

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

Document Version 2.0  
October 29<sup>th</sup>, 2010

**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Copyright © 2010 VeriSign, Inc., as an unpublished work.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
Java IDN SDK .....	5
Compiling the Java IDN SDK.....	5
Client Applications .....	6
Locating.....	6
Including.....	6
Activating .....	7
Classes Available in the Java IDN SDK.....	7
Java Sample Code .....	7
base32 .....	8
bidi .....	9
IDNA .....	9
Native .....	10
Normalize .....	12
Punycode.....	13
RACE .....	14
Unicode .....	15
C IDN SDK .....	17
Compiling the C IDN SDK .....	17
Client Applications .....	19
Environment.....	19
Locating.....	20
Including.....	21
Activating .....	21
Functions.....	22
Primary Entry Points .....	22
Auxiliary Entry Points .....	22
C Sample Code .....	23
BidiFilter .....	23
DomainToAscii .....	23
DomainToUnicode .....	24
Normalize .....	25
Punycode.....	25
ToASCII .....	26
ToUnicode .....	27

UtfConvert.....	27
Appendices.....	28
Java Error Codes .....	28
C Error Codes .....	33
Common Error Codes.....	33
Feature Specific Error Codes.....	33
Extending the Java IDN SDK.....	34
Writing New Objects.....	35
Writing New Tools.....	35
Testing with Random Data.....	35
Adding New Error Codes.....	36
Using Data Files.....	36
References .....	37

## INTRODUCTION

### PURPOSE

This guide provides details about the VeriSign Internationalized Domain Names Software Development Kit (IDN SDK) application programming interface (API). It also includes sample code. Please read the VeriSign IDN SDK User's Guide before proceeding with the Programmer's Guide. The User's Guide provides an overview of the SDK and includes details about installation and directory structure that are not covered in this document.

## JAVA IDN SDK

### 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.

## CLIENT APPLICATIONS

Some users with client applications in development may wish to leverage one or more functions from the IDN SDK. To accomplish this, developers must include the SDK functionality within their own product.

To include SDK functionality within their own product, developers must perform three tasks:

- Locating – The client application must be able to locate the IDN SDK.
- Including – The client routine must be able to load certain pieces of the SDK.
- Activating – An SDK function call or calls must be embedded within the client.

### LOCATING

In Java, locating is accomplished by including the IDN SDK JAR file on the system Class path. Your system JVM will use a system variable called CLASSPATH to look for packages of functionality. If this variable includes the location of the IDN SDK JAR file, then the JVM will find the appropriate functions during compilation.

### INCLUDING

Including is accomplished with the Import statement. The Import statement tells Java which packages are required to compile and interpret the current class. For instance, consider a Java object that will look through a string to determine whether all the characters are in the ASCII range. Developers would place the following line at the top of the Java source file:

```
import com.vgrs.xcode.common.Utf16;
```

This line tells Java that some routines in the following class may use methods from a UTF-16 object, which is located in the `com.vgrs.xcode.common` package. This statement does not tell Java which file the UTF-16 object resides in. The Java compiler will look through all the items on the CLASSPATH list until it finds the object it is looking for.

#### ACTIVATING

Activating is the use of a method or attribute from a particular Java object, as shown in the following example.

```
String myString = "Hello World \u263A";
if (Utf16.isAscii(myString.toCharArray())) {
    System.out.println("The String is all ASCII characters.");
} else {
    System.out.println("The String contains non-ASCII characters.");
}
```

#### CLASSES AVAILABLE IN THE JAVA IDN SDK

Please refer to [vrsnIdna-4.0/documentation/javadoc/index.html](http://vrsnIdna-4.0/documentation/javadoc/index.html) for a complete list of the Java classes available in the IDN SDK.

#### JAVA SAMPLE CODE

The following sections contain sample code that can be compiled and executed. See the User's Guide for a more complete description of each object's purpose.

The Javadoc, located in the doc section of the IDN SDK, provides a more complete reference for programmers familiar with standard Java documentation.

## BASE32

This class provides algorithms to encode/decode data to/from Base32.

```
import com.vgrs.xcode.common.Base32;
public class Base32Sample {

    public static void main ( String[] args ) {
        try {
            byte[] input = {(byte) 0xd2, (byte) 0x76, (byte) 0x85, (byte)
0x2e};
            char[] output = Base32.encode( input );
            byte[] roundtrip = Base32.decode( output );
            System.out.println( "input = " + toString( input ) );
            System.out.println( "output = " + new String( output ) );
            System.out.println( "roundtrip = " + toString( roundtrip ) );
        }
        catch ( Exception eX ) {
            eX.printStackTrace();
        }
    }

    private static String toString ( byte[] input ) {
        if ( input == null )return null;
        if ( input.length == 0 ) return "";
        String output = Integer.toString( ((char) input[ 0 ]) & 0xff, 16
);
        for ( int i = 1; i < input.length; i++ ) {
            output += " " + Integer.toString( ((char) input[ i ]) & 0xff, 16
);
        }
        return output;
    }
}
```



## BIDI

Bidi rules apply to IDNs that contain right-to-left characters. The Bidi class has logic to enforce the IDNA2008 Bidi rules. The following example shows code points that belong to the Arabic script. Arabic letters are written from right to left.

```
import com.vgrs.xcode.idna.Bidi;
public class BidiSample {

    public static void main ( String[] args ) {
        try {
            int[] input = {0x621, 0x622, 0x623};
            //
            // test if the input will pass through the
            // Bidi algorithm without errors
            //
            Bidi.assertCompliance( input );
            System.out.println( "Input passed Bidi without errors" );
        }
        catch ( Exception eX ) {
            eX.printStackTrace();
        }
    }
}
```

## IDNA

This class implements basic rules of the Internationalized Domain Names in Applications (IDNA) RFC.

```
import com.vgrs.xcode.idna.Idna;
import com.vgrs.xcode.idna.Punycode;
```

```
public class IdnaSample {

    public static void main ( String[] args ) {
        try {
            Idna idna = new Idna( new Punycode(), true, true );
            char[] input = "xn--hvm3583a.com".toCharArray();
            int[] output = idna.domainToUnicode( input );
            char[] roundtrip = idna.domainToAscii( output );
            System.out.println( "input = " + new String( input ) );
            System.out.println( "output = " + toHexString( output ) );
            System.out.println( "roundtrip = " + new String( roundtrip ) );
        }
        catch ( Exception eX ) {
            eX.printStackTrace();
        }
    }

    private static String toHexString ( int[] input ) {
        if ( input == null ) return null;
        if ( input.length == 0 ) return "";
        String output = Integer.toString( input[ 0 ], 16 );
        for ( int i = 1; i < input.length; i++ ) {
            output += " " + Integer.toString( input[ i ], 16 );
        }
        return output;
    }
}
```

---

## NATIVE

This class provides algorithms to convert a character sequence between UTF-16 and other Native encodings.

```
import java.util.Map;

import com.vgrs.xcode.common.Native;

public class NativeSample {

    public static void main ( String[] args ) {

        try {
            char[] inputChars = {0x1bb4, 0xc89f, 0x9, 0x90c, 0x12cc};

            String input = new String( inputChars );

            //
            // try to encode in a single encoding
            //
            String encoding = "UTF8";
            String output = Native.encode( input, encoding );
            System.out.println( "input = " + toHexString( input ) );
            System.out.println( "encoding = " + encoding );
            System.out.println( "output = " + toHexString( output ) );

            //
            // try to encode in a list of encodings
            //
            String encodings[] = {"UTF8", "UTF-16"};
            Map<String, String> outputs = Native.encode( input, encodings );
            System.out.println();
            System.out.println( "input = " + toHexString( input ) );
            for ( int i = 0; i < encodings.length; i++ ) {
                System.out.println( "encodings[" + i + "] = " + encodings[i]);
            }
            for ( String key : outputs.keySet() ) {
                output = outputs.get( key );
            }
        }
    }
}
```

```
        System.out.println("output:"+key+" = "+toHexString(output));
    }
}
catch ( Exception eX ) {
    eX.printStackTrace();
}
}

private static String toHexString ( String input ) {
    if ( input == null ) return null;
    if ( input.length() == 0 ) return "";
    char[] inputc = input.toCharArray();
    String output = Integer.toString( inputc[ 0 ], 16 );
    for ( int i = 1; i < inputc.length; i++ ) {
        output += " " + Integer.toString( inputc[ i ], 16 );
    }
    return output;
}
}
```

---

## NORMALIZE

This class provides an algorithm to normalize a domain.

```
import com.vgrs.xcode.idna.Normalize;

public class NormalizeSample {

    public static void main ( String[] args ) {
        try {
            int[] input = {0x2000, 0x2001, 0x402};
            int[] output = Normalize.execute( input );
            System.out.println( "input = " + toHexString( input ) );
            System.out.println( "output = " + toHexString( output ) );
        }
    }
}
```

```
    }  
    catch ( Exception eX ) {  
        eX.printStackTrace();  
    }  
}  
  
private static String toHexString ( int[] input ) {  
    if ( input == null ) return null;  
    if ( input.length == 0 ) return "";  
    String output = Integer.toString( input[ 0 ], 16 );  
    for ( int i = 1; i < input.length; i++ ) {  
        output += " " + Integer.toString( input[ i ], 16 );  
    }  
    return output;  
}  
}
```

---

## PUNYCODE

This class implements the Punycode ASCII-Compatible Encoding (ACE) algorithm.

The Punycode algorithm transforms a Unicode string into a sequence of characters (e.g., ASCII letters, digits, and hyphens) that are allowed in hostname labels.

The following code is almost an exact replica of the sample implementation in C provided in RFC 3492.

```
import com.vgrs.xcode.idna.Punycode;  
  
public class PunycodeSample {  
  
    public static void main ( String[] args ) {  
        try {  
            Punycode punycode = new Punycode();  

```

```
int[] input = {0x3980,0x51f7,0x4e7b7,0x5130,0xb817,0xdcaef};
char[] output = punycode.encode( input );
int[] roundtrip = punycode.decode( output );
System.out.println( "input = " + toHexString( input ) );
System.out.println( "output = " + new String( output ) );
System.out.println( "roundtrip = " + toHexString( roundtrip ) );
}
catch ( Exception eX ) {
    eX.printStackTrace();
}
}

private static String toHexString ( int[] input ) {
    if ( input == null )return null;
    if ( input.length == 0 ) return "";
    String output = Integer.toString( input[ 0 ], 16 );
    for ( int i = 1; i < input.length; i++ ) {
        output += " " + Integer.toString( input[ i ], 16 );
    }
    return output;
}
}
```

---

## RACE

This class implements the Row-Based ASCII-Compatible Encoding (RACE) algorithm.

The RACE algorithm is similar to Punycode, but it is less efficient. It is included in the IDN SDK to enable backward compatibility.

```
import com.vgrs.xcode.idna.Race;

public class RaceSample {
```

```
public static void main(String[] args) {
    try {
        Race race = new Race();
        int[] input = {0x3980,0x51f7,0x4e7b7,0x5130,0xb817,0xdcaef};
        char[] output = race.encode(input);
        int[] roundtrip = race.decode(output);
        System.out.println("input = " + toHexString(input));
        System.out.println("output = " + new String(output));
        System.out.println("roundtrip = " + toHexString(roundtrip));
    } catch (Exception eX) {
        eX.printStackTrace();
    }
}

private static String toHexString(int[] input) {
    if (input == null) return null;
    if (input.length == 0) return "";
    String output = Integer.toString(input[0], 16);
    for (int i = 1; i < input.length; i++) {
        output += " " + Integer.toString(input[i], 16);
    }
    return output;
}
}
```

---

## UNICODE

This class provides algorithms to encode/decode a UTF-16 character sequence to/from Unicode.

```
import com.vgrs.xcode.common.Unicode;
```

```
public class UnicodeSample {

    public static void main ( String[] args ) {
        try {
            char[] input = {0xda5a, 0xddf4, 0xbd20, 0xd40a, 0x7c02, 0x573a};
            int[] output = Unicode.encode( input );
            char[] roundtrip = Unicode.decode( output );
            System.out.println( "input = " + toHexString( input ) );
            System.out.println( "output = " + toHexString( output ) );
            System.out.println( "roundtrip = " + toHexString( roundtrip ) );
        }
        catch ( Exception eX ) {
            eX.printStackTrace();
        }
    }

    private static String toHexString ( char[] input ) {
        if ( input == null ) return null;
        if ( input.length == 0 ) return "";
        String output = Integer.toString( input[ 0 ], 16 );
        for ( int i = 1; i < input.length; i++ ) {
            output += " " + Integer.toString( input[ i ], 16 );
        }
        return output;
    }

    private static String toHexString ( int[] input ) {
        if ( input == null ) return null;
        if ( input.length == 0 ) return "";
        String output = Integer.toString( input[ 0 ], 16 );
        for ( int i = 1; i < input.length; i++ ) {
            output += " " + Integer.toString( input[ i ], 16 );
        }
    }
}
```



```
    return output;
}

}
```

## C IDN SDK

### COMPILING THE C IDN SDK

The VeriSign IDN SDK contains source code for the C and Java programming languages as well as logic for compiling this source code into object files. To use the C tools available with the 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.gtk.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.gtk.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.gtk.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.

Successful completion of these instructions will launch a series of steps that prepare and build the C source code. Once the C source code is built, 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 `api/c/docs`.

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>.

The Linux/Unix GLib Tarball and its associated dependencies are included under the `api/lib/linux-unix-libs` directory of the distribution file.

The `install_glib_dependencies.bash` script in step 5 of the installation procedure for Unix/Linux platforms unpacks and installs the GLib libraries from the Tarball.

## CLIENT APPLICATIONS

Some users with client applications in development may wish to leverage one or more functions from the IDN SDK. To do so, developers must include the SDK functionality within their own product. Doing so involves the following integration tasks:

- Environment – Unicode data files loaded by the SDK must be located through an environment variable. In Windows environments, the PATH environment variable would need to be modified to include the directories for Runtime libraries for cygwin, Glib, and gettext
- Locating – The client application must be able to locate the IDN SDK.
- Including – The client routine must be able to load certain pieces of the SDK.
- Activating – An SDK function call or calls must be embedded within the client.

## ENVIRONMENT

The `UNICODE_DATA_HOME` environment variable is used to locate Unicode data files loaded by the SDK. The variable should point to the IDNSDK root.

**On Windows:** At runtime, the PATH environment variable needs to include the directory lib\win32, lib\win32\cygwin-runtime, lib\win32\gettext-runtime\bin and lib\win32\glib-runtime\bin under the IDN SDK root.

**On Linux:** At runtime, the LD\_LIBRARY\_PATH environment variable needs to include the directory lib/linux directory under the IDN SDK root, in addition to the library paths of the Glib installation.

**On other platforms:** Refer to the runtime environment documentation for setting runtime library paths.

---

## LOCATING

In C, locating is accomplished by linking to the shared or static object file during the build process of the client application.

Win32 developers should link to either the static or dynamic library through their project settings.

**Note:** The xcode.dll dynamic library should not be used as a shared system library. Applications that use xcode.dll should install an application-local copy of the dll and should not install the library into the Windows system directory.

GNU Make uses the “-L” and “-l” switches to indicate the location of libraries at compile time. It uses the “-R” switch to indicate shared object location at runtime. Consult the GNU Make documentation for details.

The following example is a Makefile for an imaginary project called alpha, which uses the IDN SDK.

```
#
# Sample Makefile for a project named "alpha" that uses libxcode.so
#

CC = gcc
PROJECT_ROOT = /dev/projects/alpha
XCODE_INC = $(PROJECT_ROOT)/../xcode/inc
XCODE_LIB = $(PROJECT_ROOT)/../xcode/lib
```

```
SRCDIR = src
INCDIR = inc
INC_PATH = -I$(XCODE_INC) -I$(INCDIR)
LIB_PATH = -L$(XCODE_LIB)
CFLAGS = $(INC_PATH)
LFLAGS = $(LIB_PATH) -lxcode
RFLAGS = -Wl,-R$(XCODE_LIB)
DEBUG = -DDEBUG

SRCS = $(shell ls $(SRCDIR)/*.c)
PROG = alpha

clean:
    rm -rf $(PROG) $(PROG).log core

build:
    $(CC) $(CFLAGS) $(LFLAGS) $(RFLAGS) $(SRCS) -o $(PROG)

run:
    $(PROG) 2>&1 | tee $(PROG).log

all:    clean build run
```

---

## INCLUDING

Including is accomplished with the Include directive. This directive points the C compiler toward a header file that gives a general description of the SDK functions used in the current file. An example Include statement looks like this:

```
#include <xcode.h>
```

---

## ACTIVATING

Activating is the use of an SDK function from within the client C code, as shown in the following example.

```
int EncodeLabel(const UTF16CHAR * puzInputStr, int iInputLen){

    char szResult[1024];
    int iResultLength = 1024;
    int res = ToASCII(puzInputStr,iInputLen,szResult,&iResultLength);

    if ( res != XCODE_SUCCESS ) {
        // Error
    }

    return res;
}
```

## FUNCTIONS

The following functions are included in the C IDN SDK.

PRIMARY ENTRY POINTS	
Function	Description
Xcode_ToASCII () & Xcode_ToUnicode ()	IDNA routines for encoding and decoding domain labels.
Xcode_DomainToUnicode () & Xcode_DomainToASCII ()	IDNA routines for splitting Internet domains and processing each label within these domains.
Xcode_convertUTF16To32Bit()	Expands 16-bit UTF-16 string data to 32-bit data.
Xcode_convert32BitToUTF16()	Encodes 32-bit data into UTF-16.

AUXILIARY ENTRY POINTS	
Function	Description
Xcode_normalizeString() Xcode_prohibitString() Xcode_bidifilterString()	Enables direct access to Punycode encode/decode routines.

Xcode_puny_encodeString() & Xcode_puny_decodeString()	Enables direct access to RACE decode routines.
Xcode_race_decodeString()	Enables backward compatibility decoding of RACE-encoded domain labels.

## C SAMPLE CODE

The following sections contain sample code that can be compiled and executed. See the User's Guide for a more detailed description of each function's purpose.

### BIDIFILTER

```
#include "xcode.h"

void testBidiFilter( void ){

    int res;

    DWORD dwInput[] = { 0x1d56f, 0x1e22, 0x3a5 };
    int iInputSize = 3;
    res = Xcode_bidifilterString( dwInput, iInputSize );

    if ( res != XCODE_SUCCESS )    {
        /* Error */
    }

}
```

### DOMAINTOASCII

```
#include "xcode.h"

void testDomainToASCII( void ){

    int res;
```

```
UTF16CHAR uInput[] = { 0x0077, 0x0077, 0x0077, 0x002E,  
0x0066, 0x00FC, 0x006E, 0x0066, 0x0064, 0x3002, 0x006E, 0x0065,  
0x0074 };  
  
UCHAR8 szOutput[1204];  
int iInputSize = 13;  
int iOutputSize = sizeof(szOutput);  
res = DomainToASCII( uInput, iInputSize, szOutput, &iOutputSize );  
  
if ( res != XCODE_SUCCESS )    {  
    /* Error */  
}  
  
}
```

---

#### DOMAINTOUNICODE

```
#include "xcode.h"  
  
void testDomainToUnicode( void ){  
  
    int res;  
    UTF16CHAR uOutput[1024];  
    char * szIn = "www.xn--weingut-schnberger-n3b.net";  
    int iInputSize = strlen(szIn);  
    int iOutputSize = sizeof(uOutput);  
    res = DomainToUnicode( szIn, iInputSize, uOutput, &iOutputSize );  
    if ( res != XCODE_SUCCESS )    {  
        /* Error */  
    }  
  
}
```



**NORMALIZE**

```
#include "xcode.h"

void testNormalize( void ){

    int res;
    DWORD dwOutput[1024];
    DWORD dwInput[] = { 0x1d56f, 0x1e22, 0x3a5 };
    int iInputSize = 3;
    int iOutputSize = sizeof(dwOutput);
    res = Xcode_normalizeString( dwInput, iInputSize, dwOutput,
    &iOutputSize );

    if ( res != XCODE_SUCCESS )    {
        /* Error */
    }

}
```

**PUNYCODE**

```
#include "xcode.h"

void testPunycode( void ){

    int res;
    UCHAR8 szOutput[1024];
    DWORD dwInput[] = { 0x1d56f, 0x1e22, 0x3a5 };
    UTF16CHAR uOutput[1024];
    int iInputSize = 3;
    int iOutputSize = sizeof(szOutput);
    res = Xcode_puny_encodeString( dwInput, iInputSize, szOutput,
    &iOutputSize );

    if ( res != XCODE_SUCCESS )    {
```

```
    /* Error */
}

iInputSize = iOutputSize;
iOutputSize = sizeof(uOutput);
res = Xcode_puny_decodeString( szOutput, iInputSize, uOutput,
&iOutputSize );

if ( res != XCODE_SUCCESS )    {
    /* Error */
}

}
```

---

## TOASCII

```
#include "xcode.h"

void testToASCII( void ){
    int res;
    UTF16CHAR uInput[] = { 0x0070, 0x00E4, 0x00E4, 0x006F, 0x006D,
0x0061 };

    UCHAR8 szOutput[1204];
    int iInputSize = 6;
    int iOutputSize = sizeof(szOutput);
    res = ToASCII( uInput, iInputSize, szOutput, &iOutputSize );

    if ( res != XCODE_SUCCESS )    {
        /* Error */
    }

}
```

---

**TOUNICODE**

```
#include "xcode.h"

void testToUnicode( void ){

    int res;
    UTF16CHAR uOutput[1024];
    char * szIn = "xn--weingut-schnberger-n3b";
    int iInputSize = strlen(szIn);
    int iOutputSize = sizeof(uOutput);
    res = IDNToUnicode( szIn, iInputSize, uOutput, &iOutputSize );

    if ( res != XCODE_SUCCESS )    {
        /* Error */
    }

}
```

---

**UTFCONVERT**

```
#include "xcode.h"

void testUTFConvert(){
    int i;
    DWORD dwInput[5];
    UTF16CHAR uResult[256];

    DWORD dwResult[10];
    int iuResultLength = sizeof(uResult);
    int idwResultLength = sizeof(dwResult);

    for ( i = 0x90000; i <= 0x10FFFF; i = i + 4 )    {
        dwInput[0] = i;
    }
}
```

```
dwInput[1] = i+1;
dwInput[2] = i+2;
dwInput[3] = i+3;
Xcode_convert32BitToUTF16(dwInput, 4, uResult, &iuResultLength);
Xcode_convertUTF16To32Bit( uResult, iuResultLength, dwResult,
&i dwResultLength );

if ( memcmp( dwInput, dwResult, 4 ) != 0 )      {
    /* Error */
}
}

}
```

## APPENDICES

### JAVA ERROR CODES

When the Java logic encounters an error scenario, an `XcodeException` occurs. Each exception has exactly one associated error code that describes the error scenario. These error codes are enumerated in the **ErrorCodes.txt** file under the VeriSign IDN SDK **data** directory.

The `com.vgrs.xcode.util.XcodeError` class is generated from the **ErrorCodes.txt** file. For each error code, the following methods are generated and stored in the **XcodeError** class:

**static public XcodeException ErrorCodeName() {...}**

This method throws an `XcodeException` with a specific integer code value.

**static public XcodeException ErrorCodeName(String msg) {...}**

This method throws an `XcodeException` with a specific integer code value and appends the input **msg** variable to the existing message associated with the `XcodeException`.

**static public boolean is\_ErrorCodeName(XcodeException x) {...}**

This method returns **true** if the input `XcodeException` has a certain error code; else, it returns **false**.

The following table lists all the error codes in the **ErrorCodes.txt** file.

0: Success		
0	SUCCESS	Successful execution
# 1 – 99: Common Errors		
1	INVALID_ARGUMENT	Invalid argument.
2	EMPTY_ARGUMENT	Empty argument.
3	NULL_ARGUMENT	Null argument.
4	FILE_IO	File input/output failure.
5	INVALID_FILE_FORMAT	Invalid file format.
6	UNSUPPORTED_ENCODING	Unsupported encoding.
7	IDNSDK_INITIALIZATION_ERROR	IDNSDK Initialization Error.
# 100 – 199: Hex Errors		
100	HEX_DECODE_INVALID_FORMAT	One or more characters does not represent a hex value.
101	HEX_DECODE_ONE_BYTE_EXCEEDED	Value of input characters exceeds 0xff.
102	HEX_DECODE_TWO_BYTES_EXCEEDED	Value of input characters exceeds 0xffff.
103	HEX_DECODE_FOUR_BYTES_EXCEEDED	Value of input characters exceeds 0xffffffff.
# 200 – 299: ACE Errors		
200	ACE_ENCODE_NOT_STD3ASCII	Input does not meet STD3 rules for domain name format.
201	ACE_ENCODE_INVALID_OUTPUT_LENGTH	Resulting ACE sequence is too long or too short.
202	ACE_ENCODE_VALID_PREFIX	The input sequence already has an ACE prefix.
203	ACE_DECODE_NOT_STD3ASCII	Output does not meet STD3 rules for domain name format.
204	ACE_ENCODE_PREFIX_FOUND	Input begins with a valid prefix.
# 300 – 399: RACE Errors		
300	RACE_ENCODE_BAD_SURROGATE_USE	Surrogates should be ordered pairs of high-low during RACE

		encoding.
301	RACE_ENCODE_DOUBLE_ESCAPE_PRESENT	The 0x0099 code point is not allowed during RACE encoding.
302	RACE_ENCODE_COMPRESSION_OVERFLOW	The compressed input length exceeds expected octets during RACE encode.
303	RACE_ENCODE_INTERNAL_DELIMITER_PRESENT	Input contains a delimiter.
304	RACE_DECODE_ODD_OCTET_COUNT	Compression indicates an odd number of compressed octets.
305	RACE_DECODE_BAD_SURROGATE_DECOMPRESS	Compression indicates a stream of identical surrogates.
306	RACE_DECODE_IMPROPER_NULL_COMPRESSION	Sequence could have been compressed but was not.
307	RACE_DECODE_INTERNAL_DELIMITER_FOUND	Found a delimiter while decoding a label.
308	RACE_DECODE_DOUBLE_ESCAPE_FOUND	The 0x0099 code point was found during RACE decoding.
309	RACE_DECODE_UNNEEDED_ESCAPE_PRESENT	Found a double f escape character when u1 is zero.
310	RACE_DECODE_TRAILING_ESCAPE_PRESENT	Found a double f escape character at the end of a sequence.
311	RACE_DECODE_NO_UNESCAPED_OCTETS	The u1 character is non-zero, but all octets are escaped.
312	RACE_DECODE_NO_INVALID_DNS_CHARACTERS	Sequence should not have been encoded.
313	RACE_DECODE_DECOMPRESSION_OVERFLOW	Decompressed sequence exceeds size limitations.
314	RACE_DECODE_5BIT_UNDERFLOW	Too few pentets to create whole-number octets.
315	RACE_DECODE_5BIT_OVERFLOW	Too many pentets to create whole-number octets.
# 400 – 499: Punycode Errors		
400	PUNYCODE_OVERFLOW	The code point exceeded the maximum value allowed.
401	PUNYCODE_BAD_OUTPUT	Bad output was encountered while trying to decode the string.
402	PUNYCODE_BIG_OUTPUT	The output length exceeds expected characters.
403	PUNYCODE_DECODE_DNS_COMPATIBLE	Invalid encoding contains no international data.
404	PUNYCODE_DECODE_INTERNAL_DELIMITER_FOUND	Found a delimiter while decoding a label.
# 500 – 599: Charmap Errors		
500	CHARMAP_OVERFLOW	The output length exceeds expected characters during

		character mapping.
501	CHARMAP_LABEL_ELIMINATION	All input characters were mapped out during character mapping.
# 600 – 699: Normalize Errors		
600	NORMALIZE_BAD_CANONICALCLASS_ERROR	Bad canonical class.
601	NORMALIZE_BAD_COMPATTAG_ERROR	Bad compatibility tag.
602	NORMALIZE_BAD_DECOMPSEQUENCE_ERROR	Bad decomposition sequence.
603	NORMALIZE_NULL_CHARACTER_PRESENT	Null character.
604	NORMALIZE_CANONICAL_LOOKUP_ERROR	Error looking up canonical class.
605	NORMALIZE_NOT_IN_NFC_FORM	Not in NFC normalized form.
606	NORMALIZE_INVALID_CHARACTER	Found character(s) that cannot ever occur in NFC normalized form.
# 700 – 799: Prohibit Errors		
700	PROHIBIT_INVALID_CHARACTER	Prohibited.
# 800 – 899: Base32 Errors		
800	BASE32_ENCODE_BIT_OVERFLOW	The output length exceeds expected characters during encode.
801	BASE32_DECODE_INVALID_SIZE	Invalid input size (1, 3, or 6) for Base32 decode.
802	BASE32_DECODE_INVALID_BIT_SEQUENCE	The Base32 string ends with an invalid bit sequence.
803	BASE32_DECODE_BIT_OVERFLOW	The output length exceeds expected characters during decode.
804	BASE32_MAP_BIT_OVERFLOW	Mapping was not found for input.
805	BASE32_DEMAP_INVALID_BASE32_CHAR	Base32 input is limited to the values [a – z, 2 – 7].
# 900 – 999: DNS Compatible Encoding Errors		
900	DCE_INVALID_DELIMITER	Invalid delimiter in dns string.
901	DCE_DECODE_BIT_OVERFLOW	The output length exceeds expected characters during decode.
902	DCE_DECODE_INVALID_SIZE	Size of output dns bytes is invalid.
# 1000 – 1099: Traditional Chinese/Simplified Chinese (TC/SC) Errors		
1000	TCSC_DOES_NOT_APPLY	The input sequence is not a candidate for TC/SC variation.
1001	TCSC_CHARACTER_MAPPED_OUT	The input character has no TC/SC variant.
1002	INVALID_FILE_FORMAT_NOT_TCSC	Found an invalid TC/SC code point.
1003	NOT_CLASS_A_TCSC	The input domain name is not a Class A domain name.
# 1100 – 1199: Native Errors		
1100	NATIVE_UNSUPPORTED_ENCODING	Native encoding algorithm is not

		supported.
1101	NATIVE_INVALID_ENCODING	Encoding cannot be applied to input.
# 1200 – 1299: Unicode Errors		
1200	UNICODE_SURROGATE_DECODE_ATTEMPTED	A valid surrogate pair is invalid input to Unicode decode.
1201	UNICODE_DECODE_INVALID_VALUE	Unicode can only decode values in the range [0x10000 – 0x10FFFF].
1202	UNICODE_INVALID_VALUE	Unicode values must be in the range [0 – 0x10FFFF].
# 1300 – 1399: UnicodeFilter Errors		
1300	UNICODEFILTER_DOES_NOT_PASS	??MISSING TEXT??
1301	UNICODEFILTER_INVALID_RANGE	Low value precedes high value in a Unicode range.
# 1400 – 1499: Bidi Errors		
1400	BIDI_RULE_1_VIOLATION	The first character must be a character with Bidi property L, R, or AL.
1401	BIDI_RULE_2_VIOLATION	In an RTL label, only characters with the Bidi properties R, AL, AN, EN, ES, CS, ET, ON, BN, and NSM are allowed.
1402	BIDI_RULE_3_VIOLATION	In an RTL label, the end of the label must be a character with Bidi property R, AL, EN, or AN, followed by zero or more characters with Bidi property NSM.
1403	BIDI_RULE_4_VIOLATION	In an RTL label, if an EN is present, no AN may be present, and vice versa.
1404	BIDI_RULE_5_VIOLATION	In an LTR label, only characters with the Bidi properties L, EN, ES, CS, ET, ON, BN, and NSM are allowed.
1405	BIDI_RULE_6_VIOLATION	In an LTR label, the end of the label must be a character with Bidi property L or EN, followed by zero or more characters with Bidi property NSM.
# 1500 – 1599: IDNA Errors		
1500	IDNA_DECODE_MISMATCH	Result of ToUnicode() and then ToASCII() does not match input.
1501	IDNA_LABEL_LENGTH_RESTRICTION	The length of the ASCII sequence exceeds the 63 octet limit



		imposed by RFC 1034.
1502	IDNA_LEADING_COMBINING_MARK	Contains a leading combining-mark code point.
1503	IDNA_IDNA_HYPHEN_RESTRICTION	Must not contain hyphen in third and fourth position. Also, must not start or end with hyphen.
1504	IDNA_CONTEXTUAL_RULE_VIOLATION	Contextual rule validation failed.
# 1600 – 1699: Commingle Filter Errors		
1600	COMMINGLEFILTER_VIOLATION	The domain spans multiple scripts.

## C ERROR CODES

### COMMON ERROR CODES

0	XCODE_SUCCESS	Success.
1	XCODE_BAD_ARGUMENT_ERROR	An input argument is invalid.
2	XCODE_MEMORY_ALLOCATION_ERROR	Failed to allocate needed memory.
3	XCODE_BUFFER_OVERFLOW_ERROR	An input string was too long (>MAX_LABEL_SIZE_X).

### FEATURE SPECIFIC ERROR CODES

# 200 – 299: NORMALIZE-Specific		
200	XCODE_NORMALIZE_EXPANSIONERROR	
201	XCODE_NORMALIZE_PROHIBITEDCHAR	
202	XCODE_NORMALIZE_NULL_CHARACTER_PRESENT	
203	XCODE_NORMALIZE_FIRSTLAST_BIDIERROR	
204	XCODE_NORMALIZE_MIXED_BIDIERROR	
205	XCODE_NORMALIZE_BAD_ARGUMENT_ERROR	
206	XCODE_NORMALIZE_MEMORY_ALLOCATION_ERROR	
207	XCODE_NORMALIZE_BUFFER_OVERFLOW_ERROR	
208	XCODE_NORMALIZE_MAPPEDOUT	
209	XCODE_NORMALIZE_OUTOFRANGEERROR	
210	XCODE_NORMALIZE_NOT_IN_NFC_FORM	
# 300 – 399: TOXXX-Specific		
300	XCODE_TOXXX_STD3_NONLDH	
301	XCODE_TOXXX_STD3_HYPHENERROR	

302	XCODE_TOXXX_ALREADYENCODED
303	XCODE_TOXXX_INVALIDDNSLEN
# 400 – 499: UTIL-Specific	
400	XCODE_UTIL_UTF16DECODEERROR
401	XCODE_UTIL_UTF16ENCODEERROR
402	XCODE_UTIL_LONELY_LOW_SURROGATE
403	XCODE_UTIL_LONELY_HIGH_SURROGATE
404	XCODE_UTIL_INVALID_INPUT_VALUE
405	XCODE_UTIL_INVALID_16BIT_INPUT
406	XCODE_UTIL_INVALID_BYTE_ORDER
407	XCODE_UTIL_INVALID_CONVERTED_VALUE
408	XCODE_UTIL_INVALID_CONVERTED_SURROGATE
409	XCODE_UTIL_INVALID_U_VALUE
410	XCODE_UTIL_INVALID_8BIT_INPUT
411	XCODE_UTIL_INPUT_UNDERFLOW
# 500 – 599: CONTEXTUAL_RULE-Specific	
500	XCODE_CTXRULE_FAILED
501	XCODE_CTXRULE_UNDEFINED_CODEPOINT
502	XCODE_CTXRULE_ZW_NON_JOINER_FAILED
503	XCODE_CTXRULE_ZW_JOINER_FAILED
504	XCODE_CTXRULE_MIDDLE_DOT_FAILED
505	XCODE_CTXRULE_GREEK_LOWER_NUMERAL_SIGN_FAILED
506	XCODE_CTXRULE_HEBREW_PUNCTUATION_GERESH_FAILED
507	XCODE_CTXRULE_HEBREW_PUNCTUATION_GERSHAYIM_FAILED
508	XCODE_CTXRULE_KATAKANA_MIDDLE_DOT_FAILED
509	XCODE_CTXRULE_ARABIC_INDIC_DIGITS_FAILED
510	XCODE_CTXRULE_EXT_ARABIC_INDIC_DIGITS_FAILED
# 600 – 699: BIDI_RULE-Specific	
600	BIDI_RULE_VIOLATION
601	BIDI_RULE_1_VIOLATION
602	BIDI_RULE_2_VIOLATION
603	BIDI_RULE_3_VIOLATION
604	BIDI_RULE_4_VIOLATION
605	BIDI_RULE_5_VIOLATION
606	BIDI_RULE_6_VIOLATION

## EXTENDING THE JAVA IDN SDK

The VeriSign IDN SDK is easily extensible. Users can write a new object or test driver and compile the extension with the build logic included in the SDK distribution file. The build logic will embed the extended logic into a **JAR** file or shared object for direct use in one or

more client applications. Direct extension is an ideal solution for users with multiple applications that require the same set of IDN methods.

---

## WRITING NEW OBJECTS

The Java API in the IDN SDK contains a logic node specifically for extensions to the distribution file. The **com.vgrs.xcode.ext** package under **api/java** contains objects that leverage routines defined in the other packages. To extend the SDK, programmers can use the existing extensions as a template for new development. For instance, the DNS-Compatible Encoding (DCE) object, which ships with the SDK, leverages the Base32 object's encode and decode methods. The output of the DCE encode will be encoded using Base32.

---

## WRITING NEW TOOLS

Command-line tools for VeriSign IDN SDK extensions belong in the **com.vgrs.xcode.cmdline.ext** package. As mentioned above, programmers wishing to write test drivers can use the existing objects as a template for new development. For instance, the DCE test driver imports the **com.vgrs.xcode.ext** package and implements the `main()` method for direct use on the command line.

---

## TESTING WITH RANDOM DATA

The VeriSign IDN SDK contains a command-line tool for generating files of random test data. (Please see section 6.3 of the User's Guide for usage information.) The tool is fairly easy to configure using the command-line options. However, several attributes of the underlying object further affect the behavior of the random generation. Developers may want to alter the attribute values within the source code, and then recompile to effect a change in the output data.

The following table lists attributes and their initial values.

Type	Name	Initial Value	Description
int	MIN_LABELS	2	The minimum number of labels generated when constructing a multilabel sequence.
int	MAX_LABEL	4	The maximum number of labels generated when constructing a multilabel sequence.
int	MIN_LABEL_LEN	2	The minimum number of characters in a label.
int	MAX_LABEL_LEN	8	The maximum number of characters in a label.
int	DEFAULT_LINES	1000	The number of random lines to generate if not specified on the command line.
double	PERCENT_BMP	0.9	Random Unicode or UTF-16 sequences support

			data generation in all 17 Unicode planes. This attribute holds the percentage of code points that are in the Basic Multilingual Plane (less than 0x10000).
--	--	--	------------------------------------------------------------------------------------------------------------------------------------------------------------

## ADDING NEW ERROR CODES

The VeriSign IDN SDK reads error codes at compile time from a file in the **data** directory called **ErrorCodes.txt**. This is a human-readable text file that developers can alter to easily add new codes specific to extended functionality.

Lines in the **ErrorCodes.txt** file have the format:

**<Error Code>\t<Internal Name>\t<Description>**

*Where*

<b>&lt;ErrorCode&gt;</b>	An integer value between 0 and 9999. Codes should not be reused.
<b>&lt;Internal Name&gt;</b>	The variable name that programs will use internally to generate this code.
<b>&lt;Description&gt;</b>	Human-readable details about why the error occurred. (See sections 4.1 and 4.2 for a list of existing error codes.)

## USING DATA FILES

Some extensions to the VeriSign IDN SDK may be table driven and therefore require large amounts of input data during initialization. Properly incorporating these data files requires that they be stored in the SDK distribution JAR file. Client applications must be able to read these files as system resources during runtime. The Java Datafile object allows programmers to easily implement these requirements. The Datafile object supports file decompression, which means large data files can be compressed using gzip or zip algorithms before loading into the JAR. At runtime, these files can be read into data structures using a simple reading syntax.

To use a data file in an IDN SDK extension, follow these steps:

1. Construct the data file, herein referred to as datafile.txt, and move the file into the api/data directory.

2. Use the gzip or zip algorithms to compress the data file. The file should now be called `datafile.txt.gz` or `datafile.txt.zip`.
3. Add the following code (or its equivalent) to the object requiring data access.

```
Iterator reader = null;
String line = null;
try {
    reader = Datafile.getIterator(COMPOSITION_EXCLUSIONS_DATA);
    while (reader.hasNext()) {
        line = (String)reader.next();
        if (line == null) break;
        if (line.length() == 0) continue;
        if (line.charAt(0) == '#') continue;
        /*
        * Process the line here.
        */
    }
} catch (Exception x) {
    line = ": \""+line+"\"";
    throw XcodeError.INVALID_FILE_FORMAT(line);
}
```

## REFERENCES

Topic	URL
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/1.4.2/docs/guide/intl/encoding.doc.html">http://download.oracle.com/javase/1.4.2/docs/guide/intl/encoding.doc.html</a>