

# **International Domain Name Software Development Kit User's Guide**

**COPYRIGHT NOTIFICATION**

Copyright © 2010 VeriSign, Inc., as an unpublished work. All rights reserved.

Copyright laws and international treaties protect this document, and any VeriSign product to which it relates.

**VERISIGN PROPRIETARY INFORMATION**

This document is the property of VeriSign, Inc. It may be used by recipient only for the purpose for which it was transmitted and shall be returned upon request or when no longer needed by recipient. It may not be copied or communicated without the prior written consent of VeriSign.

**DISCLAIMER AND LIMITATION OF LIABILITY**

VeriSign, Inc. has made every effort to ensure the accuracy and completeness of all information in this document. However, VeriSign, Inc. assumes no liability to any party for any loss or damage caused by errors or omissions or by statements of any kind in this document, its updates, supplements, or special editions, whether such errors, omissions, or statements result from negligence, accident, or any other cause. VeriSign, Inc. assumes no liability arising out of applying or using the product and no liability for incidental or consequential damages arising from using this document. VeriSign, Inc. disclaims all warranties regarding the information contained herein (whether expressed, implied, or statutory) including implied warranties of merchantability or fitness for a particular purpose.

VeriSign, Inc. makes no representation that interconnecting products in the manner described herein will not infringe upon existing or future patent rights nor do the descriptions contained herein imply granting any license to make, use, or sell equipment or products constructed in accordance with this description. VeriSign, Inc. reserves the right to make changes to any information herein without further notice.

**NOTICE AND CAUTION**

Concerning U.S. Patent or Trademark Rights

The inclusion in this document, the associated online file, or the associated software of any information covered by any patent, trademark, or service mark rights shall not constitute nor imply a grant of, or authority to exercise, any right or privilege protected by such patent, trademark, or service mark. All such rights and privileges are vested in the patent, trademark, or service mark owner, and no other person may exercise such rights without express permission, authority, or license secured from the patent, trademark, or service mark owner.

10/29/2010

**VERISIGN® NAMING SERVICES**

Email [Info@verisign-grs.com](mailto:Info@verisign-grs.com)

Internet <http://www.verisign.com>

## CONTENTS

Introduction .....	5
Purpose .....	5
Overview .....	5
Installation .....	7
Overview .....	7
Directory Structure .....	7
Compilation .....	10
Overview .....	10
Compiling the Java IDN SDK .....	10
Compiling the C IDN SDK .....	11
Input Format .....	14
Overview .....	14
ASCII .....	14
UNICODE (UTF-32) .....	14
UNICODE (UTF-16) .....	14
NATIVE .....	15
OUTPUT FORMAT .....	16
Overview .....	16
TOOLS .....	18
Overview .....	18
CONVENTIONS .....	18
COMMON TOOLS .....	18
bidi .....	18
idna .....	19
normalize .....	19
punycode .....	19
unicode .....	20
JAVA-ONLY TOOLS .....	21
Overview .....	21
base32 .....	21
convert .....	21
dce .....	21
encvariants .....	22
hex .....	22
native .....	22
randomdata .....	22

race.....	23
Conversion Examples .....	24
Overview .....	24
EXAMPLE 1 .....	24
EXAMPLE 2 .....	24
EXAMPLE 3 .....	24
EXAMPLE 4 .....	25
EXAMPLE 5 .....	25
VERISIGN IDN SDK DISCUSSION LIST .....	26
Overview .....	26
APPENDICES .....	27
ERROR CODES .....	27
REFERENCES .....	27

## INTRODUCTION

### PURPOSE

This guide provides an overview of the VeriSign Internationalized Domain Names Software Development Kit (IDN SDK). It is appropriate for all users of the SDK and will give readers a better understanding of how the IDN SDK can facilitate IDN registration and maintenance. Although this guide offers details about the IDN SDK tools, it does not contain specific information about the underlying conversion software. For a detailed description of the IDN SDK application programming interface (API) as well as sample code, please refer to the VeriSign IDN SDK Programmer's Guide.

The IDN SDK contains the following items:

<b>Java API</b>	A set of Java objects that implement the Internationalized Domain Names for Applications (IDNA) Request for Comment (RFC).
<b>C API</b>	A set of C routines that implement the IDNA RFC.
<b>Sample code</b>	Code examples that illustrate how to use the objects and routines provided in the SDK. This sample code is included in the Programmer's Guide.
<b>Tools</b>	Executable programs that wrap around API calls, allowing users to execute the SDK algorithms from the command line.
<b>Documentation</b>	A User's Guide, a Programmer's Guide, a Registrar's Guide, and Javadoc documentation.

### OVERVIEW

Internationalized Domain Names in Applications (IDNA) is a collection of standards that allow client applications to convert some mnemonic strings expressed in Unicode to an ASCII-Compatible Encoding (ACE) form that contains only letters, digits, and hyphens and is a valid DNS label. The first version of IDNA was published in 2003 and is referred to here as IDNA2003 to differentiate it from the current version, which is known as IDNA2008. IDNA2003 used only Unicode version 3.2. To accommodate new characters added in new versions of Unicode, IDNA2008 decouples its rules from any particular version of Unicode. Instead, the attributes of new characters in Unicode, supplemented by a small number of exception cases, determine how and whether the characters can be used in IDNA labels.

IDNA2008 RFC documents specify new rules and algorithms for determining invalid and/or disallowed code points in IDN registration independent of Unicode version. A new set of

rules has also been defined for bidirectional (Bidi) characters to ensure that entire character sets from certain languages that were once prohibited in IDN registrations (e.g., Divehi) are now allowed. The IDNA2008 Protocol document sets forth an algorithm for determining whether the code points contained in an IDN are valid.

VeriSign IDN SDK version 4.0 implements the rules and restrictions for IDNs as specified in the IDNA2008 RFCs.

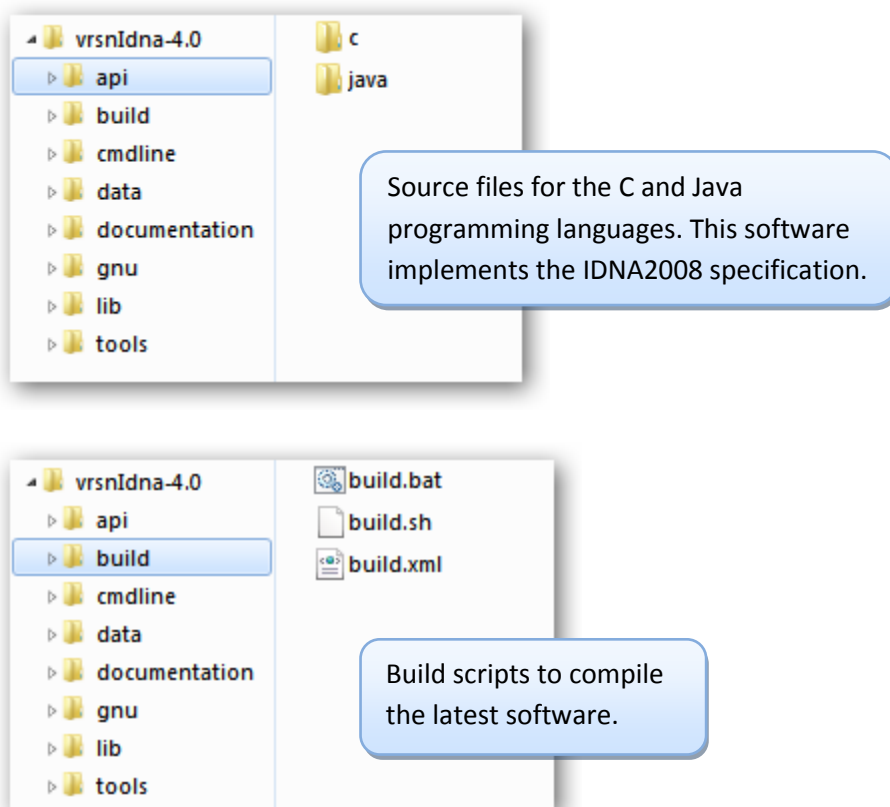
## INSTALLATION

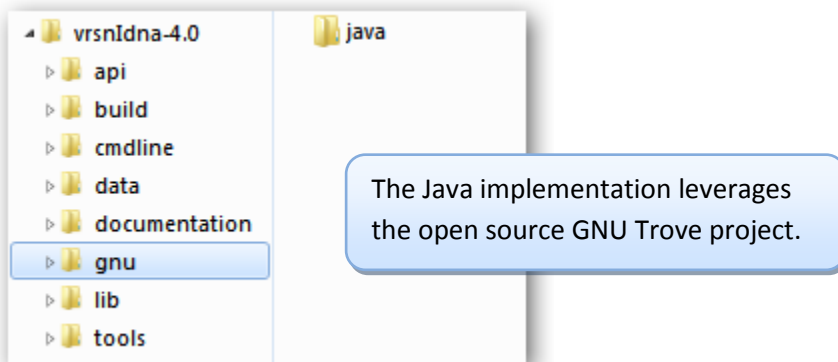
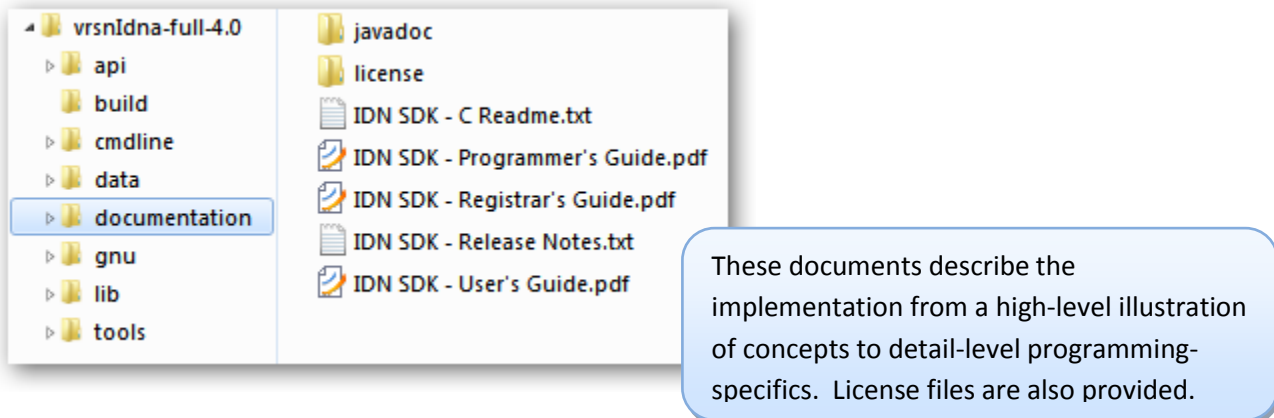
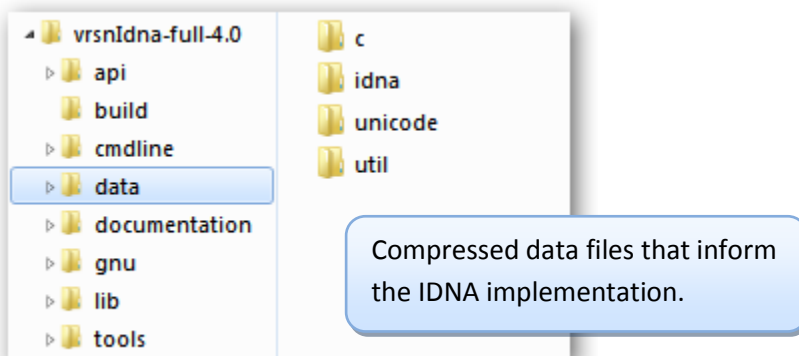
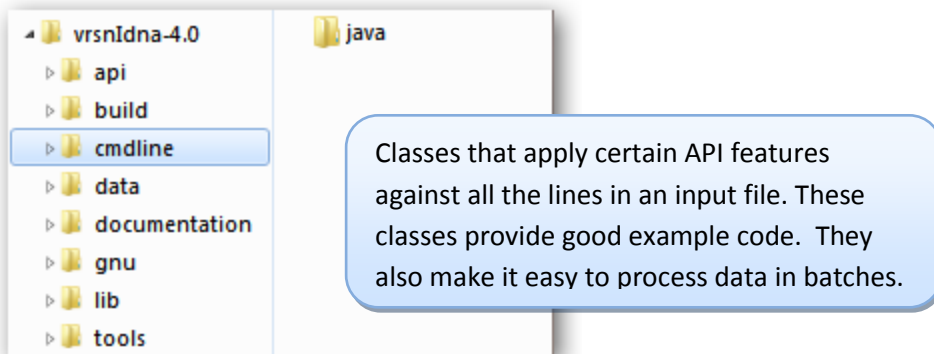
### OVERVIEW

The VeriSign IDN SDK is distributed in a compressed zip file format. This is a common format, and conventional operating systems provide tools to inspect and decompress these files. To install the IDN SDK, simply decompress the zip file to a convenient location on your hard drive.

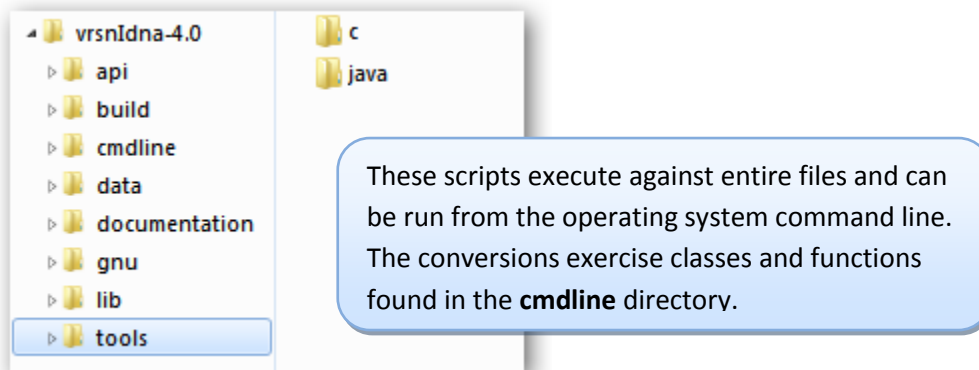
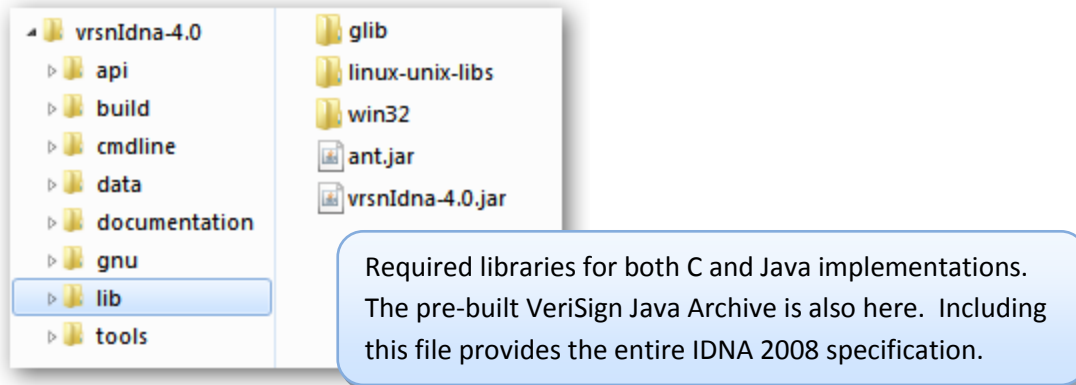
### DIRECTORY STRUCTURE

The following directory structure should appear as a result of the SDK installation process.









## COMPILATION

### OVERVIEW

The VeriSign IDN SDK contains source code for the Java and C programming languages as well as logic for compiling this source code into object files.

### COMPILING THE JAVA IDN SDK

The IDN SDK distribution file contains object files for the Java programming language and includes logic for compiling the Java code. Execution of the Java tools requires only a Java Virtual Machine (JVM), version 1.5 or later, for the target operating system. A suitable JVM is freely available on the Java Web site at <http://www.java.sun.com>.

To build the Java source code into object and executable files, follow these steps:

1. Open a terminal window.
2. Change the directory to the `api/build` directory under the IDNSDK root.
3. Set the `JAVA_HOME` environment variable to point to the directory where the JVM is installed.  
**Unix:** Set this variable by executing  
    `set JAVA_HOME=<JVM location>`  
**Windows:** Set this variable by executing  
    `set JAVA_HOME=<JVM location>`
4. Set the `ANT_HOME` environment variable to point to the directory where Apache Ant is installed.  
**Unix:** Set this variable by executing  
    `set ANT_HOME=<ANT location>`  
**Windows:** Set this variable by executing  
    `set ANT_HOME=<ANT location>`
5. Set the `PATH` environment variable to point to the directory where the `JAVA_HOME/bin` and Apache Ant executables are installed.  
**Unix:** Set this variable by executing  
    `set PATH=${JAVA_HOME}/bin:${ANT_HOME}/bin`  
**Windows:** Set this variable by executing

```
set PATH=%JAVA_HOME%/bin;%ANT_HOME%/bin
```

6. Start the build process.

**Unix:** Execute the build.sh shell script

**Windows:** Execute the build.bat batch script

Completion of these instructions will launch a series of steps that prepare and build the Java source code. The result of this build process is the update of the IDN SDK JAR file, which should already exist in the lib directory. Code changes made before the build will be incorporated in this new JAR file. The Java tools in the tools/java directory use the IDN SDK JAR file and will immediately reflect any updates to the JAR file.

## COMPILING THE C IDN SDK

To use the C tools available with the IDN SDK distribution file, users must compile the C source code. The C library supports compilation through the Make utility.

To use the Make utility to build the C source code into object and executable files, follow these steps:

### For Win32:

1. Ensure that cygwin and gcc are installed.
2. Open a cygwin terminal window.
3. Change the directory to the api/c/build directory under the IDNSDK root.
4. Execute `uncompress_data_files.bash`.
5. Download/Copy the following into the respective locations:
  - a. Copy `cygwin1.dll` from `cygwin/bin` of your cygwin installation to the `lib/win32/cygwin-runtime` directory under the IDNSDK root
  - b. Copy `cyggcc_s-1.dll` from `cygwin/bin` of your cygwin installation to the `lib/win32/cygwin-runtime` directory under the IDNSDK root
  - c. Download the `gettext-runtime` package – version 0.18.1.1 from <http://www.gnu.org/download/windows.html> . Unzip the package into the `lib/win32/gettext-runtime` directory under the IDNSDK root.
  - d. Download the `Glib Dev` package – version 2.26.0 from <http://www.gnu.org/download/windows.html> . Unzip the package into the `lib/glib` directory under the IDNSDK root.
  - e. Download the `Glib runtime` package – version 2.26.0 from <http://www.gnu.org/download/windows.html> . Unzip the package into the `lib/win32/glib-runtime` directory under the IDNSDK root.
6. Issue the `make` command – default target builds the IDN SDK in the `api/c/build` directory.

7. Issue `make all_tools` – only if rebuild of tools is required. The `tools/c/win32` contains executables as part of the distribution.

### For Linux/Unix:

1. Open a terminal window for a bash shell, and ensure the user has `sudo` (superuser do) privileges.
2. Change the directory to the `api/c/build` directory under the IDNSDK root.
3. Execute `uncompress_data_files.bash`.
4. Download/Move the following into the respective locations:
  - a. Download the `pkgconfig` tarball from `ftp://ftp.gtk.org/pub/gtk/v2.2/dependencies` into the `lib/linux-unix-libs` directory under the IDNSDK root.
  - b. Download the `gettext-runtime` tarball – version 0.18.1.1 from `http://ftp.gnu.org/pub/gnu/gettext/gettext-0.18.1.1.tar.gz`, into the `lib/linux-unix-libs` directory under the IDNSDK root.
  - c. Download the `Glib` tarball – version 2.26.0 from `http://ftp.gnome.org/pub/gnome/sources/glib/2.26/` into the `lib/linux-unix-libs` directory under the IDNSDK root.
5. Execute `install_glib_dependencies.bash` in the `api/c/build` directory.
6. Upon successful completion of step 5, issue the `make` command – default target builds the IDN SDK.
7. Upon successful completion of step 6, issue `make all_tools` – builds the tools.

Completion of the preceding instructions will launch a series of steps that prepare and build the C source code. After completion, the IDN SDK library will be available as `lib/win32/xcode.dll` on Windows, as `lib/linux/libxcode.so` in Linux, and the C tools will be available in the `tools/c` directory under the IDNSDK root.

The C library supports a number of useful constants, types, and compile configuration switches, which are configured through a single configuration file. This file, `xcode_config.h`, is located in the `api/c/inc` directory. For specific information on these header files, see the C library's `README.txt` file located in the documentation folder of the IDN SDK distribution.

The `xcode.h` file is the only file that must be included to compile the C IDN SDK.

The C implementation of the IDN SDK has been successfully compiled and built on the following platforms:

Operating System	Compiler
Linux 2.6.18-128.1.6.el5 SMP	gcc
Windows XP Professional	cygwin gcc
Windows 7 Enterprise	cygwin gcc

The C implementation of the IDN SDK uses the standard C libraries, and has dependencies on the GLib 2.26.0 utility library, which, as in the instructions above, may be downloaded by following platform-specific links at <http://www.gtk.org/download.html> .

## INPUT FORMAT

### OVERVIEW

The tools in the VeriSign IDN SDK use file-based input. This feature facilitates bulk data processing and simplifies the command-line interface. The tools expect exactly one input sequence per line. The format of each line varies depending on the type of data the tool expects.

There are four possible data types: ASCII, Unicode, UTF-16, and Native. The Unicode, UTF-16, and Native types represent binary data. Because operating systems often have differing methods for interpreting binary data, the IDN SDK tools use a hexadecimal notation to represent code points in these data types. Each single code point is constructed of the characters 0 – 9 and a – f. A code point can optionally be prefixed by the characters 0x or \u. The interpreter will ignore these prefixes. Sequences of code points must be separated by white space. The four input data types are described in the following sections.

### ASCII

The ASCII format is used for interpretation of ACE-encoded data. This format does not require hexadecimal representation. This feature implies the characters in the file are used without interpretation.

The largest possible value is system dependant, but characters in a file generally will not exceed 0xFF (hex notation).

The following example illustrates a single input sequence.

```
www.xn--rsum-bpad.com
```

### UNICODE (UTF-32)

The most recent Unicode specification allows for data points as big as 21 bits. The largest possible Unicode code point is 0x10FFFF (hex notation).

The following example illustrates a single input sequence. Note that the use of prefixes is optional and case is ignored.

```
77 77 77 \Uff61 0x2f99d 9fa5 2e 63 6f 6d
```

### UNICODE (UTF-16)

The UTF-16 data format uses a 16-bit data sequence to represent Unicode code points. Unicode code points greater than 0xFFFF are represented with two 16-bit values. The largest possible UTF-16 code point is 0xFFFF (hex notation).

The following example illustrates a single input sequence. Note that this data sequence is identical to the previous Unicode sequence, except that the wide Unicode code points (0x2f99d) have been represented using a pair of surrogate values (d87e dd9d).

**77 77 77 ff61 d87e dd9d 9fa5 2e 63 6f 6d**

#### NATIVE

The Native data type is used only in the Java API and tools. This data format is used to represent and interpret various binary data types like UTF-8, GB, Big5, and S-JIS. The full list of supported encodings can be found at <http://download.oracle.com/javase/6/docs/technotes/guides/intl/encoding.doc.html>. Native data is interpreted one byte at a time. The largest possible Native code point is 0xFF (hex notation).

The following example illustrates a single input sequence. Note that this data sequence is identical to the previous UTF-16 sequence, except that the wide UTF-16 code points have been represented using the binary UTF-8 encoding.

**77 77 77 ef bd a1 f0 af a6 9d e9 be a5 2e 63 6f 6d**

## OUTPUT FORMAT

### OVERVIEW

The same operational process is used across all VeriSign IDN SDK tools. Upon execution, the program reads parameters and an input file from the command-line arguments. Files are processed line by line. Each line from the input file is a single input sequence. This sequence is interpreted, stored in an appropriate data type, and passed to one or more API calls. If the API call is successful, the output sequence is written directly to STDOUT, one output sequence per line.

If the API call results in an error, the output is written to STDERR in the following format:

```
<input>\tERROR:<error code>\t<description>
                                or
<input>\tFATAL:<error code>\t<description>
                                or
<input>\tMISMATCH\t<round-trip-result>
```

Where

<b>&lt;input&gt;</b>	The exact input sequence from the input file.
<b>\t</b>	A single TAB character (0x09).
<b>&lt;error code&gt;</b>	An integer code uniquely identifying the type of error. Error codes are used across language implementations so the same error should elicit the same error code and error description in both the C and Java implementations.
<b>&lt;description&gt;</b>	A verbose description of the cause of the error.
<b>&lt;round-trip-result&gt;</b>	The data sequence resulting from first encoding the <input> and then decoding the result using the reverse algorithm. In symmetric programs that guarantee a loss-less conversion, the <round-trip-result> should exactly match the <input>.
<b>ERROR</b>	Indicates that an exception was caught on the first leg of a round-trip test, or on a one-way test. When issuing invalid input, these errors are expected.
<b>FATAL</b>	Indicates that an exception was caught during the second leg of a round-trip test.*



**MISMATCH**

A round-trip test is used for symmetric algorithms, which can be reversed without loss of information. These algorithms, like RACE and Punycode, have both encode and decode routines. A round-trip test first encodes the <input>, and then decodes the result of the encoding step. The <round-trip-result> retrieved from the process must match the <input> exactly. If this is not the case, an error has occurred.\*

\*FATAL and MISMATCH errors are unexpected. If an execution of the software yields a FATAL or MISMATCH error, please contact VeriSign Customer Support at [idsdk@verisign-grs.com](mailto:idsdk@verisign-grs.com).

## TOOLS

### OVERVIEW

The C tools reside in the `tools/c` directory. The C programs must be compiled before use. (Please see section 3 on Compilation.) Once compiled, the `tools/c` directory contains a set of executable programs, which can be invoked from the command line.

Scripts to run the Java tools are provided for both the Unix and Windows platforms.

The `tools/java/unix` directory contains a set of csh scripts suitable for most Unix platforms. The `tools/java/win32` directory contains a set of executable batch files. These tools do not require compilation. They may be run directly after downloading and unpacking the IDN SDK.

### CONVENTIONS

Users familiar with Unix usage standards will recognize the following conventions:

<b>[item]</b>	Indicates that the item is optional. It can be omitted.
<b>[item1   item2   item3]</b>	Indicates a group from which zero or one item can appear on the command line.
<b>(item1   item2   item3)</b>	Indicates a group from which exactly one item can appear on the command line.
<b>&lt;item description&gt;</b>	The text inside the angle brackets describes the information required on the command line. Often, a further description can be found below the bracketed description.

### COMMON TOOLS

The following tools are provided for both the Java and C implementations within the IDN SDK. (Please refer to sections 4 and 5 for details about the Input and Output data types.)

#### BIDI

<b>Purpose</b>	The bidirectional (Bidi) tool. This class asserts that the given set of Unicode code points complies with IDNA2008 Bidi rules.
----------------	--

<b>Usage</b>	<code>bidirectional file=&lt;file&gt;</code>
<b>Input type</b>	Unicode
<b>Output type</b>	Error condition if Bidi rules are not met.

---

## IDNA

<b>Purpose</b>	Internationalized Domain Names in Applications. This set of algorithms defines a way to encode and decode Unicode data so that it is compatible with the Domain Name System (DNS).
<b>Usage</b>	<code>idna [-3ars] (--ToASCII   --ToUnicode) file=&lt;file&gt;</code> <code>-3 =&gt; do NOT enforce Std 3 ASCII Rules</code> <code>-c =&gt; do NOT perform IDNA2008 Protocol validation</code> <code>-r =&gt; use RACE for ACE encoding (Punycode by default)</code> <code>-x =&gt; allow eXceptions during ToUnicode</code>
<b>Input type</b>	Unicode for ToASCII; ASCII for ToUnicode
<b>Output type</b>	ASCII for ToASCII; Unicode for ToUnicode

---

## NORMALIZE

<b>Purpose</b>	This tool normalizes the input sequence using the NFKC algorithm.
----------------	---

<b>Usage</b>	<code>normalize [-c] file=&lt;file&gt;</code> <code>-c =&gt; test for conformance</code>
--------------	---

Note: When the -c option is specified, the input is processed as an entry from the Unicode conformance test. Each line should have five fields delimited by a semi-colon (;). The format is c1;c2;c3;c4;c5;

The conformance test is applied as follows:

`c4 == NFKC(c1) == NFKC(c2) == NFKC(c3) == NFKC(c4) == NFKC(c5)`

where c1 is the first field, c2 is the second field, and so on.

A sample line is shown here:

`1E0A;1E0A;0044 0307;1E0A;0044 0307;`

<b>Input type</b>	Unicode
<b>Output type</b>	Unicode

---

## PUNYCODE

<b>Purpose</b>	This algorithm compresses and converts Unicode data into an ASCII-compatible sequence. It was designed for use with IDNA. The IETF does not support any other ACE encoding. The IDNA RFC permits applications to choose whether or not to exclude ASCII characters that are not a letter, digit, or hyphen. If the <code>-3</code> switch is given, these code points are allowed to be encoded by Punycode.
<b>Usage</b>	<pre>punycode [-3] (--encode --decode) file=&lt;file&gt; -3 =&gt; do NOT enforce Std 3 ASCII rules</pre>
<b>Input type</b>	Unicode
<b>Output type</b>	ASCII

---

## UNICODE

<b>Purpose</b>	This tool converts between Unicode data and UTF-16 by using the surrogate arithmetic specified in the UTF-16 RFC. For more information about this RFC, please see <a href="http://www.ietf.org/rfc/rfc2781.txt">http://www.ietf.org/rfc/rfc2781.txt</a> .
<b>Usage</b>	<pre>unicode (--encode --decode) file=&lt;file&gt;</pre>
<b>Input type</b>	UTF-16 for encode; Unicode for decode
<b>Output type</b>	Unicode for encode; UTF-16 for decode

## JAVA-ONLY TOOLS

### OVERVIEW

The following tools are provided only for the Java implementation within the IDN SDK. (Please refer to sections 4 and 5 for details about the Input and Output data types.)

#### BASE32

<b>Purpose</b>	This tool uses only the characters a – z and 2 – 7 to convert a string of binary bytes into an ASCII-compatible sequence. The input sequence is read five bits at a time, and each five bits is converted into one of the 32 allowed characters.
<b>Usage</b>	<code>base32 (--encode --decode) file=&lt;file&gt;</code>
<b>Input type</b>	Native for encode; ASCII for decode
<b>Output type</b>	ASCII for encode; Native for decode

#### CONVERT

<b>Purpose</b>	This tool converts input directly between RACE, Punycode, and any Java-supported Native encoding, thereby bypassing intermediate steps. This routine is useful in applications migrating RACE-encoded domains to Punycode. This tool also prepares Natively encoded data for IDN registration. For a complete list of supported Native encodings, please see the <a href="#">Java Encodings Page</a> for details.
<b>Usage</b>	<pre>convert [-3cx] file=&lt;file&gt; &lt;input type&gt; &lt;output types&gt; -3 =&gt; do NOT enforce Std 3 ASCII rules -c =&gt; do NOT perform round-trip check during ToUnicode -x =&gt; allow eXceptions during ToUnicode --list =&gt; output a list of supported encoding types</pre>
<b>Input type</b>	ASCII for RACE and Punycode; Native for all others
<b>Output type</b>	ASCII for RACE and Punycode; Native for all others

#### DCE

<b>Purpose</b>	This tool uses DNS-Compatible Encoding (DCE) to encode binary input. DCE is identical to Base32 encoding except that labels longer than 63 characters are split using an ASCII FULL STOP character.
<b>Usage</b>	<code>dce (--encode --decode) file=&lt;file&gt;</code>

<b>Input type</b>	Native for encode; ASCII for decode
<b>Output type</b>	ASCII for encode; Native for decode

---

#### ENCVARIANTS

<b>Purpose</b>	This tool uses the Native object to generate a list of encoding variants. Input sequences must be ACE encoded. The same ACE algorithm will be used to re-encode and return valid variants.
<b>Usage</b>	<code>encvariants [-r] file=&lt;file&gt; &lt;encoding list&gt;</code> <code>-r =&gt; use RACE for ACE encoding (Punycode by default)</code>
<b>Input type</b>	ASCII
<b>Output type</b>	ASCII

---

#### HEX

<b>Purpose</b>	The hex tool converts raw binary data into a hexadecimal format that is suitable for use in the other conversion tools. Hexadecimal data can also be decoded and written to a file in raw format.
<b>Usage</b>	<code>hex (--encode --decode) file=&lt;file&gt;</code>
<b>Input type</b>	Raw binary for encode; hexadecimal for decode
<b>Output type</b>	Hexadecimal for encode; raw binary for decode

---

#### NATIVE

<b>Purpose</b>	This tool converts between Native language encodings like Big5, S-JIS, and UTF-8. Input data can be converted between one or more Native encodings in a space-separated list. The full list of supported encodings can be found at <a href="http://download.oracle.com/javase/1.4.2/docs/guide/intl/encoding.doc.html">http://download.oracle.com/javase/1.4.2/docs/guide/intl/encoding.doc.html</a> .
<b>Usage</b>	<code>native (--encode --decode) file=&lt;file&gt; &lt;encoding list&gt;</code>
<b>Input type</b>	UTF-16 for encode; Native for decode
<b>Output type</b>	Native for encode; UTF-16 for decode

---

#### RANDOMDATA

<b>Purpose</b>	This routine generates random data in any of several formats.
----------------	---

<b>Usage</b>	<pre>randomdata (&lt;format&gt; tcsc=&lt;tcsclist&gt;) [&lt;type&gt;] [&lt;lines&gt;]</pre> <p>&lt;format&gt; one of the following</p> <ul style="list-style-type: none"> <li>&lt;native&gt; any of several supported Native formats such as UTF-8 or Big5. 8-bit data in hexadecimal representation</li> <li>utf16 16-bit data in hexadecimal representation</li> <li>Unicode 21-bit data in hexadecimal representation</li> <li>Ace Base32 sequence with no prefix</li> <li>Race Base32 sequence with bq-- prefix</li> <li>Punycode Base32 sequence with xn-prefix</li> </ul> <p>&lt;tcsclist&gt; a comma-separated list of the following</p> <ul style="list-style-type: none"> <li>tc Traditional Chinese (TC) code points</li> <li>sc Simplified Chinese (SC) code points</li> <li>ascii ASCII code points</li> <li>none code points that are not TC, SC, or ASCII</li> </ul> <p>&lt;type&gt; one of the following</p> <ul style="list-style-type: none"> <li>label data does not include delimiters</li> <li>domain data can include delimiters</li> </ul>
<b>Input type</b>	None
<b>Output type</b>	Various

---

## RACE

<b>Purpose</b>	This tool compresses and converts Unicode data into an ASCII-compatible.
<b>Usage</b>	<pre>race [-3] (--encode --decode) file=&lt;file&gt;</pre> <p>-3 =&gt; do NOT enforce Std 3 ASCII rules</p>
<b>Input type</b>	Unicode
<b>Output type</b>	ASCII

## CONVERSION EXAMPLES

### OVERVIEW

The following examples illustrate how to use various tools to convert data from one format to another.

### EXAMPLE 1

Generate three rows of random Unicode data with a single label. Place output into a file called "x.u"

#### Command

```
randomdata unicode 3 label > x.u
```

#### File Output

```
f9cd 8af1  
2aee0 19c2 5a2b3 2524 d3c2 a34d  
7446 7160 a6c4 5371 1c9a
```

### EXAMPLE 2

Decode the rows from Unicode to UTF-16. Place output into a file called "x.16"

#### Command

```
unicode --decode file=x.u > x.16
```

#### File Output

```
f9cd 8af1  
d86b dee0 19c2 d928 deb3 2524 d3c2 a34d  
7446 7160 a6c4 5371 1c9a
```

### EXAMPLE 3

Encode the UTF-16 labels into UTF-8. Place output into a file called "x.8."

#### Command

```
native --encode file=x.16 UTF8 > x.8
```



**File Output**

```
ef a7 8d e8 ab b1
f0 aa bb a0 e1 a7 82 f1 9a 8a b3 e2 94 a4 ed 8f 82 ea 8d 8d
e7 91 86 e7 85 a0 ea 9b 84 e5 8d b1 e1 b2 9a
```

**EXAMPLE 4**

Use IDNA to encode the original Unicode labels into RACE. Store the result in “x.r” (making sure to capture the error output as well).

**Command**

```
idna -r --toAscii file=x.u > x.r
```

**File Output**

```
bq--3b2vtcxr
2aee0 19c2 5a2b3 2524 d3c2 a34d ERROR:1300 Prohibited 2aee0
7446 7160 a6c4 5371 1c9a ERROR:1300 Prohibited a6c4
```

**EXAMPLE 5**

Convert the UTF-8 encoded labels directly to Punycode. Store the result in “x.p.”

**Command**

```
convert UTF8 punycode file=x.8 > x.p
```

**File Output**

```
xn--v62a169s
xn--0jf514clo6hyrubwp87a6yr4a
xn--t4f715u20s1scv05g
```

## VERISIGN IDN SDK DISCUSSION LIST

### OVERVIEW

VeriSign maintains a discussion list devoted to the VeriSign IDN SDK. This list provides an ideal forum for implementers to exchange ideas and insights about the software. VeriSign engineers maintain a presence on the list to answer questions and offer suggestions.

Like some newer software, the IDN SDK attempts to quantify error conditions so that users can more easily report errors to developers. If the software generates a FATAL or MISMATCH error during any execution, users are encouraged to share the error on the IDN SDK discussion list so that VeriSign can correct the error condition as quickly as possible.

The address of the discussion list is [idsn-sdk@verisign-grs.com](mailto:idsn-sdk@verisign-grs.com). To become a member of the IDN SDK discussion list, simply send an email to [majordomo@verisign-grs.net](mailto:majordomo@verisign-grs.net) with the following text in the message body:

```
subscribe idn-sdk <your e-mail address>
```

The discussion list will send an email to which you must reply. This reply will activate your membership. Once you have completed these steps, you will begin to receive the discussion list correspondence.

## APPENDICES

### ERROR CODES

For details on error codes, please see sections 4.1 and 4.2 of the VeriSign IDN SDK Programmer's Guide.

### REFERENCES

IDNA2008 Protocol	<a href="ftp://ftp.rfc-editor.org/in-notes/rfc5891.txt">ftp://ftp.rfc-editor.org/in-notes/rfc5891.txt</a>
IDNA2008 Tables	<a href="ftp://ftp.rfc-editor.org/in-notes/rfc5892.txt">ftp://ftp.rfc-editor.org/in-notes/rfc5892.txt</a>
IDNA2008 Bidi	<a href="ftp://ftp.rfc-editor.org/in-notes/rfc5893.txt">ftp://ftp.rfc-editor.org/in-notes/rfc5893.txt</a>
Punycode	<a href="ftp://ftp.rfc-editor.org/in-notes/rfc3492.txt">ftp://ftp.rfc-editor.org/in-notes/rfc3492.txt</a>
RACE	<a href="http://tools.ietf.org/html/draft-ietf-idn-race-03">http://tools.ietf.org/html/draft-ietf-idn-race-03</a>
UTF-16	<a href="http://www.ietf.org/rfc/rfc2781.txt">http://www.ietf.org/rfc/rfc2781.txt</a>
Encodings	<a href="http://download.oracle.com/javase/6/docs/technotes/guides/intl/encoding.doc.html">http://download.oracle.com/javase/6/docs/technotes/guides/intl/encoding.doc.html</a>