.1). The function will return an error if the code is no longer valid or not correct at all. For more details on error return codes see 5.25. Please note that separate unlock codes for validation and geocoding are required.

The use of multiple unlock codes is supported. The unlock codes have to be passed to the initialize function one after another, as shown in the example below.

If there is more than one unlock code for a country the one with the longest valid date is used. Outdated unlock codes are ignored as long as there is one code that is still valid, e.g.:

```
Code A unlocks DEU and USA validation until 31.12.2009
Code B unlocks CHE and USA validation until 31.12.2010
Code C unlocks CHE and USA geocoding until 31.12.2010
```

In this case DEU validation will be unlocked until 31.12.2009 while CHE and USA validation and geocoding continue to be unlocked until 31.12.2010. Please note that unlock codes also carry a start date and will be invalid before that date, information on unlock codes may be queried using `AD_GetConfigSettingsXML()`, see the chapter 6.6 for details.

The following very simple code example shows how to use multiple unlock codes:



```
AD_Initialize(
    "<?xml version='1.0' encoding='iso-8859-1' ?>\n"
    "<SetConfig>\n"
    "<General /\>\n"
    "<UnlockCode>(Enter Code A here)</UnlockCode>\n"
    "<UnlockCode>(Enter Code B here)</UnlockCode>\n"
    "<UnlockCode>(Enter Code C here)</UnlockCode>\n"
    "<DataBase CountryISO3='USA' Type='GEOCODING' Path='/ADDB' PreloadingType='NONE' /\>\n"
    "<DataBase CountryISO3='CHE' Type='GEOCODING' Path='/ADDB' PreloadingType='NONE' /\>\n"
    "<DataBase CountryISO3='ALL' Type='BATCH_INTERACTIVE' Path='/ADDB'
PreloadingType='NONE' /\>\n"
    "</SetConfig>\n",
    NULL,
    NULL,
    NULL
);
```

Or in Java:

Java

```
AddressDoctor.initialize(
    "<?xml version='1.0' encoding='UTF-16LE'?>\n" +
    "    <!DOCTYPE SetConfig SYSTEM 'SetConfig.dtd'\n" +
    "    <SetConfig\n" +
    "    <General WriteXMLEncoding='UTF-16LE'/>\n" +
    "    // Engine & DB Unlock Code\n" +
    "    <UnlockCode>(Enter Code A here)</UnlockCode>\n" +
    "    <UnlockCode>(Enter Code B here)</UnlockCode>\n" +
    "    <UnlockCode>(Enter Code C here)</UnlockCode>\n" +
    "    <DataBase CountryISO3='USA' Type='GEOCODING' Path='/ADDB' PreloadingType='NONE'/>\n" +
    "    <DataBase CountryISO3='CHE' Type='GEOCODING' Path='/ADDB' PreloadingType='NONE'/>\n" +
    "    <DataBase CountryISO3='ALL' Type='BATCH_INTERACTIVE' Path='/ADDB'\n" +
    "    PreloadingType='NONE'/>\n" +
    "    </SetConfig>",
    null, null, null);
```

## 6.5 ...configure reference databases?

While for convenience reasons the virtual ISO code "ALL" is provided for defining default settings, you may adjust paths and pre-loading settings (see chapter 5.27) for each country reference database type separately. The following lines would be an example of a non-trivial SetConfig.xml (see chapter 6.1 as well and the DTD in chapter 10.1):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE SetConfig SYSTEM "C:/AddressDoctor/DTD/SetConfig.dtd">

<SetConfig>
  <General WriteXMLEncoding="UTF-16" MaxMemoryUsageMB="2048" MaxAddressObjectCount="10"
    MaxThreadCount="2"></General>

  <UnlockCode>Address Validation Unlock Code</UnlockCode>
  <UnlockCode>Geocoding Unlock Code</UnlockCode>
  <DataBase CountryISO3="USA" Type="CERTIFIED" Path="C:/AddressDoctor/DB/CASS"
    PreloadingType="FULL"></DataBase>
  <DataBase CountryISO3="CAN" Type="CERTIFIED" Path="C:/AddressDoctor/DB/SERP"
    PreloadingType="FULL"></DataBase>
  <DataBase CountryISO3="USA" Type="SUPPLEMENTARY" Path="C:/AddressDoctor/DB/Enrichment"
    PreloadingType="PARTIAL"></DataBase>
  <DataBase CountryISO3="GBR" Type="SUPPLEMENTARY" Path="C:/AddressDoctor/DB/Enrichment"
    PreloadingType="PARTIAL"></DataBase>
  <DataBase CountryISO3="ALL" Type="GEOCODING" Path="C:/AddressDoctor/DB/Geocoding"
    PreloadingType="NONE"></DataBase>
  <DataBase CountryISO3="ALL" Type="BATCH_INTERACTIVE" Path="C:/AddressDoctor/DB"
    PreloadingType="NONE"></DataBase>
  <DataBase CountryISO3="ALL" Type="FASTCOMPLETION" Path="C:/AddressDoctor/DB"
    PreloadingType="NONE"></DataBase>
</SetConfig>
```

Please note that any country specific settings must precede the "DataBase" elements with CountryISO3="ALL" to be actually applied (the effective database settings and their unlock status may

be verified through GetConfig.xml, as described in chapter 6.6; specifically check for "DataBase" element entries with Status attributes other than "ACTIVE"). In case of conflicting "DataBase" elements, the first occurrence in SetConfig.xml will always have precedence.

There are a few notable deviations from that standard behaviour for most countries:

For CountryISO3="USA" and Type="CERTIFIED" the GetConfig.xml output will only list a subset of the available reference database files as "DataBase" element with Type="CERTIFIED". Also, USA5BI.md must always be available, as this database is the basis for CASS processing (see chapter 6.24) as well. Furthermore, US CERTIFIED mode will not work without pre-loading and full pre-loading will always be enforced on some of the CASS databases described in chapter 3.3.2, irrespectively of the settings. For CountryISO3="CAN" and Type="CERTIFIED" the GetConfig.xml output will now list CAN5C1.MD as the database, as there is now a separate database necessary for certified processing (see chapter 6.24.2).

For CountryISO3="JPN" and Type="FASTCOMPLETION" the GetConfig.xml output will only list one "DataBase" element with Type="BATCH\_INTERACTIVE" as well, due to a slightly different internal database layout.

## 6.6 ...determine the current engine settings?

On the global AddressDoctor level, calling `AD_GetConfigSettingsXML()` will return a GetConfig.xml with the engine configuration, which has been set upon calling `AD_Initialize()`, see chapter 6.1.

Accordingly, calling `AD_GetParametersSettingsXML()` will return a Parameters.xml with the engine default set of parameters, which again have been set upon calling `AD_Initialize()`. In contrast, the parameters effectively applied when processing each AddressObject (which may well be identical to these global settings unless explicitly set using `AD_SetParametersXML()`, see the preceding chapter 6.3) can be queried via `AD_GetParametersXML()`. See the API reference in chapter 10.2 for details.

## 6.7 ...assign an address to the AddressObject?

In order to achieve the best possible processing, it is important to understand the structure of your input data. One can then decide on the best way to input the data into AddressDoctor's AddressObject (see chapter 5.7).

In general address data will exist as one of the following:

- **Fielded data.** In some databases, particularly ones driven by direct input all the data may be fielded (e.g. Street, City, State, ZIP/PostCode are all stored in individual fields).

Street name: Blodgett Road  
 House number: 17413  
 PO Box: PO Box 123  
 City: Mount Vernon  
 State: Washington  
 ZIP: 97273

- **Partially fielded data.** In many databases address data has been partially broken out. For example a separate state or postal code field. But some of the address is left in generic "address lines".

Address1: 17413 Blodgett Road  
 Address2: PO Box 123  
 City: Mount Vernon  
 State: Washington  
 ZIP: 97273

- **Unfielded data.** This could be data derived from scanning address labels for example.

Address1: 17413 Blodgett Road  
 Address2: PO Box 123  
 Address3: Mount Vernon WA 97273

## 6.7.1 General Overview

If an old input address is still present, it must be cleared by a call of `AD_ClearData()`. The input address data can be set either with the direct API functions or via the `AD_SetInputDataXML()` function.

Example for fielded input (direct API):



```
AD_SetInputAddressElement( hAOHandle, "Country", 1, NULL, "Canada" );
AD_SetInputAddressElement( hAOHandle, "PostalCode", 1, NULL, "G1R 3X2" );
AD_SetInputAddressElement( hAOHandle, "Locality", 1, NULL, "Toronto" );
AD_SetInputAddressElement( hAOHandle, "DeliveryService", 1, NULL, "PO Box 1827" );
```

Or in Java:

## Java

```
m_oAO.setInputAddressElement("Country", 1, null, "Canada");
m_oAO.setInputAddressElement("PostalCode", 1, null, "G1R 3X2");
m_oAO.setInputAddressElement("Locality", 1, null, "Toronto");
m_oAO.setInputAddressElement("DeliveryService", 1, null, "PO Box 1827");
```

Example for fielded input (XML API):

## C

```
AD_SetInputDataXML( hAOHandle,
    "<?xml version='1.0' encoding='ISO-8859-1'?>\n"
    "<!DOCTYPE InputData SYSTEM 'InputData.dtd'>\n"
    "<InputData>\n"
    "<AddressElements>\n"
    "  <Country Item='1' Type='NAME'>SGP</Country>\n"
    "  <Locality Item='1' Type='COMPLETE'>Singapore</Locality>\n"
    "  <PostalCode Item='1' Type='FORMATTED'>048624</PostalCode>\n"
    "  <Street Item='1' Type='COMPLETE'>Raffles Place</Street>\n"
    "  <Number Item='1' Type='COMPLETE'>80</Number>\n"
    "  <Building Item='1' Type='COMPLETE'>#50-01 UOB Plaza 1</Building>\n"
    "  <Organization Item='1' Type='NAME'>AddressDoctor GmbH</Organization>\n"
    "</AddressElements>\n"
    "</InputData>\n"
);
```

For comparison in Java:

## Java

```
m_oAO.setInputDataXML(
    "<?xml version='1.0' encoding='UTF-16'?>"+
    "<!DOCTYPE InputData SYSTEM InputData.dtd'>"+
    "<InputData>"+
    "<AddressElements>"+
    "  <Key>4711</Key>"+
    "  <Country Item='1' Type='NAME'>SGP</Country>"+
    "  <Locality Item='1' Type='COMPLETE'>Singapore</Locality>"+
    "  <PostalCode Item='1' Type='FORMATTED'>048624</PostalCode>"+
    "  <Street Item='1' Type='COMPLETE'>Raffles Place</Street>"+
    "  <Number Item='1' Type='COMPLETE'>80</Number>"+
    "  <Building Item='1' Type='COMPLETE'>#50-01 UOB Plaza 1</Building>"+
    "  <Organization Item='1' Type='NAME'>AddressDoctor GmbH</Organization>"+
    "</AddressElements>"+
    "</InputData>");
```

## 6.7.2 Fielded address input

Fully fielded addresses will typically provide the most reliable results when cleansing an address. Even in databases that have the address components in separate columns it is not uncommon to have the house number and the street name in the same field.

The structure may look like this:

COUNTRY	FIRSTNAME	NAME	STREET	TOWN	STATE	ZIPCODE
United States	Mark	Myers	7563 Bangor Ave	Hesperia	CA	
United States	Istvan	Edgars	87 MILL LN	New York	NY	10123

An address is still considered to be fielded when house number and street name reside in the same field. In this case the field containing the house number and street name may be assigned to the Street attribute together.

To support environments where databases contain address data broken into discrete fields AddressDoctor allows direct input of each address component (including addressing information such as contact, organization etc.) via the "AddressElements" element of InputData.xml (see DTD in chapter 10.1). Possible address elements are: Key, Country, Locality, PostalCode, Province, Street, Number, Building, SubBuilding, DeliveryService, Organization and Contact.

Though the data is input to specific address elements, AddressDoctor can still perform parsing in case the data has been stored in the incorrect fields (depending on the "OptimizationLevel" chosen, see chapter 5.26). Incorrect fielding of data is particularly common for international addresses. If there is a high level of incorrect fielding it may be desirable to explore other input strategies (pre-processing to correct the fielding, concatenation and partially structured input, etc.).

Please note that InputData.xml does not only allow assigning an item attribute to each input address element but even supports flagging each of these items with corresponding type information, down to address sub-element level (see chapter 5.10 for reference).

For example, available types for the "Contact" address element or sub-elements are:

COMPLETE (element), FIRST\_NAME (sub-element), MIDDLE\_NAME (sub-element), LAST\_NAME (sub-element), NAME (sub-element), TITLE (sub-element), FUNCTION (sub-element), SALUTATION (sub-element) and GENDER (sub-element)

Or for "Organization":

COMPLETE (element), NAME (sub-element), DESCRIPTOR (sub-element) and DEPARTMENT (sub-element).

Consequently, you might either want to assign "AddressDoctor GmbH Support" as one "Organization" address element item of type "COMPLETE" or in sub-element items of: "AddressDoctor" with type "NAME", "GmbH" with type "DESCRIPTOR" and "Support" with type "DEPARTMENT". That such type attributes are provided on input for each address sub-element (i.e. item) is absolutely crucial for correct output formatting in the case of "Contact" and "Organization" addressing information, which is not covered by postal reference data.

See the DTD in chapter 10.1 for a complete and up-to-date list of all the “AddressElements” item types supported by AddressDoctor, noting the limitations described in chapter 5.10.

## 6.7.3 Partially fielded address input

Often databases contain contact information separate from address data. But the address itself is broken into “address lines”.

For example:

COUNTRY	CUSTOMER	ADDRESS_LINE_1	ADDRESS_LINE_2	CITY	STATE	ZIP
USA	John Smith	7563 Bangor Ave	Suite 107	Hesperia	CA	92345
USA	Vlad Marcos	Acme Products	3198 MARINO ST	El Paso	TX	79925

In this case the address data is input using the fielded address elements where possible (e.g. Contact, Province, Locality, Country, PostalCode), and then the “AddressLines” element of Input.xml (see DTD in chapter 10.1) is used to input the remaining data. Typically, that will involve filling the “DeliveryAddressLines” (DAL) sub-element with input data, but in case data is available in that specific format, using the “RecipientLine” and “CountrySpecificLocalityLine” (CSLL) sub-elements is also possible. As in the case of fully fielded address data, when data has been partially broken out, the best results are obtained by assigning that data to the appropriate address element.

## 6.7.4 Unfielded address input

Since unfielded data has no explicit structure (other than line feeds) this input is the most flexible. However, for the same reason, it will also produce the least reliable results.

To populate an AddressObject with unfielded data, developers will need to use the “AddressComplete” element of Input.xml. The address is simply passed to “AddressComplete” as a set of strings separated by line feeds (see the following example).

To return the best results it is important to set the most appropriate of the following “FormatType” attribute (see chapter 5.13 for details) of the Input element:

- ALL
- ADDRESS\_ONLY
- WITH\_ORGANIZATION
- WITH\_CONTACT
- WITH\_ORGANIZATION\_CONTACT
- WITH\_ORGANIZATION\_DEPARTMENT

The use of “AddressComplete” must not be combined with other address input, except for “Country”. Also, better results will be obtained if the addresses resemble at least some of the structure used in the respective country.

As an example,

John Smith  
7563 Bangor Ave  
Hesperia CA 92345  
USA

will yield significantly better results than:

John Smith  
7563  
Bangor Ave  
Hesperia  
CA  
92345  
USA

A typical database structure might look like this:

ADDRESS_1	ADDRESS_2	ADDRESS_3	ADDRESS_4	ADDRESS_5	ADDRESS_6
John Smith	7563 Bangor Ave	Suite 107	Hesperia CA	92345	USA
AddressDoctor GmbH	Stefan Vogel	Röntgenstr. 9	67133 Maxdorf		Deutschland
Vlad Marcos	c/o Acme Products	123 Main Street #12	El Paso TX 79925		United States

In case of such data, the FormattedAddressLine (FAL) sub-element of AddressLines might be a more appropriate alternative, which allows for input of up to 19 unfielded address lines

## 6.8 ...validate an address?

The AddressObject must have been filled with an input address. Processing an address is achieved by calling `AD_Process()`. The process mode must have been set to BATCH, INTERACTIVE, FAST\_COMPLETION or CERTIFIED, otherwise the default processing mode BATCH is used. Detailed results are retrieved by specific API functions, see below in chapter 6.11.

Example for C:

```
AD_Process( hAOHandle );
```

And for Java:

```
AddressDoctor.process(m_oAO);
```

## 6.9 ...parse an address?

The AddressObject must have been filled with an input address. Processing an address is achieved by calling `AD_Process()`. The process mode must have been set to PARSE. Detailed results are retrieved by specific API functions, see below in chapter 6.11.

For example:

```
C
AD_SetParametersXML( hAOHandle,
    "<?xml version='1.0' encoding='iso-8859-1' ?>\n"
    "<!DOCTYPE Parameters SYSTEM 'Parameters.dtd'\n"
    "<Parameters>\n"
    "<Process Mode='PARSE'/>\n"
    "</Parameters>\n",
    NULL
);
AD_Process( hAOHandle );
```

Or in Java:

```
Java
m_oAO.setParametersXML(
    "<?xml version='1.0' encoding='UTF-16LE' ?>\n" +
    "<Parameters>\n" +
    "<Process Mode='PARSE'/>\n" +
    "// Java uses UTF-16LE as default encoding for its String method
    "<Input Encoding='UTF-16LE'/>" +
    "<Result Encoding='UTF-16LE'/>" +
    "</Parameters>",
    null);
AddressDoctor.process(m_oAO);
```

## 6.10 ...check the process mode?

After `AD_Process()` has been called, the "ModeUsed" attribute of the "Result" element will allow checking that the process mode used was actually the one intended (see chapter 5.11 for possible process mode fallbacks):

```
C
char sResultParameters[32];
AD_GetResultParameter( hAOHandle, "ModeUsed", sResultParameters, sizeof( sResultParameter) );
```

Or in Java:

```
Java
System.out.println(m_oAO.getResultParameter("ModeUsed"));
```

## 6.11 ...retrieve a suggested correction?

`AD_Process()` must have been called upfront to process the input address, only then results are available: The return code of `AD_Process()` already gives some indication of fatal errors (country not identified etc. – see chapter 5.25).

When using the direct API, the first step before calling `AD_GetResultAddressElement()` is always retrieving the number of results first by calling

```
AD_GetResultCount()
```

, while the number of items or lines for a specific item can be retrieved by calling

```
AD_GetResultAddressElementItemCount()
```

or

```
AD_GetResultAddressLineCount()
```

, respectively.

Example (direct API, no error handling):



```
AD_U32 ulNumResults;
size_t stCurResult;

AD_GetResultCount( hAOHandle, &ulNumResults );
for( stCurResult = 1; stCurResult <= ulNumResults; stCurResult++ )
{
    char sStreet[ 256 ];
    AD_U32 ulNumItems;
    size_t stCurItem;

    AD_GetResultAddressElementItemCount( hAOHandle, 1, "Street", &ulNumItems );
    for( stCurItem = 1; stCurItem <= ulNumItems; stCurItem++ )
    {
        AD_GetResultAddressElement( hAOHandle, stCurResult, "Street", stCurItem,
            "COMPLETE", sStreet, sizeof( sStreet ) );
        printf( "Result %u: Street item %u: %s\n", stCurResult, stCurItem, sStreet );
    }
}
```

Or in Java:



```
int NumResults = m_oAO.getResultCount();
int CurResult;
for (CurResult = 1; CurResult <= NumResults; CurResult++) {
    int NumItems = m_oAO.getResultAddressElementItemCount(CurResult, "Street");
    int CurItem;
    for (CurItem = 1; CurItem <= NumItems; CurItem++) {
        System.out.println(m_oAO.getResultAddressElement(CurResult, "Street", CurItem,
"COMPLETE"));
    }
}
```

Example (C XML API, no error handling):

```
char sResultXML[ 16 * 1024 ];
AD_GetResultXML( hAOHandle, sResultXML, sizeof( sResultXML ) );
```

Example (Java XML API, no error handling):

```
String sResultXML = "";
sResultXML = m_oAO.getResultXML();
```

## 6.12...retrieve the result status and additional information?

For the direct API, `AD_GetResultParameter()` will return more detailed processing result information (see the code shown in chapter 6.10 for another example), for example the process status value explained in chapter 5.15:



```
char sResultParameters[32];
AD_GetResultParameter(hAOHandle, "ProcessStatus", sResultParameters,
    sizeof(sResultParameter));
```

Or in Java:



```
System.out.println(m_oAO.getResultParameter("ProcessStatus"));
```

To get a detailed status for any specific address element result, `AD_GetResultDataParameter()` can be called. Likewise, for Enrichments you may call `AD_GetResultEnrichmentDataParameter()`.

For a list of all parameters available to "Result", "ResultData" and "ResultEnrichmentData", see the attributes for those elements of Result.dtd (chapter 10.1): For instance, the "Result" element provides parameter attributes like "ProcessStatus" or "ModeUsed", while the "ResultData" element provides parameter attributes like "ElementMatchStatus" or "ElementResultStatus".

The XML API on the other hand, has only a single function `AD_GetResultXML()` which writes a complete result XML text to the passed buffer (see chapter 6.11 above). The level of detail contained in that XML construct may be influenced using the three attributes "AddressElements" (NONE, STANDARD, DETAILED), "AddressLines" (ON, OFF) and "AddressComplete" (ON, OFF) of the "Result" element in Parameters.xml (see DTD in chapter 10.1 and chapter 6.3).

Please note that XML output of all possible address element Types (see chapter 5.10) is only available when the "AddressElements" attribute for Result.xml is set to "DETAILED". Many of the types described in the DTD (see chapter 10.1) are only available where supported by the available reference data and thus may vary greatly from country to country and even address to address. They are primarily provided for analytical purposes for now, while for most practical applications the default result output Type "COMPLETE" is best suited, as returned for "AddressElements" set to "STANDARD".

## 6.13...retrieve address enrichments?

For the time being, AddressDoctor 5 supports the following enrichments (for all Process Modes, except FAST\_COMPLETION):

1. GeoCoding (set `EnrichmentGeoCoding="ON"`)
2. SupplementaryUS (presently providing `COUNTY_FIPS_CODE`, `STATE_FIPS_CODE`, `MSA_ID`, `CBSA_ID`, `FINANCE_NUMBER`, `RECORD_TYPE`, `CSMA_ID`, `TIME_ZONE_CODE`, `TIME_ZONE_NAME`, `CENSUS_TRACT_NO`, `CENSUS_BLOCK_NO`, `CENSUS_BLOCK_GROUP`, `PMSA_ID`, `MCD_ID` & `PLACE_FIPS_CODE`) set `EnrichmentSupplementaryUS="ON"`)
3. SupplementaryGB (presently providing `DELIVERY_POINT_SUFFIX`, set `EnrichmentSupplementaryGB="ON"`)
4. SERP (set `EnrichmentSERP="ON"`)
5. CASS (set `EnrichmentCASS="ON"`)
6. SNA (set `EnrichmentSNA="ON"`)
7. AMAS (set `EnrichmentAMAS="ON"`)

Enabled enrichments are processed as the last processing step when calling `AD_Process()`: To enable Geocoding for example, the "Process" attribute "EnrichmentGeoCoding" within `Parameters.xml` (see Appendix 10.1) must be set to ON (default for all enrichments is OFF). Respective switches are provided for all enrichments, see the `Parameters.xml` DTD in Appendix 10.1 for details.

Enrichments might be subject to providing an extra unlock code (as is the case for Geocoding, see chapter 6.4) and will usually require extra database files (see chapter 6.5 for examples).

Enrichment results can then be obtained in the direct API case by first calling the function

```
AD_GetResultEnrichmentElementExists()
```

to check for their existence and then

```
AD_GetResultEnrichmentElement()
```

for actually retrieving them.

`AD_GetResultEnrichmentDataParameter()` is provided to access the enrichment specific result information, like "GeoCodingStatus" (for a list of the available parameter attributes see the elements of `Result.dtd` in chapter 10.1). When using the XML API, calling `AD_GetResultXML()` provides all enabled enrichment results as well.

For example code see chapter 6.11, also see chapters 5.17 to 2 for GeoCoding, CASS, SERP, AMAS, SNA and the Supplementary status values and chapter 6.24 for details on the certified CASS, SERP, AMAS and SNA enrichments.

## 6.14...analyze error conditions?

For C, `AD_GetLastError()` provides you with the last error return code (see chapter 5.25 for a return code overview) and `AD_GetExtendedErrorMsg()` allows access to extended information pertaining to the

last error. Error messages often point to configuration issues that are best analyzed by referring to GetConfig.xml or Parameters.xml (see chapter 6.6 on how to obtain those).

For Java you use `AddressDoctorException.getExtendedMessage()` for that same purpose. Please make sure to wrap AddressDoctor and AddressObject calls with `try/catch` blocks for proper exception handling – for a more detailed example see the code in chapter 4.1:

## Java

```
try
{
    AddressDoctor.process(m_oAO);
    iLastError = AddressDoctor.getLastError();
    System.out.println("Process returned " + iLastError);
} catch (AddressDoctorException ex)
{
    System.out.println("Exception during process: " + ex.toString());
}
```

The ConsoleDemo test application in C and Java provided by AddressDoctor (see chapter 7.1) may prove helpful in analyzing error conditions. Please collect the information listed in chapter 9.3 before contacting AddressDoctor Support.

## 6.15...assign and process addresses in non-latin script?

In pretty much the same way as the examples shown in the preceding chapters 6.7 and 6.8. Simply make sure to input your addresses using the appropriate bit width for the source character set you are using (see chapter 5.8, UTF-16 is typically the safe choice for non-latin character sets).

Here is an example of a Japanese Kanji address:

```
<?xml version="1.0" encoding="UTF-16"?>
<InputData>
  <AddressElements>
    <Country Item="1" Type="NAME">JAPAN</Country>
  </AddressElements>
  <AddressLines>
    <FormattedAddressLine Line="1">〒 949-7277</FormattedAddressLine>
    <FormattedAddressLine Line="2">新潟県南魚沼市国際町 777 番地</FormattedAddressLine>
    <FormattedAddressLine Line="3">国際大学</FormattedAddressLine>
  </AddressLines>
</InputData>
```

The Rōmaji result, illustrating the AddressDoctor transliteration capabilities via PreferredScript set to "LATIN" in Result.xml (see DTD in chapter 10.1), would look like this:

```
<?xml version="1.0" encoding="UTF-16"?>
<Result ProcessStatus="C4"
  ModeUsed="BATCH"
  Count="1"
```

```

CountOverflow="NO"
CountryISO3="JPN"
PreferredScript="LATIN"
PreferredLanguage="DATABASE">
<ResultData ResultNumber="1"
  MailabilityScore="3"
  ResultPercentage="83.20"
  ElementResultStatus="F0F8F040400040000060"
  ElementInputStatus="60606020200020000060"
  ElementRelevance="10111000000000000010">
<AddressElements>
  <Country Type="NAME_EN" Item="1">JAPAN</Country>
  <Locality Item="1">MINAMIUONUMA-SHI</Locality>
  <Locality Item="2">ANAJISHINDEN</Locality>
  <PostalCode Item="1">949-7277</PostalCode>
  <Province Item="1">NIIGATA-KEN</Province>
  <Street Item="1">KOKUSAI-CHŌ</Street>
  <Number Item="1">777 BANCHI</Number>
  <Building Item="1">KOKUSAIDAIGAKU</Building>
</AddressElements>
<AddressLines>
  <DeliveryAddressLine Line="1">777 BANCHI KOKUSAI-CHŌ KOKUSAIDAIGAKU</DeliveryAddressLine>
  <CountrySpecificLocalityLine Line="1">MINAMIUONUMA-SHI NIIGATA-KEN 949-
    7277</CountrySpecificLocalityLine>
  <FormattedAddressLine Line="1">777 BANCHI KOKUSAI-CHŌ
    KOKUSAIDAIGAKU</FormattedAddressLine>
  <FormattedAddressLine Line="2">ANAJISHINDEN</FormattedAddressLine>
  <FormattedAddressLine Line="3">MINAMIUONUMA-SHI NIIGATA-KEN 949-
    7277</FormattedAddressLine>
  <FormattedAddressLine Line="4">JAPAN</FormattedAddressLine>
</AddressLines>
<AddressComplete>777 BANCHI KOKUSAI-CHŌ KOKUSAIDAIGAKU
ANAJISHINDEN
MINAMIUONUMA-SHI NIIGATA-KEN 949-7277
JAPAN
</AddressComplete>
</ResultData>
</Result>

```

Similarly a Russian example, in Cyrillic script:

```

<?xml version="1.0" encoding="UCS-2LE"?>
<InputData>
  <AddressElements>
    <Country Item="1" Type="NAME">RUS</Country>
  </AddressElements>
  <AddressLines>
    <FormattedAddressLine Line="1">Международный университет в Москве</FormattedAddressLine>
    <FormattedAddressLine Line="2">Ленинградский проспект 17</FormattedAddressLine>
    <FormattedAddressLine Line="3">125040 Москва</FormattedAddressLine>
  </AddressLines>
</InputData>

```

```
</AddressLines>
</InputData>
```

Results in (with PreferredScript set to “ASCII\_SIMPLIFIED” this time, to suppress special characters like the “ž” in “Meždunarodnyj”, see chapter 5.12.1 for reference):

```
<?xml version="1.0" encoding="UCS-2LE"?>
<Result ProcessStatus="C4"
  ModeUsed="BATCH"
  Count="1"
  CountOverflow="NO"
  CountryISO3="RUS"
  PreferredScript="ASCII_SIMPLIFIED"
  PreferredLanguage="DATABASE">
<ResultData ResultNumber="1"
  MailabilityScore="4"
  ResultPercentage="82.50"
  ElementResultStatus="F0F080F0F000400000E0"
  ElementInputStatus="606000060600020000060"
  ElementRelevance="10101000000000000010">
<AddressElements>
  <Country Type="NAME_EN" Item="1">RUSSIAN FEDERATION</Country>
  <Locality Item="1">Moskva</Locality>
  <PostalCode Item="1">125040</PostalCode>
  <Province Item="1">Moskva</Province>
  <Street Item="1">Leningradskij pr-kt</Street>
  <Number Item="1">17</Number>
  <Building Item="1">Mezhdunarodnyj Universitet V Moskve</Building>
</AddressElements>
<AddressLines>
  <DeliveryAddressLine Line="1">Mezhdunarodnyj Universitet V Moskve</DeliveryAddressLine>
  <DeliveryAddressLine Line="2">Leningradskij Pr-Kt 17</DeliveryAddressLine>
  <CountrySpecificLocalityLine Line="1">Moskva</CountrySpecificLocalityLine>
  <FormattedAddressLine Line="1">Mezhdunarodnyj Universitet V Moskve</FormattedAddressLine>
  <FormattedAddressLine Line="2">Leningradskij Pr-Kt 17</FormattedAddressLine>
  <FormattedAddressLine Line="3">Moskva</FormattedAddressLine>
  <FormattedAddressLine Line="4">125040</FormattedAddressLine>
  <FormattedAddressLine Line="5">Russian Federation</FormattedAddressLine>
</AddressLines>
<AddressComplete>Mezhdunarodnyj Universitet V Moskve
Leningradskij Pr-Kt 17
Moskva
125040
Russian Federation
</AddressComplete>
</ResultData>
</Result>
```

## 6.16 ...use AddressDoctor with multiple processor cores?

Let us assume a four processor core machine on which three cores are to be used for address processing: The main thread of the program integrating the AddressDoctor 5 software library calls `AD_Initialize()`; (see chapter 6.1) with `MaxThreadCount=3` and `MaxAddressObjectCount=3` (see chapter 5.29) and creates three worker threads for processing addresses.

Each worker thread then acquires one `AddressObject` handle via `AD_GetAddressObject( &hAOHandle )`; and subsequently keeps repeating the following sequence (see chapters 6.7.1, 6.8 and 6.11):



```
AD_SetInputDataXML( hAOHandle, <XML string> );
AD_Process( hAOHandle );
AD_GetResultXML( hAOHandle, sResultXML, sizeof( sResultXML ) );
AD_ClearData( hAOHandle );
```

When you are finally shutting down, the main thread destroys all worker threads and de-initializes AddressDoctor (see chapter 6.1):

```
AD_ReleaseAllAddressObjects();
AD_DeInitialize();
```

## 6.17 ...produce valid AddressDoctor XML?

Any XML input to the AddressDoctor Engine should always be well-formed and validated against the DTDs provided for that purpose by AddressDoctor (see chapter 10.1). Please do note that the sequence of the XML elements does matter (but not that of their attributes), which can be checked through DTD validation as well.

Refer to <http://wikipedia.org/wiki/XML> for an introduction to XML. Apart from XML functionality being an integral part of most modern Integrated Development Environments (IDEs), there is a diverse choice of free validating XML editors, like WMHelp XMLPad or XML Copy Editor from SourceForge.net.

When dealing with XML files produced on different platforms, please note that end-of-line (EOL) characters differ between Windows (CR+LF) and UNIX (LF), see <http://wikipedia.org/wiki/Linebreak>.

## 6.18 ...use AddressDoctor XML for flexible Business Processes?

Standards like BPEL (the Business Process Execution Language) allow for more flexible business processes implemented using information technology. For instance, you might model and implement a business process including global address verification based on an `InputData.xml` template that contains placeholder variables mapping to certain input data columns provided by data sources.

Some of the external influences (like new postal regulations) the business side might have to react on, may thus be implemented without programming knowledge, simply by adjusting these placeholders in the XML template.

For example, let us assume you are dealing with addresses for a country that has recently introduced a postal code system.

So far, your InputData.xml template might have looked like this (the “\$” character is used to delimit placeholder names here):

```
<?xml version='1.0' encoding='UTF-16'?>
<!DOCTYPE InputData SYSTEM 'InputData.dtd'>
<InputData>
  <AddressElements>
    <Key>$COLUMN1$</Key>
    <Country Item='1' Type='NAME'>$COLUMN7$</Country>
    <Locality Item='1' Type='COMPLETE'>$COLUMN6$</Locality>
    <Street Item='1' Type='COMPLETE'>$COLUMN5$</Street>
    <Building Item='1' Type='COMPLETE'>$COLUMN4$</Building>
    <Organization Item='1' Type='NAME'>$COLUMN2$</Organization>
    <Contact Item='1' Type='NAME'>$COLUMN3$</Contact>
  </AddressElements>
</InputData>
```

Due to that new postal regulation, postal codes have been added to the data source, which now need to be verified as well. A new eighth column has thus been made available and can be mapped as easily as follows:

```
<?xml version='1.0' encoding='UTF-16'?>
<!DOCTYPE InputData SYSTEM 'InputData.dtd'>
<InputData>
  <AddressElements>
    <Key>$COLUMN1$</Key>
    <Country Item='1' Type='NAME'>$COLUMN7$</Country>
    <Locality Item='1' Type='COMPLETE'>$COLUMN6$</Locality>
    <PostalCode Item='1' Type='UNFORMATTED'>$COLUMN8$</PostalCode>
    <Street Item='1' Type='COMPLETE'>$COLUMN5$</Street>
    <Building Item='1' Type='COMPLETE'>$COLUMN4$</Building>
    <Organization Item='1' Type='NAME'>$COLUMN2$</Organization>
    <Contact Item='1' Type='NAME'>$COLUMN3$</Contact>
  </AddressElements>
</InputData>
```

All that is needed for facilitating this kind of change is a simple editor as described in the preceding chapter 6.17.

## 6.19 ...use AddressDoctor for Master Data Management?

AddressDoctor provides a batch validation mode that was designed for mass data address quality, e.g. for use in Master Data Management (MDM) or Data Integration systems. This validation mode (see chapter 5.11.1 for details) allows address input into the AddressObject irrespective of data quality. The input is then automatically corrected to the extent possible, returning the single most likely candidate as the processing result.

When designing an application for batch processing, call the `AD_Process()` function with the `BATCH` validation process mode (see chapter 5.11.1). The AddressDoctor engine will return a single corrected result whenever possible (Process Status “Vx” or “Cx”, see chapter 5.15).

For tackling severe address quality challenges, a recommended batch usage pattern for AddressDoctor 5 based on the “OptimizationLevel” concept is described in chapter 5.26.

## 6.20 ...use AddressDoctor in an eBusiness Environment?

AddressDoctor provides an interactive validation mode that was designed for point of data entry address quality, e.g. for use in online registration forms, be it for a web shop, an auction platform or a customer feedback system. This validation mode (see chapter 5.11.2 for details) allows for address input into the AddressObject irrespective of data quality. The input is then automatically corrected to the extent possible, returning a choice of likely candidates. If processing can identify one definite candidate, the result returned will only be that candidate

When designing an application for interactive entry, call the `AD_Process()` function with the `INTERACTIVE` validation process mode (see chapter 5.11.2), e.g. using the web form content that has been posted to a web server online.

The AddressDoctor engine will return a number of possible results (candidates), which are then to be presented to the user entering data for picking the most correct result. If the input data entered was already complete and correct, there would obviously be no need for such user interaction.

Please note that the user should usually have the option to edit the returned result once more before final submission: For instance in the case of new construction activity, an address might not yet be featured even in the most recent set of postal reference data.

## 6.21 ...use the Quick Address Entry Feature?

AddressDoctor features a validation mode (Fast Completion) that can be used in call center environments where data entry personnel should be assisted in their data entry task. The same use case will usually apply to Customer Relationship Systems (CRM), Property & Reservation Management Systems (PMS) or Point of Sales (POS) systems. This validation mode (see chapter 5.11.3 for details) allows for incomplete address input into the AddressObject. This input is automatically completed to the extent possible.

When designing an application for quick address entry it is possible to call the `AD_Process()` function with the `FASTCOMPLETION` validation process mode (see chapter 5.11.3) after each keystroke. Provided the reference databases are either accessible quickly or even stored locally, pick lists can be displayed in real time.

As an example we are going to input the following data:

Country: USA  
Locality: Wash  
Street: Pennsylv

The AddressDoctor engine will return 20 results (suggestions) and an overflow indication will be set: If the "CountOverflow" attribute of Result.xml (see DTD in chapter 10.1) is set to YES, this indicates that potentially more results would be available. It is then recommended that the `AD_Process()` function is called again with additional input data.

## 6.22 ...use AddressDoctor in a multi-tenant hosted environment?

A multi-tenant hosted solution requires initialization of separate AddressDoctor instances with a customer specific unlock code for each, in order to meet the terms and conditions set by the different reference data providers.

AddressDoctor has examined making use of a RAM disk to share the reference database files across these several AddressDoctor instances (typically threads):

- Internal benchmarks using ramfs on Linux have shown that the address validation throughput of a RAM disk (with `PreloadingType="NONE"`, see chapter 5.27) is only about 12% less compared to full preloading (`PreloadingType="FULL"`), in the case of 4 threads running on a 4 core machine.
- Blocking of the different threads thus seems reduced by the low latency of RAM compared to hard disk storage.
- This is a good compromise between speed and hardware requirements, as about 8-10G of RAM should suffice to hold the world Batch/Interactive reference databases in a shared ramdisk.

Please note that with AddressDoctor 5.1.4 a new default method of preloading has been introduced (`PreloadingMethod="MAP"`, see chapter 5.27) which allows sharing memory mapped reference database files across instances out of the box, without the performance hit due to the ramdisk driver.

Where several GB RAM are not available for memory mapped files, each customer will then at least require separate storage with their own copy of the reference database files for performance reasons:

- These are then only partially preloaded (`PreloadingType="PARTIAL"`) to their own AddressDoctor instance (0,5 to 1 GB RAM per customer should usually suffice here), which features a caching facility for this use case as well.
- Keeping separate copies of the reference database files should ensure that customer instance I/O accesses to these files don't block each other - please do note that full preloading is a pre-requisite for proper multicore scalability (see chapter 5.29), so such a partially preloaded setup will probably limit the usable processor cores per customer thread to no more than two (because of I/O blocking again, this time between the multiple threads used for one customer).
- We have found SATA Solid State Disks to improve performance vastly in such a setup, for reference see chapter 6.25.

## 6.23 ...use AddressDoctor for Web Services?

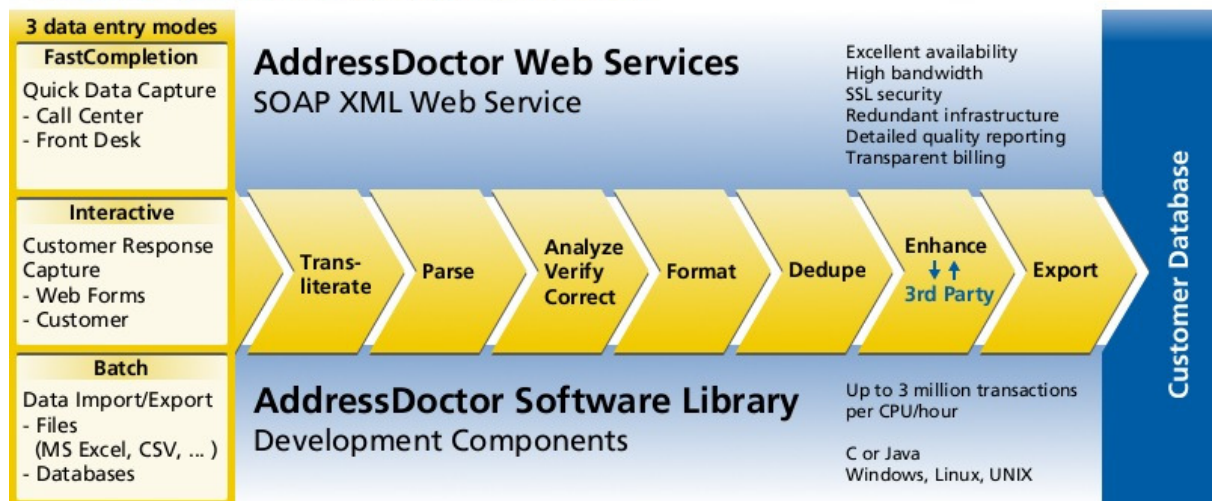
As demonstrated in chapter 4.1, AddressDoctor 5 introduced an XML API (see Appendix 10.2 for reference) that makes it even easier to integrate global address correction in Web Services environments - be it Software as a Service (SaaS) SOAP calls for Internet cloud computing or an Enterprise Service Bus (ESB) as part of a Service Oriented Architecture (SOA) in the Intranet.

Simply feed address data from your web service in XML format directly into AddressDoctor via `AD_SetInputDataXML()` (see chapter 6.7.1 for details), which only requires prior XML transformation (using broadly adopted technologies like XSLT, see <http://wikipedia.org/wiki/XSLT>) on the basis of the DTD information made available by AddressDoctor (see chapters 6.17 & 10.1).

Please note that AddressDoctor also offers secure and synchronous Web Services for direct and ready-to-use integration via SOAP (see the product line overview in chapter 2). The following figure provides an overview of the whole AddressDoctor Data Quality Platform:

### 240 Countries - 1 Solution

#### The AddressDoctor Data Quality Platform



- Batch (max. 10 addresses per SOAP call)
  - o Validation and automatic correction of addresses in batch with immediate results.
- Interactive
  - o Online validation of addresses with interactive correction.
  - o Ideal for online shops and CRM systems.
- FastCompletion
  - o Support for call centers.

The AddressDoctor Web Services have a proven track record in both, high availability (> 99.9 %) and high volume throughput. Web Service pricing is very competitive and transaction based. Also, address enrichment options are available, for details see <http://www.addressdoctor.com/en/products/ecommerce>.

## 6.24...validate an address in CERTIFIED mode?

For some countries, the AddressDoctor engine offers a special validation process mode "CERTIFIED" which is used to validate an address according to the certification rules defined by the local postal authority. This validation type allows integrators to develop their own application for certification by the respective postal organization. Special database files may be necessary for CERTIFIED processing - for details see the following chapters.

It is very important to note that the following Parameters must not be changed from their default settings to ensure proper CERTIFIED processing:

- PreferredLanguage (see chapter 5.12.2)
- MatchingAlternatives and MatchingScope (see chapter 5.12.5)
- GlobalMaxLength, GlobalCasing and AddressElementStandardize (MaxLength or Casing, see chapter 5.14)
- OptimizationLevel (see chapter 5.26)

### 6.24.1 ...process an address following the rules for CASS certification?

For US addresses, the AddressDoctor engine offers a special validation process mode "CERTIFIED" which is used to validate an address according to the USPS CASS rules. This validation type allows integrators to develop their own CASS Application for certification by USPS. Special database files are necessary for CASS processing - for details see chapter 3.3.2.

For CountryISO3="USA" and Type="CERTIFIED" the GetConfig.xml output will only list a subset of the available reference database files as "DataBase" element with Type="CERTIFIED". Also, USA5BI.md must always be available, as this database is the basis for CASS processing (see chapter 6.24) as well. Furthermore, US CERTIFIED mode will not work without pre-loading and full pre-loading will always be enforced on some of the CASS databases described in chapter 3.3.2, irrespectively of the settings.

During the validation process the input address is corrected according to CASS rules. In this process all CASS attributes are generated and the ZIP + 4 is added to the ZIP code. The output address is retrieved from the AddressObject as usual (see chapter 6.11). A CASS processing status value (see chapter 5.18) can be retrieved from the AddressObject through the "CASSStatus" attribute returned with the "EnrichmentData" element of the Result.xml.

To actually have CASS attributes available in the CASS element of Result.xml, the Process attribute "EnrichmentCASS" within Parameters.xml (see Appendix 10.1) must be set to ON (default is OFF). With "EnrichmentCASS" set to OFF, the AddressDoctor engine will still provide ZIP+4 codes (as PostalCode item type "ADD\_ON"), as long as USA5BI.MD is available in the database folder. For convenience reasons ZIP+4 codes are also provided by the US BATCH process mode, although some result variations may very well occur – the definite ZIP+4 reference is available in US CERTIFIED process mode only!

You may check for correct initialization of all required CASS databases (see chapter 3.3.2 for the full list) by querying GetConfig.xml (see the respective DTD in Appendix 10.1) for the EnrichmentSupportInfo element:

```
<EnrichmentSupportInfo CountryISO3="USA" Type="CERTIFIED">FULL</EnrichmentSupportInfo>
```

Please note that the CASS attribute output provided by AddressDoctor is only valid for use during a special validation period, varying depending on the product.

The valid time ranges are as follows (as defined on the USPS PS 3553 form, which will need to be created by the calling application to qualify for USPS mailing discounts, for an example please see [http://ribbs.usps.gov/cassmass/documents/tech%5Fguides/PS FORM 3553](http://ribbs.usps.gov/cassmass/documents/tech%5Fguides/PS_FORM_3553)):

	From Date	To Date
ZIP + 4/DPV Coded	30 days before (the 15th of each month or bi-monthly) or no later than 105 days after the file date.	180 days after the ZIP + 4 valid "From" date.
Total Delivery Point Barcoded	30 days before (the 15th of each month or bimonthly) or no later than 105 days after the ZIP + 4 product file date.	180 days after the DPBC valid "From" date.
Total Carrier Route Coded	30 days before or up to 105 days after the ZIP + 4, Five-Digit ZIP, or the Carrier Route product date (the 15th of each month or bimonthly) or up to 105 days after the file date.	90 days after the Carrier Route Valid "From" date.
Five-Digit Coded	30 days before (the 15th of each month or bimonthly) or no later than 105 days after the ZIP + 4, Five-digit ZIP, or the Carrier Route product date.	365 days after the Five-Digit Valid "From" date.

Please note that any application based on AddressDoctor 5 must meet the CASS Terms & Conditions to qualify for USPS mailing discounts:

[http://ribbs.usps.gov/cassmass/documents/tech\\_guides/FORMS/CASSDEVS.pdf](http://ribbs.usps.gov/cassmass/documents/tech_guides/FORMS/CASSDEVS.pdf)

The following list shows which CASS attributes are available (see Result.dtd in Appendix 10.1 as well) – for an explanation of the different attributes please refer to the CASS documentation at

[http://ribbs.usps.gov/cassmass/documents/tech\\_guides/TECHNICAL\\_GUIDES/CASSTECH\\_N.PDF](http://ribbs.usps.gov/cassmass/documents/tech_guides/TECHNICAL_GUIDES/CASSTECH_N.PDF):

Carrier Route Answer	CARRIER_ROUTE
Record Type Code	RECORDTYPE
Delivery Point Answer	DELIVERY_POINT
Delivery Point Check Digit Answer	DELIVERY_POINT_CHECK_DIGIT
High-rise Default	HIGHRISE_DEFAULT
High-rise Exact	HIGHRISE_EXACT
Rural route Default	RURALROUTE_DEFAULT
Rural route Exact	RURALROUTE_EXACT
DSF <sup>2</sup> LACS Indicator	LACS
DPV Confirmation Indicator*	DPV_CONFIRMATION
DPV CRMA Indicator*	DPV_CMRA
DPV False Positive Indicator*	DPV_FALSE_POSITIVE
DPV Footnote 1*	DPV_FOOTNOTE_1
DPV Footnote 2*	DPV_FOOTNOTE_2
DPV Footnote 3*	DPV_FOOTNOTE_3
Concatenation of DPV Footnotes*	DPV_FOOTNOTE_COMPLETE
Result of the call to the DPV NOSTATS Table	DSF2_NOSTATS_INDICATOR
Result of the call to the DPV VACANT Table	DSF2_VACANT_INDICATOR
LACSLink Return Code*	LACSLINK_RETURNCODE
SUITELink Return Code*	SUITELINK_RETURNCODE
ZIPMove Return Code*	ZIPMOVE_RETURNCODE
Early Warning System (EWS) Return Code	EWS_RETURNCODE
Congressional District	CONGRESSIONAL_DISTRICT
CASS Error Code	ERRORCODE
Barcode	BARCODE
Residential Delivery Indicator **	RDI_INDICATOR

Attributes marked with \* will only be populated for US customers as per USPS licensing restrictions.

Please see chapter 7 in [http://ribbs.usps.gov/dpv/documents/tech\\_guides/DPV\\_LPR.PDF](http://ribbs.usps.gov/dpv/documents/tech_guides/DPV_LPR.PDF) for details on how to programmatically act on DPV\_FALSE\_POSITIVE being set.

Attributes marked \*\* requires customers to acquire the data from USPS to enable the optional part of the processing. See Chapter 3.3.2 on how to acquire the data and rename the necessary files.

### 6.24.2 ...process an address following the rules for SERP certification?

For Canada addresses, the AddressDoctor engine offers a special validation process mode "CERTIFIED" which is used to validate an address according to the Canada Post SERP rules. This validation type allows integrators to develop their own SERP Application for certification by Canada Post. As the new databases now contain PoCAD (Point of Call Address Data) data, an additional CAN5C1.MD is needed for certified mode. Those who want to use the new engine with older databases will have to make a copy of CAN5BI.MD and rename the copy to CAN5C1.MD. See chapter 6.5 as well.

You may check for correct initialization of all required databases by querying GetConfig.xml (see the respective DTD in Appendix 10.1) for the EnrichmentSupportInfo element:

```
<EnrichmentSupportInfo CountryISO3="CAN" Type="CERTIFIED">FULL</EnrichmentSupportInfo>
```

A SERP processing status value (see chapter 5.19) can be retrieved from the AddressObject through the "SERPStatus" attribute returned with the "EnrichmentData" element of the Result.xml. To actually have SERP attributes available in the SERP sub-element of EnrichmentData in Result.xml, the Process attribute "EnrichmentSERP" within Parameters.xml (see Appendix 10.1) must be set to ON (default is OFF).

Please note that SERP certification requirements are only met, when the "PreferredScript" attribute is set to "ASCII\_SIMPLIFIED" (see chapter 5.12.1).

If the Validation type is CERTIFIED and the SERP Enrichment Status is ON, two enrichments are provided: CATEGORY and EXCLUDED\_FLAG

The category provides the following possible values:

V: Verified. The process status is of type Vx.

C: Corrected. The process status is of type Cx.

N: Incorrect. The process status is of type Ix.

VQ: Valid, but questionable. Rural addresses (those with a '0' as second digit in the PostalCode, e.g. "K0A 1L0") are usually considered valid because they are determined by the PostalCode.

Questionable means that either delivery information is missing in the input or that some part or all of the delivery input has not been verified by the database. See also the address accuracy handbook provided by Canada Post.

V1A: Valid, residential type record. Some records in the database containing buildings are marked as apartment type records, either residential or commercial. This information is provided in the enrichment.

V2A: Valid, commercial type record. This refers to commercial building records in the database.

C1A: Corrected, residential type record.

C2A: Corrected, commercial type record.

Since December 2010, PoCAD data has been added to the databases to provide more detailed suite information. (Please note that the corresponding database CAN5C1.MD will be made available early January 2011.)

The EXCLUDED\_FLAG informs about PoCAD addresses with wrong user input. The AddressDoctor output for this flag can either be empty or the Text 'EXCLUDED': EXCLUDED: Incorrect suite input for a PoCAD address, category N, process status lx

Effective January 17, 2011, the statement of accuracy has to report addresses as Excluded. However, starting August 1, 2011 this flag will no longer be needed. Addresses will no longer show up as being excluded.

See Result.dtd in Appendix 10.1 as well. Please see:

<http://www.canadapost.ca/cpo/mc/business/productservices/atoz/addressaccuracy.jsf>

### 6.24.3 ...process an address following the rules for AMAS certification?

For Australian addresses, the AddressDoctor engine offers a special validation process mode "CERTIFIED" which is used to validate an address according to the Australia Post AMAS rules. This validation type allows integrators to develop their own AMAS Application for certification by Australia Post. Special databases are necessary for AMAS processing - for details see chapter 3.3.24.

These new databases contain Postal Address File (PAF) data including Australia Post's Delivery Point Identifiers (DPIDs).

The additional AMAS information can be found in the section EnrichmentData of the Result.xml, You may check for correct initialization of all required AMAS databases (see chapter 3.3.24 for the full list) by querying GetConfig.xml (see the respective DTD in Appendix 10.1) for the EnrichmentSupportInfo element:

```
<EnrichmentSupportInfo CountryISO3="AUS" Type="CERTIFIED">FULL</EnrichmentSupportInfo>
```

The following status codes or parameters are available (see AMAS documentation for more details):

ERRORCODE	Internal error code
RECORD TYPE	Type of address (e.g. S for Street)
DELIVERY_POINT_ID	Delivery point identifier DPID, 8 numeric characters
LOT_NBR	Lot number (e.g. 100)
POSTAL_DELIVERY_NBR	Postal delivery number (e.g. "00123" of "123A")
POSTAL_DELIVERY_NBR_PFX	Postal delivery number prefix (e.g. "A" of "A123")
POSTAL_DELIVERY_NBR_SFX	Postal delivery number (e.g. "A" of "123A")
HOUSE_NBR_1	House (street) number 1 (e.g. "00123" of "123A-456B)
HOUSE_NBR_SFX_1	House (street) number 1 suffix (e.g. "A" of "123A-456B)
HOUSE_NBR_2	House (street) number 2 (e.g. "00456" of "123A-456B)
HOUSE_NBR_SFX_2	House (street) number 2 suffix (e.g. "B" of "123A-456B)

Other AMAS relevant fields are regular address elements and can be found in the standard ResultData section.

#### 6.24.4 ...process an address following the rules for SNA certification?

For French addresses, the AddressDoctor engine offers a special validation process mode "CERTIFIED" which is used to validate an address according to the La Poste SNA rules. This validation type allows integrators to develop their own SNA Application for certification by La Poste. No special database files apart from FRA5BI.md are necessary for CERTIFIED processing, see chapter 6.5 as well.

You may check for correct initialization of all required databases by querying GetConfig.xml (see the respective DTD in Appendix 10.1) for the EnrichmentSupportInfo element:

```
<EnrichmentSupportInfo CountryISO3="FRA" Type="CERTIFIED">FULL</EnrichmentSupportInfo>
```

For SNA certified processing (see: <http://www.laposte.fr/sna>) it is required to enter addresses in a six line FormattedAddressLine format, including empty lines wherever a part of the address is missing:

- Line 1: ORGANISATION IDENTIFICATION or IDENTITY OF THE ADDRESSEE
- Line 2: INDIVIDUAL IDENTIFICATION (i.e. Company Contact) or DELIVERY POINT ACCESS INFORMATION (i.e. Subbuilding)
- Line 3: DELIVERY POINT LOCATION (i.e. Building)
- Line 4: STREET NUMBER or PLOT and THOROUGHFARE
- Line 5: DELIVERY SERVICE or THOROUGHFARE COMPLEMENTARY IDENTIFICATION
- Line 6: POSTCODE and LOCALITY or CEDEX POSTCODE and DISTRIBUTION AREA INDICATOR

A SNA processing status value (see chapter 5.20) can be retrieved from the AddressObject through the "SNAStatus" attribute returned with the "EnrichmentData" element of the Result.xml. To actually have SNA attributes available in the SNA sub-element of EnrichmentData in Result.xml, the Process attribute "EnrichmentSNA" within Parameters.xml (see Appendix 10.1) must be set to ON (default is OFF).

Please note that SNA certification requirements are only met, when the "PreferredScript" attribute is set to "ASCII\_SIMPLIFIED" (see chapter 5.12.1) and "GlobalMaxLength" to 38 (with the "MaxLength" for each AddressElement set to 0, see chapter 5.14).

The only SNA attribute available is "CATEGORY" with possible values of "ORI/RES/AVE/NOK", as per La Poste definition (see Result.dtd in Appendix 10.1 as well). Please note that the SNA certification of the CERTIFIED mode for FRA is still pending.

#### 6.25 ...optimize performance?

The speed of your application will depend on the functionality of the AddressDoctor engine that is used. Parsing and Country Recognition, as implemented by AddressDoctor (AD\_Process() for C) with the Process Modes PARSE or COUNTRYRECOGNITION, do not access any databases whereas all other modes (BATCH, INTERACTIVE, FAST\_COMPLETION etc.) do.

When validating an address, a number of read operations access the corresponding country database. These accesses are random in nature. To reduce the number of read operations accessing the hard disk, preloading part of or all of a database is very much recommended. Please see chapter 5.27 for details on this topic.

However, if the size of the free physical memory (or the part made available to the process running AddressDoctor) is too small to fully preload every country database needed, AddressDoctor must access the hard disk. To reduce the number of file system calls needed, the AddressDoctor engine manages its own cache (see section 5.28). The operating system will also cache file accesses, thereby significantly speeding up subsequent calls. Naturally, the OS must have sufficient free memory available for this purpose.

Preloading on the other hand, reduces the amount of memory the operating system can use efficiently for file caching and it may even temporarily swap out the preloaded data blocks to the hard disk. For this reason it is recommended to limit the memory amount used for preloading, if it can be foreseen that additional hard disk accesses are necessary (as would for instance be the case, when the total memory available is not sufficient to allow full pre-loading of all country reference databases needed).

As minimizing accesses to the hard disk is a key to validation performance, installing more memory (see chapter 5.30 on memory allocation) will speed up processing significantly, as hard disk accesses can be avoided this way. See chapter 6.22 for reference if you are running a multi-tenant installation of AddressDoctor 5.

Please note that some operating systems can use more memory for file caching than the supported memory size per process: For example, 32 Bit Windows 2003 Server Enterprise Edition can address up to 32 GB of RAM, but the limit per process is still 2 GB (or rather 3 GB in case of using the /3GB boot.ini switch, for reference see <http://support.microsoft.com/kb/291988>) and the limit for standard CPU memory access (unless using AWE/PAE, see <http://support.microsoft.com/kb/283037> for reference) is 4 GB.

Monitoring whether you have enough memory installed is possible using a tool to monitor resource utilization (such as the Performance Monitor perfmon.exe on Windows). If sufficient memory is installed, there should be almost a 100% utilization of one processor core when processing a large batch of records in single-thread mode.

While AddressDoctor provides the means to utilize multi-threading internally (see chapter 5.29), having more than one core or processor in the system speeds up processing in itself already, because other threads in the system can run independently.

Here is the summary of tips to optimize the performance of validation

- Install as much memory as possible to allow country databases to be fully pre-loaded into memory. At least as much memory as the size of the most often used country databases combined plus

256MB should be available. If all countries available from AddressDoctor are to be used simultaneously, add more memory to cover the entire size of all databases.

- Preload at least the databases of frequently used countries with the proper parameters set in the SetConfig.xml passed to the `AD_Initialize()` function.
- When full preloading is not an option, store the database files on a fast hard disk or even better a SATA Solid State Disk (ideally exceeding 200MB/sec read transfer rate - for development purposes, high-speed USB or FireWire flash modules exceeding 30MB/sec read transfer rate might suffice). Especially the access latency (average seek time) should be minimized: Internal AddressDoctor benchmarks for "PreloadingType=NONE" with an Intel X25M G2 SATA SSD have shown a typical performance increase of a factor 20.
- Keep the AddressDoctor reference databases on a separate hard drive. Read and write address data from other drives. Make absolutely sure to keep the database files defragmented, internal tests have shown that performance may easily decrease by as much as 35% when the files are heavily fragmented.
- The AddressDoctor engine is very data-intensive, with a significant amount of non-localized memory accesses during processing: As such, it greatly benefits from direct multi-channel memory access (e.g. via Quick Path Interconnect or HyperTransport) with high bandwidth and low latency, combined with large processor caches, such as found in top-of-the line server processors.
- Use high performance multi-core processors, like Intel Xeon X55xx/65xx/75xx and higher, AMD Opteron 24xx/84xx and higher or IBM POWER7 and higher. Provided there is enough memory available for full preloading, the processor clock frequency will directly determine the speed of address processing. See <http://www.spec.org/cpu2006/results/rint2006.html> for a comparison of integer processing throughput between different processor architectures.
- When running batch processes without having a sufficient amount of memory installed, try to process records ordered by country with intermittent re-initialization of AddressDoctor using the appropriate pre-loading settings (see chapter 5.27). The engine will also benefit from internal and OS caches for addresses sorted by country as compared to addresses in random order, as they would for instance occur in a Web Service environment.

Examples for typical performance-oriented settings:

Given Ressources	System 1	System 2	System 3	System 4	System 5
Cores for AddressDoctor	1	2	4	6	12
RAM for AddressDoctor	512	1024	2048	6000	16000
<b>Chosen Settings</b>					
MaxMemoryUsageMB	450	950	1950	5950	15950
CacheSize	SMALL	LARGE	LARGE	LARGE	LARGE
MaxThreadCount	1	2	4	6	12
MaxAddressObjectCount*	1	2	4	6	12
PreloadingMethod	MAP	MAP	MAP	MAP	MAP
PreloadingType for very important countries	PARTIAL	FULL	FULL	FULL	FULL
PreloadingType for important countries	NONE	PARTIAL	PARTIAL	FULL	FULL
PreloadingType for remaining countries	NONE	NONE	NONE	PARTIAL	FULL

\* This setting depends on the implementation of the calling code. In some scenarios with double buffering two times the given value may be used

In reality the PreloadingType depends on the size of the databases, so for system 2 a FULL preload for a couple of countries may not be possible in case of large databases.

The above examples System 1 and 2 are typical for 32 bit Java usage scenarios.

When benchmarking AddressDoctor, please consider the following:

The operating system will have to read data from the hard disk if any of the databases used are not fully preloaded. These file system accesses are cached, at least until the OS file cache is full. This leads to the effect that physical hard disk accesses are always necessary for the first addresses of a specific country.

Later on some or even all of these accesses will hit the file cache. For this reason, the processing speed of the first addresses (first meaning the first few thousand) of a specific country is usually much lower than for the later ones. Thus, it is recommended to use at least 50.000 addresses per country to produce realistic benchmark results. If, on the other hand, all accessed databases are fully preloaded, speed is not expected to vary with the number of addresses already processed so far.

## 7 Demo Applications

The AddressDoctor software library package is accompanied by demo applications that can be used to quickly test the functionality of the library. See the ZIP archive structure described in chapter 3.2 for reference.

### 7.1 AddressDoctor 5 Console Demo

The ConsoleDemo application provided as source code under *src* and also contained as an executable in the *bin* directory, gives an overview of the basic address validation process.

Before running the application, copy the example XML files from *etc* over to your working directory, so that they may be found by the executable and edited for experimentation purposes. Specifically, a sample XML file *InputData.xml* containing an address for XML processing via `ConsoleDemo -xml` OR `ConsoleDemoJava -xml`, respectively is provided in *etc*. Ensure that the minimal SetConfig configuration XML provided contains a valid Unlock Code that you received when purchasing the AddressDoctor library and the correct destination path your reference database files have been unpacked to (see chapters 6.4 and 6.5 for details):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- <!DOCTYPE SetConfig SYSTEM "SetConfig.dtd" --> -->
- <SetConfig>
  <General WriteXMLEncoding="ISO-8859-1" MaxMemoryUsageMB="512" MaxAddressObjectCount="10"
    MaxThreadCount="1" />
  <UnlockCode>Unlockcode for Validation</UnlockCode>
  <UnlockCode>Unlockcode for Geocoding</UnlockCode>
  <DataBase CountryISO3="ALL" Type="BATCH_INTERACTIVE" Path=".." PreloadingType="NONE" />
  <DataBase CountryISO3="ALL" Type="FASTCOMPLETION" Path=".." PreloadingType="NONE" />
  <DataBase CountryISO3="ALL" Type="GEOCODING" Path=".." PreloadingType="NONE" />
</SetConfig>
```

Alternatively, make sure to copy (or link) at least the Swiss reference database (CHE5BI.MD, see chapter 3.3) to the working directory before running the ConsoleDemo executable: The ConsoleDemo application will attempt to validate a sample address from Switzerland that requires this database (otherwise, that example address will only be parsed!).

Remember that the contents of the *lib* directory may have to be added to your shared library path (`set PATH=%PATH%;.\lib` on Windows or `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./lib` on Unix) for an executable using the C-API to work...

For the Java-API simply call (for UNIX, see chapter 3.2.2 as well):

```
java -Xss2048k -cp bin:lib/AddressDoctor5.jar -Djava.library.path=lib ConsoleDemoJava
```

And for Windows:

```
java -Xss2048k -cp bin;lib/AddressDoctor5.jar -Djava.library.path=lib ConsoleDemoJava
```

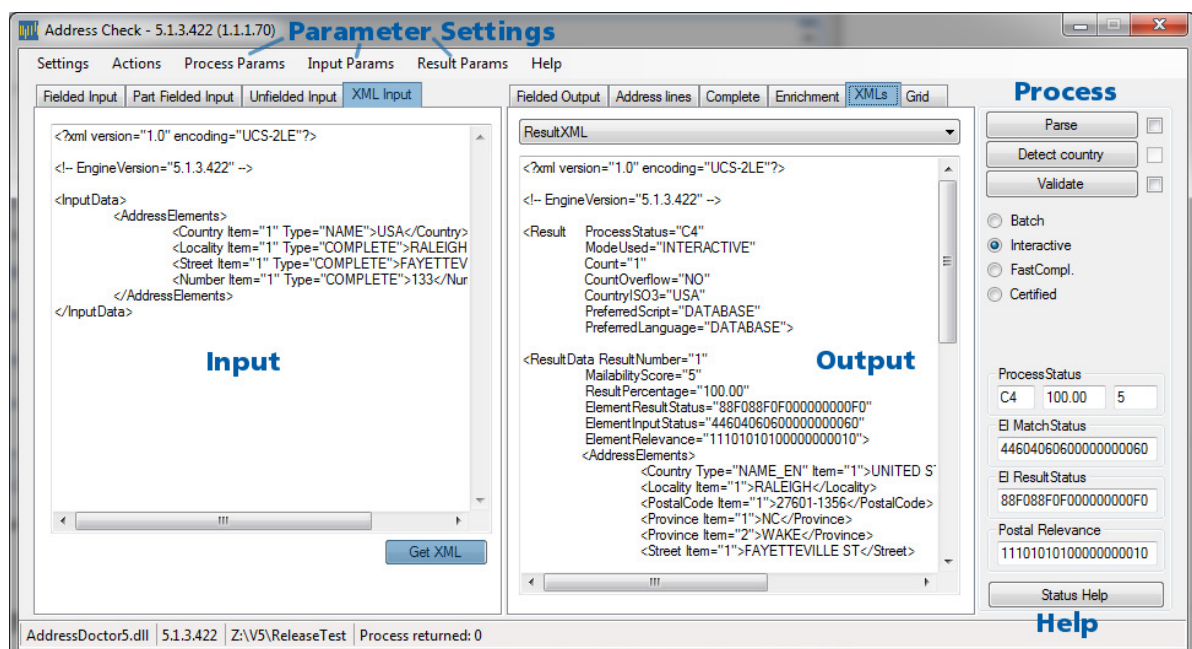
## 7.2 AddressCheck (Windows only)

Starting with AddressDoctor 5.1, the AddressCheck demonstration application featuring a Windows GUI is made available in binary form in the *bin* sub-directory (AD5\_WIN\_32 ZIP archive only). Please note that AddressCheck requires installation of the Microsoft .NET Framework 2.0 or higher to run and is provided as is for experimentation purposes, without warranty or support of *any* kind. Please note that you will have to copy the *AddressDoctor5.dll* from *lib* and the *SetConfig.xml* example from *etc/C* to the *bin* directory, for AddressCheck to be able to locate them (a corresponding *AddressCheck.cfg* that allows for configuring these paths is written upon the first successful initialization).

Like for the ConsoleDemo, please make sure that the minimal SetConfig configuration XML provided contains a valid Unlock Code that you received when purchasing the AddressDoctor library and the correct destination path your reference database files have been unpacked to (see the preceding chapter 7.1 for an example and chapters 6.4 and 6.5 for details, AddressCheck requires a "MaxAddressObjectCount" of at least 6). For 64 Bit Windows systems, the following command might be needed for running AddressCheck:

```
corflags addresscheck.exe /32bit+
```

AddressCheck allows for interactive entry of fielded, partially fielded or unfielded address data (see chapter 6.7) and processing that in different ProcessModes (see chapter 5.11), using the processing parameter (see chapters 5.12, 5.13 and 5.14) settings chosen via menus:



Also, it may be used for producing valid AddressDoctor InputData (hit the "Get XML" button on the "XML Input" Tab after parsing or validating your address entered on one of the other Tabs "Fielded

Input”, “Partially Fielded Input” or “Unfielded Input”), GetConfig, Parameters and Result XML files (see chapter 6.17 as well) for submission to AddressDoctor Support (see chapter 9.3).

The “Status Help” button is very useful in analysis of the Element Input and Result Status values (see chapter 5.23) after processing.

## 8 Sample Address Data for Testing

The following addresses are provided for you to test your implementation of the AddressDoctor library. For each address the status code values are provided and explained. If not otherwise mentioned the addresses have been processed using the Validation process mode Suggestions (INTERACTIVE) that is explained in chapter 5.11.2. The data was input using the FormattedAddressLine element, see chapter 6.7.4. Further example address input and output may be found in chapters 4.1 and 6.15.

### 8.1 Addresses with Status Code Vx

Addresses whose processing results in a status code of Vx were correct on input. Depending on other parameters some minor standardizations may take place.

#### 8.1.1 Correct Address

The following input address is entirely correct, the postal code is properly spaced and the address also is in the proper capitalization for a Swedish address. Because of this, no standardization will have to take place.

```
VASAGATAN 22  
111 20 STOCKHOLM  
SVERIGE
```

The ElementInputStatus (see chapter 5.23.1) would be: 6060006060000000060

With the PreferredScript parameter set to Latin Script (LATIN) and PreferredLanguage set to English (ENGLISH) see Chapter 5.12.2 the result would be (process status value V4, see chapter 5.15):

```
Street: VASAGATAN  
HouseNumber: 22  
POBox:  
Locality: STOCKHOLM  
PostalCode: SE-111 20  
Province: STOCKHOLMS LÄN  
Country: SWEDEN
```

The ElementResultStatus (see chapter 5.23.2) would be: F0F080F0F00000000E0

#### 8.1.2 Address with Exonym replaced

The following input address is written with the English exonym for München (Munich). Because this is a correct name for the city the overall status value would now be V3.

**Prinzregentenstr. 93**  
**81677 Munich**  
**Germany**

The ElementInputStatus would be: 6050006060000000060

With the PreferredLanguage parameter set to English (ENGLISH) the result would be:

Street: **Prinzregentenstr.**  
HouseNumber: **93**  
POBox:  
Locality: **Munich**  
PostalCode: **81677**  
Province: **Bavaria**  
Country: **GERMANY**

The ElementResultStatus would be: F0D080F0F00000000E0

With the PreferredLanguage parameter set to the reference data standard (DATABASE) and CountryType set to "NAME\_DE" the result would then be:

Street: **Prinzregentenstr.**  
HouseNumber: **93**  
POBox:  
Locality: **München**  
PostalCode: **81677**  
Province: **Bayern**  
Country: **DEUTSCHLAND**

The ElementResultStatus would still be: F0D080F0F00000000E0

## 8.2 Addresses with Status Code Cx

Addresses that AddressDoctor can automatically correct will result in a status code of Cx. This indicates that either some address components were missing or incorrect. The returned address can be used instead of the original input address.

### 8.2.1 Address with missing Postal Code

The following input address is basically correct, but it is missing the postal code. The AddressDoctor engine will automatically append the correct postal code and return a status value of C4 for the address.

**2827 yonge street**  
**toronto on**  
**Canada**

The ElementInputStatus would be: 0060606060000000060

With PreferredLanguage set to DATABASE the result would be:

Street: **YONGE STREET**  
 HouseNumber: **2827**  
 POBox:  
 Locality: **TORONTO**  
 PostalCode: **M4N 2J4**  
 Province: **ON**  
 Country: **CANADA**

The ElementResultStatus would be: 80F0F0F0F00000000E0

## 8.2.2 Address with Misspellings in Street and City Name

The following input address is basically correct, but has misspellings in the street and city name. The AddressDoctor engine will automatically correct these misspellings and return a status value of C4 for the address.

**100 GOULD ST**  
**Neu York NY 10038**  
**United States**

The ElementInputStatus would be: 6040604060000000060

With PreferredLanguage set to DATABASE the result would be:

Street: **GOLD ST**  
 HouseNumber: **100**  
 POBox:  
 Locality: **NEW YORK**  
 PostalCode: **10038-1605**  
 Province: **NY**  
 Province Item 2 (County): **NEW YORK**  
 Country: **UNITED STATES**

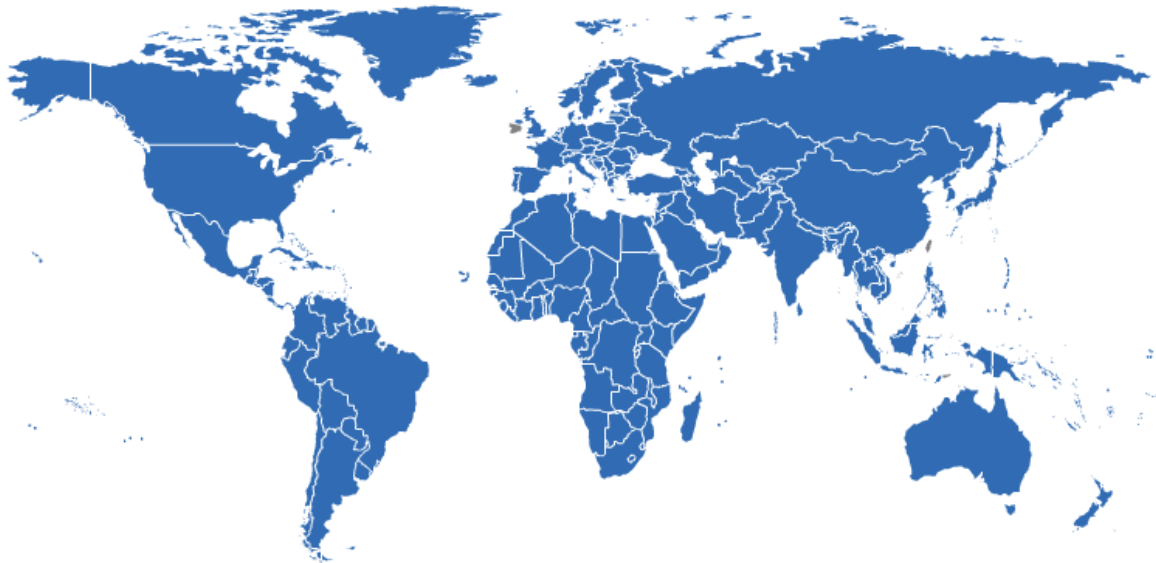
The ElementResultStatus would be: F870F870F00000000E0

## 9 Miscellaneous Topics

This chapter lists various topics that have not been discussed before.

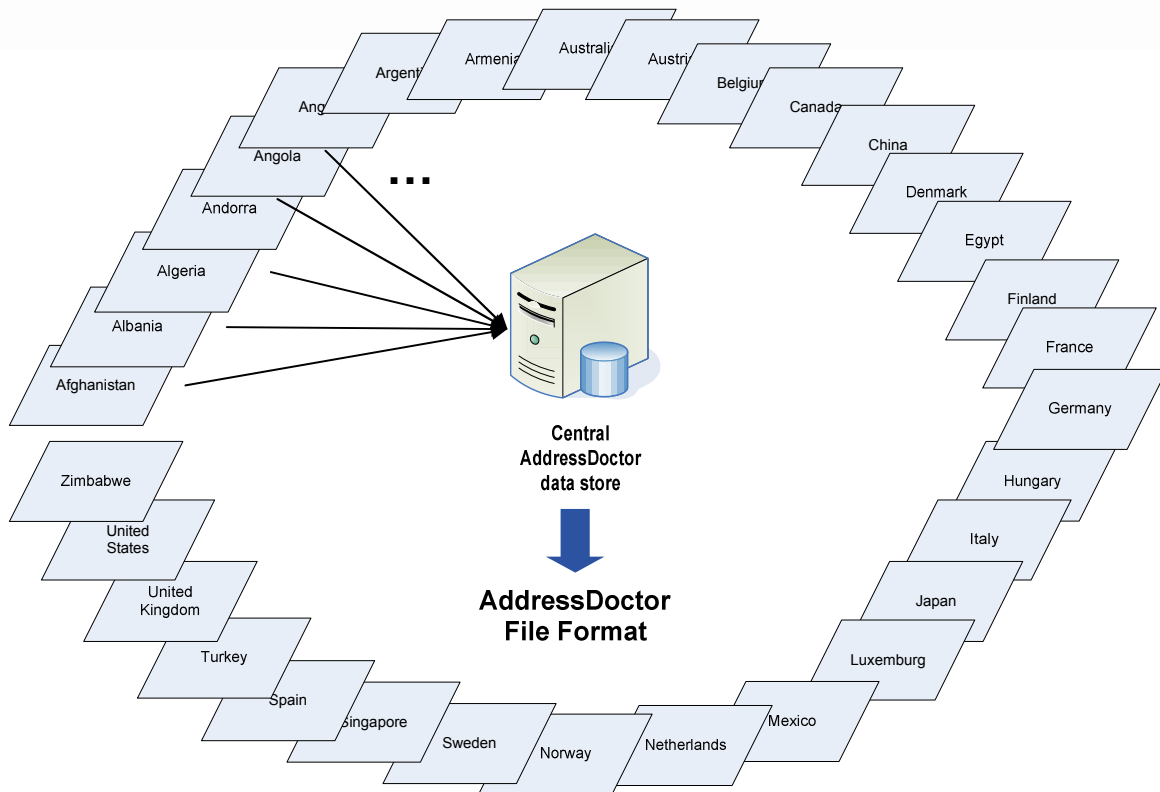
### 9.1 Background on the (Postal) Reference Database

In order to validate postal addresses, so called postal reference data is required. This reference data is typically a collection of locality (city) names, streets, provinces, building numbers, postal codes (ZIP codes), and Post Office Box numbers. AddressDoctor obtains this data from various sources around the world and updates it regularly. The specific update schedule for a country can be found on the AddressDoctor Web Site. Also available online is an interactive map that illustrates the latest updates and data coverage:



Screenshot of the interactive world map (blue countries are available form AddressDoctor) on the AddressDoctor website ([http://www.addressdoctor.com/en/countries\\_data](http://www.addressdoctor.com/en/countries_data))

The reference data is typically provided by postal organizations around the globe. The AddressDoctor development team checks each dataset and then transfers the data into a central data store. This master database is then used to create the postal reference data files in AddressDoctor's unique, proprietary, platform independent file (database) format.



Each country has its own reference database. The databases follow a specific naming scheme that makes it easy to tell them apart.

XXX5BI.MD

XXX represents the ISO3 code of the country. A list of these codes can be found at the AddressDoctor website: [http://www.addressdoctor.com/en/countries\\_data/isocodes.asp](http://www.addressdoctor.com/en/countries_data/isocodes.asp)

The databases are self-contained and platform independent that is they can be used on Windows, Solaris, Unix, or Linux without changes. An external database system or run-time files are not required.

## 9.1.1 Database Format

The AddressDoctor postal reference database is a read only file that stores the postal reference data and all required indexes for fast data access. The data conversion process to create this database format is very resource intensive and is performed on a cluster of high speed computers. While the creation of the database is resource intensive, the access to the data is very fast.

The databases contain information for fuzzy (fault tolerant) searching as well as for all process modes supported by AddressDoctor.

### 9.1.2 Database Size

The postal reference databases for all countries combined (without enrichments) require approximately 15 to 20 GB storage space.

### 9.1.3 Database Updates

Updates to the postal databases are available regularly. Their frequency depends on updates made available to AddressDoctor by the data providers. For some countries monthly updates are available, while others only have an irregular update frequency. Once the AddressDoctor team receives new data it is first checked for accuracy and consistency. Then, the data is transferred into the central data store where enrichment operations take place. Exonyms (alternate names) for places and streets are added and indexes for fast access are stored in the database.

To replace a database with an updated version, simply copy the new database file over the existing file. While doing this, however, no application may be accessing the databases.

## 9.2 Postal Certifications

Some postal operators have instituted a certification process for software vendors. The certification will ensure that the software conforms to the rules and regulations of a specific postal organization. Depending on the intended use of the product, a certification might not produce the best results for poor input data. Certifications tend to be very strict and their major goal is to avoid that improperly addressed mail enters the postal system. The primary goal is not to improve all addresses that can possibly be corrected.

The AddressDoctor 5 engine was certified by USPS for CASS Cycle M and will be regularly submitted to USPS for re-certification. The AddressDoctor 5 engine was furthermore certified by Canada Post for SERP in 2010 and will be regularly submitted to Canada Post for re-certification. In 2011, the engine was as well certified for the AMAS Cycle 2011 and will be regularly submitted to Australia Post for re-certification. To process addresses according to the specific rules defined by postal organizations, a special process mode is available (process mode CERTIFIED, see chapter 5.11.4). Further certifications are planned for France (SNA) and New Zealand (SendRight).

The current engine 5.2.7 meets the following certification cycles:

- CASS Cycle N
- SERP cycle 2011
- AMAS cycle 2011

## 9.3 Support Information

You may contact AddressDoctor Support at: [support@AddressDoctor.com](mailto:support@AddressDoctor.com)

When doing so, please make sure to provide the following 4 XML files (see chapters 6.6 and 6.12 for more details and 10.1 for the corresponding DTDs) in a ZIP archive, after having run them through the ConsoleDemo (see chapter 7.1) application provided by AddressDoctor to check for reproducibility of your issue:

```
SetConfig.xml - may be retrieved using AD_GetConfigSettingsXML() (in Java: getConfigXML())
Parameters.xml - may be retrieved using AD_GetParametersXML() (in Java: getParametersXML())
InputData.xml - may be retrieved using AD_GetInputDataXML() (in Java: getInputDataXML())
Result.xml - may be retrieved using AD_GetResultXML() (in Java: getResultXML())
```

These XML files will provide AddressDoctor support with a basic set of information like the software library and reference database versions as well as the parameter settings used to process an input address facing issues. Additionally, the following information will be needed to assist with your problem:

- Platform version and patch level, the AddressDoctor 5 software library is run on (for supported platforms please make sure to check chapter 2.2), including bitness (32 or 64 bit)
- In case of Java: JDK version and the parameters used to initialize the JVM (e.g. `-Xmx` for maximum heap, see chapter 3.2.2 for examples). Additionally, the Java stack trace (in case of a crash).
- A detailed description of the steps required to trigger the problem
- In case of a crash that could not be reproduced using the AddressDoctor ConsoleDemo, a compact binary test application that actually triggers the crash

A constantly updated list of frequently asked questions (FAQ) can be found on the AddressDoctor Web Site at: <http://www.addressdoctor.com/en/support/FAQ>

## 9.4 Recommended Database Layout for International Addresses

Postal addresses come in numerous varieties around the world. The formats vary in the placement of postal codes, the placement of building numbers, the usage of provinces and the length of address elements. AddressDoctor recommends using just one database layout to store addresses from all countries of the world. The fields of the proposed format are then mapped to the various elements that appear in different countries. As an example the United States have states, while Canada has provinces. Japan is divided into prefectures and Switzerland into cantons. Instead of having separate fields for each, AddressDoctor maps all of these subdivisions to the “province” field. This mapping is done for all address elements that can be represented in an AddressObject (see 5.7). Thus, we recommend storing addresses in a format amenable to AddressObject mapping. As business and consumer addresses vary in the information they require, some fields are not required for consumer addresses. Please note that all fields are of type character to allow for any combination of numeric and alpha content.

Field name	Field length (min.)	Content
Organization	50	Company or Organization name including a company type descriptor such as Inc., AG, or GmbH
Department	50	Department or Mail Stop information
Function	60	Function of the contact
Gender	1	Gender of the contact
FirstName	40	First name of the contact
MiddleName	40	Middle name of the contact
LastName	50	Last name of the contact
Building	50	Building name. Frequently used in the United Kingdom
Subbuilding_1	50	Information that further subdivides a Building, e.g. the floor.
Subbuilding_2	50	Information that further subdivides a Building, e.g. the suite or apartment number.
Street_1	50	Name of the street or thoroughfare
Street_2	50	Dependent street or thoroughfare
Number_1	15	Number of a Building/House in a street. Placement varies by country.
Number_2	15	Number of a Building/House in a dependent street. Placement varies by country.
DeliveryService_1	50	Code of the respective post office in charge of delivery.
DeliveryService_2	50	Post Box descriptor (POBox, Postfach, Case Postale etc.) and number.
Locality_1	50	Primary place name. Typically a "province" is subdivided into localities. Some countries may contain yet another hierarchy level for subdividing provinces. Examples are counties in the US and Kreise in Germany
Locality_2	50	Dependent place name that further subdivides a Locality. Examples are colonias in Mexico, Urbanisaciones in Spain
Locality_3	50	Dependent place name that further subdivides a Locality. An example would be Mahalle in Turkey.
SortingCode	10	Speeds up delivery in certain countries for large localities, like for example Prague or Dublin.
PostalCode	10	Postal code or ZIP code.
Province_1	50	Store the state, province, canton, prefecture or other sub-division of a country.
Province_2	50	Dependent province information that further

CountryName	50	subdivides a province. An example would be a US county. Optionally needed if required for display. It is recommended to just store the ISO code so that the country name can be displayed in any language. ISO alpha3 code according to ISO 3166. Can be used to generate the name of a country in any language.
CountryISO	3	

When data has been stored in the format suggested above, this is of major benefit when using AddressDoctor's functionality for automatically generating addresses for printing and display.

## 10 Appendix

### 10.1 API Document Type Definitions

The following accompanying DTD files are provided in the documentation package (see chapter 3.1.2):

SetConfig.dtd - Configuration settings passed with `AD_Initialize()`

Parameters.dtd - Parameters passed with `AD_SetParametersXML()` or `AD_Initialize()` (with SetConfig)

InputData.dtd - Structure of data input as XML, using `AD_SetInputDataXML()` and `AD_GetInputDataXML()`

Result.dtd - Structure of the XML result from `AD_GetResultXML()`

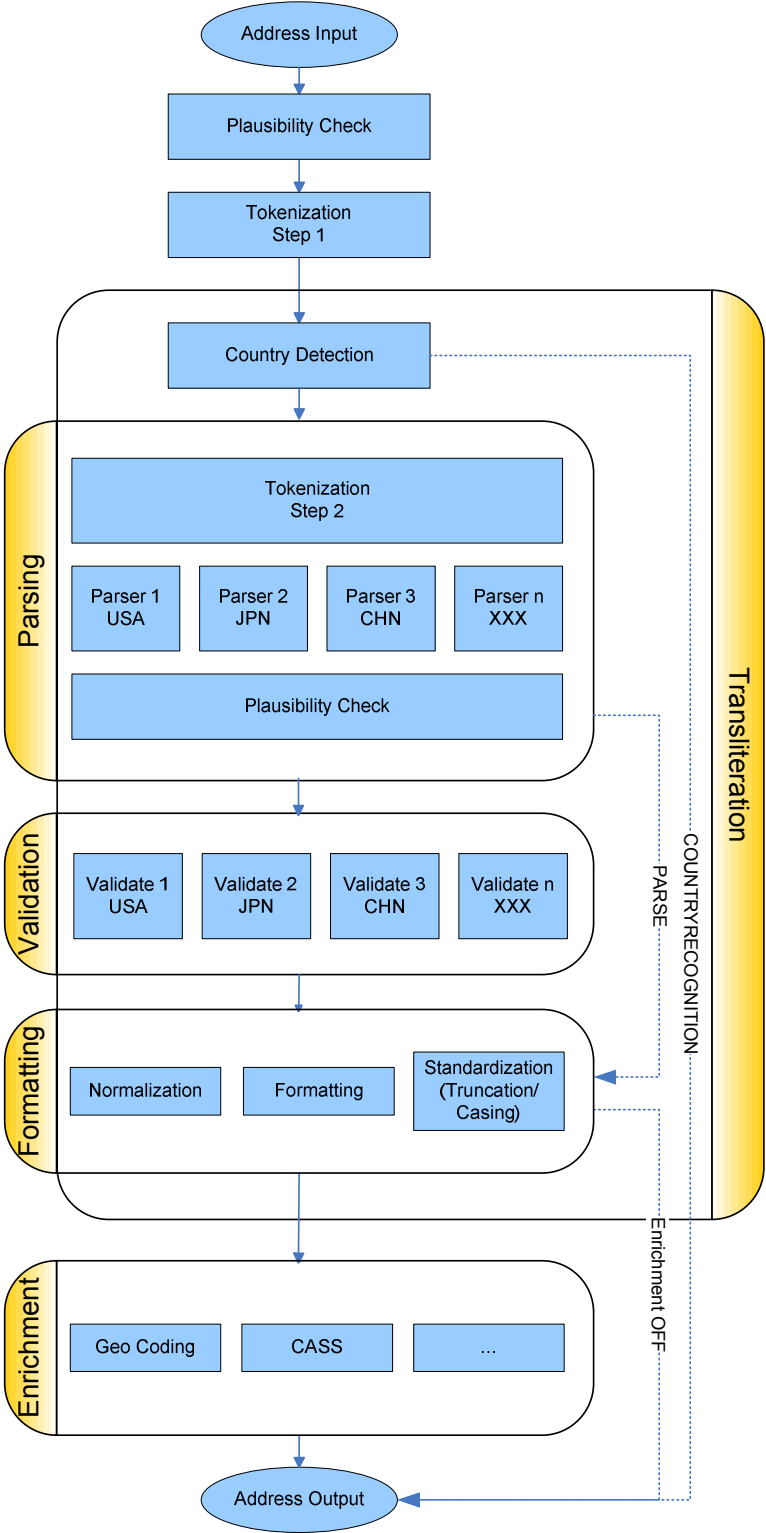
GetConfig.dtd - Structure of the XML result from `AD_GetConfigSettingsXML()`

**IMPORTANT Notice:** As AddressDoctor sees the DTDs referred to here as part of the API definition, where disruptive changes must be minimized, these files may at any given time contain elements without apparent functionality. Please do refer to the chapters 5 and 60 of this document to understand what functionality is actually available in AddressDoctor 5 and how it should be used. Simply relying on the DTDs with their comments will not suffice as basis for a successful integration of the AddressDoctor 5 software library!

### 10.2 API Reference

For details on the available function calls and parameters, see the accompanying Application Programming Interface Reference provided in HTML format for C and Java as part of the documentation package (see chapter 3.1.2). Again, the API Reference should not be used without prior consultation of this documentation, as explained before in the notice to chapter 10.1.

## 10.3 Schematic AddressDoctor Processing Flow



## 10.4 AddressElement Output Examples

See chapter 5.9 for an explanation of AddressDoctor Items, the following characteristic representations of AddressElement Items are based on the example address formats and samples presented on the AddressDoctor website: [http://www.addressdoctor.com/de/countries\\_data/addressformats.asp](http://www.addressdoctor.com/de/countries_data/addressformats.asp)

Legend:

- Typically needed for correct address representation
- Typically not populated for a correct address\*
- Not available through the AddressDoctor 5 API
- Will be needed for future country support
- Not address relevant (will be copied over to output)

\*but may contain certain input elements copied over to output

### 10.4.1 Argentina

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
<b>Organization</b>	Company Name					
<b>Contact</b>	Mr. Doe	Mrs. Doe				
<b>DeliveryService</b>						
<b>SubBuilding</b>						
<b>Building</b>						
<b>Street</b>	Street Name					
<b>Number</b>	House Number					
<b>Province</b>	Province					
<b>PostalCode</b>	Postal Code					
<b>Locality</b>	Locality					
<b>Country</b>	Argentina					
<b>Key</b>	Key1	Key2	Key3			

### 10.4.2 Australia

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
<b>Organization</b>	Company Name					
<b>Contact</b>	Mr. Doe	Mrs. Doe				
<b>DeliveryService</b>						
<b>SubBuilding</b>	Apartment					
<b>Building</b>						
<b>Street</b>	Street Name & Type					
<b>Number</b>	House Number					
<b>Province</b>	Province Abbreviation					
<b>PostalCode</b>	Postal Code					
<b>Locality</b>	Locality					
<b>Country</b>	Australia					
<b>Key</b>	Key1	Key2	Key3			

## 10.4.3 Austria

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding	Flat/Entrance/Floor					
Building						
Street	Street					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Locality					
Country	Austria					
Key	Key1	Key2	Key3			

## 10.4.4 Belarus

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street Name					
Number	House Number					
Province	Rayon					
PostalCode	Postal Code					
Locality	Locality					
Country	Belarus					
Key	Key1	Key2	Key3			

## 10.4.5 Belgium

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street					
Number	House Number					
Province						
PostalCode	Postal Code					
Locality	Locality					
Country	Belgium					
Key	Key1	Key2	Key3			

## 10.4.6 Bulgaria

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding	Floor					
Building						
Street	Street Name					
Number	House Number					
Province						
PostalCode	Postal Code					
Locality	Locality					
Country	Bulgaria					
Key	Key1	Key2	Key3			

## 10.4.7 Canada

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street					
Number	House Number					
Province	Province Abbreviation					
PostalCode	Postal Code					
Locality	Locality					
Country	Canada					
Key	Key1	Key2	Key3			

## 10.4.8 Chile

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding	Floor					
Building						
Street	Street Name					
Number	House Number					
Province						
PostalCode	Postal Code					
Locality	Locality					
Country	Chile					
Key	Key1	Key2	Key3			

## 10.4.9 China

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
<b>Organization</b>	Company Name					
<b>Contact</b>	Mr. Doe	Mrs. Doe				
<b>DeliveryService</b>						
<b>SubBuilding</b>	Apartment					
<b>Building</b>	Building					
<b>Street</b>	Street					
<b>Number</b>	House Number					
<b>Province</b>	Province					
<b>PostalCode</b>	Postal Code					
<b>Locality</b>	Locality					
<b>Country</b>	China					
<b>Key</b>	Key1	Key2	Key3			

## 10.4.10 Croatia

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
<b>Organization</b>	Company Name					
<b>Contact</b>	Mr. Doe	Mrs. Doe				
<b>DeliveryService</b>						
<b>SubBuilding</b>						
<b>Building</b>						
<b>Street</b>	Street Name					
<b>Number</b>	House Number					
<b>Province</b>	Province					
<b>PostalCode</b>	Postal Code					
<b>Locality</b>	Locality					
<b>Country</b>	Croatia					
<b>Key</b>	Key1	Key2	Key3			

## 10.4.11 Cyprus

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
<b>Organization</b>	Company Name					
<b>Contact</b>	Mr. Doe	Mrs. Doe				
<b>DeliveryService</b>						
<b>SubBuilding</b>						
<b>Building</b>						
<b>Street</b>	Street Name					
<b>Number</b>	House Number					
<b>Province</b>	Province					
<b>PostalCode</b>	Postal Code					
<b>Locality</b>	Locality					
<b>Country</b>	Cyprus					
<b>Key</b>	Key1	Key2	Key3			

## 10.4.12 Czech Republic

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding	Entrance					
Building						
Street	Street					
Number	House Number					
Province						
PostalCode	Postal Code					
Locality	Locality & Sorting Code					
Country	Czech Republic					
Key	Key1	Key2	Key3			

## 10.4.13 Denmark

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street Name					
Number	House Number					
Province						
PostalCode	Postal Code					
Locality	Locality & Sorting Code					
Country	Denmark					
Key	Key1	Key2	Key3			

## 10.4.14 Ecuador

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street Name					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Locality					
Country	Ecuador					
Key	Key1	Key2	Key3			

## 10.4.15 Estonia

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Parish	Village or Borough				
Country	Estonia					
Key	Key1	Key2	Key3			

## 10.4.16 Finland

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street Name					
Number	House Number					
Province						
PostalCode	Postal Code					
Locality	Locality					
Country	Finland					
Key	Key1	Key2	Key3			

## 10.4.17 Germany

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street					
Number	House Number					
Province						
PostalCode	Postal Code					
Locality	Locality					
Country	Germany					
Key	Key1	Key2	Key3			

## 10.4.18 Greece

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street Name					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Locality					
Country	Greece					
Key	Key1	Key2	Key3			

## 10.4.19 Guatemala

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Locality	Zona/Colonia				
Country	Guatemala					
Key	Key1	Key2	Key3			

## 10.4.20 Hong Kong

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding	Floor	Apartment				
Building	Building					
Street	Street Name					
Number	House Number					
Province						
PostalCode	Postal Code					
Locality	Locality					
Country	Hong Kong					
Key	Key1	Key2	Key3			

## 10.4.21 Hungary

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding	Floor					
Building						
Street	Street Name					
Number	House Number					
Province						
PostalCode	Postal Code					
Locality	Locality					
Country	Hungary					
Key	Key1	Key2	Key3			

## 10.4.22 India

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street					
Number	House Number					
Province						
PostalCode	Postal Code					
Locality	Locality		Dependent Locality			
Country	India					
Key	Key1	Key2	Key3			

## 10.4.23 Italy

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street Type and Name					
Number	House Number					
Province	Province Abbreviation					
PostalCode	Postal Code					
Locality	Locality					
Country	Italy					
Key	Key1	Key2	Key3			

## 10.4.24 Japan

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building	Building					
Street	Street Name					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Locality	Dependent Locality				
Country	Japan					
Key	Key1	Key2	Key3			

## 10.4.25 Jordan

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Locality					
Country	Jordan					
Key	Key1	Key2	Key3			

## 10.4.26 Luxembourg

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street Name					
Number	House Number					
Province						
PostalCode	Postal Code					
Locality	Locality					
Country	Luxembourg					
Key	Key1	Key2	Key3			

## 10.4.27 Madagascar

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding	Lot Section Number					
Building						
Street	Street					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Locality					
Country	Madagascar					
Key	Key1	Key2	Key3			

## 10.4.28 Malaysia

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding	Floor					
Building	Building					
Street	Street					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Locality					
Country	Malaysia					
Key	Key1	Key2	Key3			

## 10.4.29 Mexico

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street					
Number	House Number					
Province	Province Abbreviation					
PostalCode	Postal Code					
Locality	Locality	Village	Colonia			
Country	Mexico					
Key	Key1	Key2	Key3			

## 10.4.30 Netherlands

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street Name					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Locality					
Country	Netherlands					
Key	Key1	Key2	Key3			

## 10.4.31 New Zealand

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street Name					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Locality	Dependent Locality				
Country	New Zealand					
Key	Key1	Key2	Key3			

## 10.4.32 Norway

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street Name					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Locality					
Country	Norway					
Key	Key1	Key2	Key3			

## 10.4.33 Pakistan

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Locality	Area				
Country	Pakistan					
Key	Key1	Key2	Key3			

## 10.4.34 Panama

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street					
Number	House Number					
Province	Provincia de Province					
PostalCode	Postal Code					
Locality	Post Office					
Country	Panama					
Key	Key1	Key2	Key3			

## 10.4.35 Philippines

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Locality	Municipality	Barangay			
Country	Philippines					
Key	Key1	Key2	Key3			

## 10.4.36 Poland

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street Type and Name					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Locality					
Country	Poland					
Key	Key1	Key2	Key3			

## 10.4.37 Portugal

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street Type and Name					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Locality					
Country	Portugal					
Key	Key1	Key2	Key3			

## 10.4.38 Russia

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street					
Number	House Number					
Province	Oblast or Kray or Respublika					
PostalCode	Postal Code					
Locality	Locality					
Country	Russian Federation					
Key	Key1	Key2	Key3			

## 10.4.39 Singapore

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding	Floor	Apartment				
Building	Building					
Street	Street					
Number	House Number					
Province						
PostalCode	Postal Code					
Locality	Locality					
Country	Singapore					
Key	Key1	Key2	Key3			

## 10.4.40 Slovakia

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Delivery Office & Sorting Code					
Country	Slovakia					
Key	Key1	Key2	Key3			

## 10.4.41 Slovenia

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street Name and Type					
Number	House Number					
Province						
PostalCode	Postal Code					
Locality	Locality					
Country	Slovenia					
Key	Key1	Key2	Key3			

## 10.4.42 South Africa

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street Name					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Locality	Dependent Locality				
Country	South Africa					
Key	Key1	Key2	Key3			

## 10.4.43 Sweden

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street Name					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Locality					
Country	Sweden					
Key	Key1	Key2	Key3			

## 10.4.44 Switzerland

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street Name					
Number	House Number					
Province	Kanton					
PostalCode	Postal Code					
Locality	Locality					
Country	Switzerland					
Key	Key1	Key2	Key3			

## 10.4.45 Turkey

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService	PO Box					
SubBuilding						
Building						
Street	Street					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Administrative District		Mahalle			
Country	Turkey					
Key	Key1	Key2	Key3			

## 10.4.46 Ukraine

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street Name					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Locality					
Country	Ukraine					
Key	Key1	Key2	Key3			

## 10.4.47 United Arab Emirates

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService	PO Box & Number					
SubBuilding						
Building						
Street						
Number						
Province						
PostalCode						
Locality	Locality					
Country	United Arab Emirates					
Key	Key1	Key2	Key3			

## 10.4.48 United Kingdom

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building	Building					
Street	Street Name					
Number	House Number					
Province	Province					
PostalCode	Postal Code					
Locality	Locality					
Country	United Kingdom					
Key	Key1	Key2	Key3			

## 10.4.49 USA

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street Name with Type and Direction					
Number	House Number					
Province	Province Abbreviation	County				
PostalCode	ZIP+4					
Locality	Locality					
Country	USA					
Key	Key1	Key2	Key3			

## 10.4.50 Venezuela

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Organization	Company Name					
Contact	Mr. Doe	Mrs. Doe				
DeliveryService						
SubBuilding						
Building						
Street	Street					
Number						
Province	Province Abbreviation					
PostalCode	Postal Code					
Locality	Locality	Población or Sector or Urbanización				
Country	Venezuela					
Key	Key1	Key2	Key3			

## 10.5 Province Output

When retrieving the Province of a validated or corrected address from the AddressObject, you will receive either the Province name or the Abbreviation, according to the postal rules of this country. The following table shows what is returned for a specific country:

ISO Code	Province output form
----------	----------------------

ABW	Province name
AFG	Province name
AGO	Province name
AIA	Province name
ALB	Province name
AND	Province name
ANT	Province name
ARE	Province name
ARG	Province name
ARM	Province name
ATA	Province name
ATG	Province name
AUS	Abbreviation
AUT	Province name
AZE	Province name
BDI	Province name
BEL	Province name
BEN	Province name
BFA	Province name
BGD	Province name
BGR	Province name
BHR	Province name
BHS	Province name
BIH	Province name
BLR	Province name
BLZ	Province name
BMU	Province name
BOL	Province name
BRA	Abbreviation
BRB	Province name
BRN	Province name
BTN	Province name
BWA	Province name
CAF	Province name
CAN	Abbreviation
CHE	Abbreviation
CHL	Province name
CHN	Province name
CIV	Province name
CMR	Province name
COD	Province name
COG	Province name

COK	Province name
COL	Province name
COM	Province name
CPV	Province name
CRI	Province name
CUB	Province name
CYM	Province name
CYP	Province name
CZE	Province name
DEU	Province name
DJI	Province name
DMA	Province name
DNK	Province name
DOM	Province name
DZA	Province name
ECU	Province name
EGY	Province name
ERI	Province name
ESH	Province name
ESP	Province name
EST	Province name
ETH	Province name
FIN	Province name
FJI	Province name
FLK	Province name
FRA	Province name
FRO	Province name
GAB	Province name
GBR	Province name
GEO	Province name
GHA	Province name
GIB	Province name
GIN	Province name
GMB	Province name
GNB	Province name
GNQ	Province name
GRC	Province name
GRD	Province name
GRL	Province name
GTM	Province name
GUY	Province name
HKG	Province name

HND	Province name
HRV	Province name
HTI	Province name
HUN	Province name
IDN	Province name
IND	Province name
IOT	Province name
IRN	Province name
IRQ	Province name
ISL	Province name
ISR	Province name
ITA	Abbreviation
JAM	Province name
JOR	Province name
JPN	Province name
KAZ	Province name
KEN	Province name
KGZ	Province name
KHM	Province name
KIR	Province name
KNA	Province name
KOR	Province name
KWT	Province name
LAO	Province name
LBN	Province name
LBR	Province name
LBY	Province name
LCA	Province name
LIE	Province name
LKA	Province name
LSO	Province name
LTU	Province name
LUX	Province name
LVA	Province name
MAR	Province name
MCO	Province name
MDA	Province name
MDG	Province name
MDV	Province name
MEX	Abbreviation
MKD	Province name
MLI	Province name

MLT	Province name
MMR	Province name
MNG	Province name
MOZ	Province name
MRT	Province name
MSR	Province name
MUS	Province name
MWI	Province name
MYS	Province name
NAM	Province name
NER	Province name
NFK	Province name
NGA	Province name
NIC	Province name
NIU	Province name
NLD	Province name
NOR	Province name
NPL	Province name
NRU	Province name
NZL	Province name
OMN	Province name
PAK	Province name
PAN	Province name
PCN	Province name
PER	Province name
PHL	Province name
PNG	Province name
POL	Province name
PRK	Province name
PRT	Province name
PRY	Province name
QAT	Province name
ROU	Province name
RUS	Province name
RWA	Province name
SAU	Province name
SCG	Province name
SDN	Province name
SEN	Province name
SGP	Province name
SGS	Province name
SHN	Province name

SLB	Province name
SLE	Province name
SLV	Province name
SMR	Province name
SOM	Province name
STP	Province name
SUR	Province name
SVK	Province name
SVN	Province name
SWE	Province name
SWZ	Province name
SYC	Province name
SYR	Province name
TCA	Province name
TCD	Province name
TGO	Province name
THA	Province name
TJK	Province name
TKL	Province name
TKM	Province name
TON	Province name
TTO	Province name
TUN	Province name
TUR	Province name
TUV	Province name
TZA	Province name
UGA	Province name
UKR	Province name
URY	Province name
USA	Abbreviation
UZB	Province name
VAT	Province name
VCT	Province name
VEN	Province name
VGB	Province name
VNM	Province name
VUT	Province name
WSM	Province name
YEM	Province name
ZAF	Province name
ZMB	Province name
ZWE	Province name

## 10.6 Reference Data Copyright Notices

### **Australia:**

Copyright 2009. Based on data provided under licence from PSMA Australia Limited (www.pasma.com.au).

### **Canada:**

In case Licensed User licensed the Canadian reference database, it contains Postal Code OM data copied under license from Canada Post Corporation. The Canada Post Corporation file from which this data was copied is from the most current data available from Canada Post Corporation at the time AddressDoctor made the data available to Licensed User respective Integrator.

### **Great Britain:**

You are receiving or have received information which is derived from databases (or parts or extracts thereof) of which Royal Mail is the owner or creator, or otherwise authorised to use (the "Data"). Royal Mail owns, or is licensed, all Intellectual Property Rights which subsist in and/or relate to that Data from time to time. You must not at any time copy, reproduce, publish, sell, let, lend, extract, reutilise or otherwise part with possession or control of or relay or disseminate any part of this information or use it for any purpose other than your own private or internal use.

### **New Zealand:**

The address data within the PAF is sourced from New Zealand Post, Land Information New Zealand and the Crown. New Zealand Post and Crown copyright reserved.

### **United States of America:**

© United States Postal Service® 2009. Prices are not established, controlled or approved by the United States Postal Service®. The following trademarks and registrations are owned by the USPS®: CASS Certified™, CASST™, DPV™, United States Postal Service®, USPS®, ZIP + 4®, ZIP Code™, ZIP™

### **Geocodes:**

The data ("Data") is provided for your personal, internal use only and not for resale. It is protected by copyright, and is subject to the terms and conditions which are agreed to by you, on the one hand, and AddressDoctor ("AddressDoctor") and its licensors (including their licensors and suppliers) on the other hand.

© 2009 NAVTEQ. All rights reserved.

The Data for areas of Canada includes information taken with permission from Canadian authorities, including: © Her Majesty the Queen in Right of Canada, © Queen's Printer for Ontario, © Canada Post Corporation, GeoBase®, © Department of Natural Resources Canada. All rights reserved.

NAVTEQ holds a non-exclusive license from the United States Postal Service® to publish and sell ZIP+4® information.

© United States Postal Service® 2009. Prices are not established, controlled or approved by the United States Postal Service®. The following trademarks and registrations are owned by the USPS: United States Postal Service, USPS, and ZIP+4.

Data for Europe and World Markets:

Territory	Notice
Australia	"Copyright. Based on data provided under license from PSMA Australia Limited (www.pdma.com.au)."
Austria	"© Bundesamt für Eich- und Vermessungswesen"
Croatia, Cyprus, Estonia, Latvia, Lithuania, Moldova, Poland, Slovenia & Ukraine	"© EuroGeographics"
France	"Source: © IGN 2009 – BD TOPO ®"
Germany	"Die Grundlagendaten wurden mit Genehmigung der zuständigen Behörden entnommen"
Great Britain	"Based upon Crown Copyright material."
Greece	"Copyright Geomatics Ltd."
Hungary	"Copyright © 2003; Top-Map Ltd."
Italy	"La Banca Dati Italiana è stata prodotta usando quale riferimento anche cartografia numerica ed al tratto prodotta e fornita dalla Regione Toscana."
Jordan	"© Royal Jordanian Geographic Centre"
Norway	"Copyright © 2000; Norwegian Mapping Authority"
Portugal	"Source: IgeoE – Portugal"
Spain	"Información geográfica propiedad del CNIG"
Sweden	"Based upon electronic data: © National Land Survey Sweden."
Switzerland	"Topografische Grundlage: © Bundesamt für Landestopographie."

## 11 Glossary

### ISO Country Codes

The international standard for Country Codes ISO 3166 is one of the most widely used standards maintained by ISO TC 46. It provides a standard numeric and 2-letter and 3-letter alphabetic codes for 240 countries or areas of special sovereignty. First released in 1974, ISO 3166 has grown to encompass three parts, including two new sections on codes for subdivisions (states, regions, major cities, etc.) and a listing of retired codes. For a list, see [http://www.addressdoctor.com/en/countries\\_data/isocodes.asp](http://www.addressdoctor.com/en/countries_data/isocodes.asp)

### Normalization

Normalization refers to the consolidation of address element descriptors, e.g. to treat "Street", "ST" and "St." all as equivalent to "St.". It is on one hand applied in an internal step before validation to aid in matching. On the other hand, normalization is used to produce address element output meeting the postal regulations for each country.

### Parsing

Parsing is the capability to split an unstructured address string into meaningful entities. That means an unstructured address such as

Platon Data Technology GmbH  
Röntgenstr. 9  
D-67133 Maxdorf

would be split into  
Company: Platon Data Technology GmbH  
Street: Röntgenstr.  
Building Number: 9  
Postal Code: 67133  
Locality: Maxdorf  
Country: Germany

Parsing can also be used to rearrange incorrectly fielded data.

### Romanization

Romanization is a method of using letters of the Roman alphabet (ABCD...) to recreate the sounds of a language whose writing system may or may not use the Roman alphabet. A Chinese Hanzi romanization system would thus be a method of using the Roman alphabet to pronounce Chinese Hanzi characters.

### Standardization

Postal regulations and target database standards require address element length and casing to be adjusted or standardized.

## Tokenization

Address input needs to be separated into tokens for mapping these to address elements/items.

## Transformation

Transformation is the process of changing one character into other characters of the same character set. A string like 'Änderung' would be transformed using the Transform method to e.g a HTML encoded version of this string: '&Auml;nderung'.

## Transcription / Transliteration

Transcription / Transliteration is the process of changing one character of one character set into other characters of another character set, such as converting from Greek to Latin, or Japanese Katakana to Latin. This conversion makes use of either the sound of the character or the spelling.

## Unified Ideographs

Unified ideographs are characters of the CJK writing system. They consist of Chinese Hanzi, Japanese Kanji, and Korean Hanja.

## Validation

Validation is the process of checking individual address elements against postal reference data. The validation process will, for instance, verify if a postal code or a locality exist. The validation process will also check if a street name is spelled properly and if the postal code and locality combination given is correct for the building number provided for this street.

## 12 Index

- Batch 44
- BPEL 89
- Call Center 91
- Carrier Route 19
- CASS 38, 45, 94
- Certified 45
- CJK 142
- Cloud Computing 93
- Country Recognition 45
- CRM 91
- CSLL 80
- Cyrillic 34
- DAL 80
- Data
  - Entry, Point of 91
  - fielded 12, 36, 77
  - Integration 90
  - partially fielded 11, 77
  - unfielded 11, 36, 39, 45, 77
- DTD 89
- EBCDIC 32
- EOL 89
- ESB 93
- Exonym 48
- FAL 81
- False Positives 38
- Fast Completion 44
- File Handle 19
- Formatting 48
- Greek 33
- Interactive 44
- ISO 141
- ISO 3166 18, 114, 141
- Kana
  - Katakana 33
- Kanji 33
- MDM 90
- Parsing 36, 45
- PMS 91
- POS 91
- SaaS 93
- SERP 97
- SNA 98, 99
- SOA 93
- SOAP 93
- Standardization 48
- Transliteration 33
- UPU 48
- USPS 45, 94
- Validation 37
- Web Shop 91
- XML 89
- XSLT 93