V1R2M0
April 18, 2008

User Guide
for
the Intelligent Mail® Barcode
Multi-Platform Encoding Software
on
MS Windows for C and Java

Prepared by: The Powell Group
887 Richmond Road
Ottawa, Ontario, K2A 0G8

Prepared For: United States Postal Service
Dewey Building
8403 Lee Highway
Merrifield, VA 22082-8412

Contract: 1BCHSO-05-B-3090

Version 1 R 2 M 0

uspsEncoderMsWindows32-1.2.0

Table of Contents

Change History

Rev	Date	Section	Narrative (Items affected)
V1R1M0	Apr 24/06	All	Creation of a separate User Guide for users of C and Java in the Windows Environment with USPS requested changes.
V1R1M01	May 4/06	As specified by Reviewer's email: 1.6, 2.3, 3.0, 3.1, 3.4.1.3, 3.4.1.4, 5.0,	Added TrueType font to list of fonts, removed footnote, standardized on WINDOWS, defined MinGW, capitalized IVP, replaced encodetr with usps4cb, removed yellow highlighting.
V1R2M0	Oct 2/07	All	Name change to Intelligent Mail® barcode in text and in sample programs.
V1R2M0	April 1, 2008	All	Final review and text changes to reflect name change to Intelligent Mail barcode.
V1R2M0	April 1, 2008	1.6 2.2 2.3 5.2 ALL 2.3	Add True Type to 1.2 List of Fonts, "usps4cb.o" becomes "usps4cb.so" for C distribution Clarification of names in distribution files and Java name for gov.usps.USPS4CB Updates to Table 1. new sample Java programs. Addition of usps4cb.jar to the Java files Version, Release, and Modification (V1R2M0) numbering convention added to the name of the software release. Therefore, V1R2M0 becomes "uspsEncoderMsWindows32-1.2.0" for this release and version of the software. MS Word version of the User Guides will no longer be distributed and the Zip file will contain only the PDF.

Trademark Legal Notices

The following trademarks are used in this document.

Microsoft, Visual Basic, Visual C++, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

United States Postal Service and ZIP + 4 are registered trademarks of the U.S. Postal Service. ZIP Code is a trademark of the U.S. Postal Service.

Intelligent Mail® is a registered trademark of the U.S. Postal Service.

All other trademarks mentioned are the property of their respective owners, including IBM's MVS, z/OS, OS/390, TSO, PL/I, and PKWARE's ZIP and PKZIP.

In this User Guide, the Pkware Zip software is referenced as "Zip" or "zip" if included in Java code. All references to postal "ZIP codes" are "ZIP."

1.0. Introduction

The US Postal Service (USPS) has adopted the Intelligent Mail® barcode for use in the USPS mail stream. It is also known as the USPS 4-State Customer Barcode, the OneCode^{SOLUTION} Barcode and abbreviated in a number of ways including: OneCode (4CB), OneCode (4-CB), IM™ barcode, 4CB or 4-CB in this and other documents.

This document describes how to install and use the Intelligent Mail barcode multi-platform encoding software. The software module is also referred to as USPS4CB in this document. The software encodes a 20-digit tracking string and a 0-, 5-, 9- or 11-digit routing string into a string of 65 specific characters.

This *User Guide* is for Microsoft Windows (referenced in this document as “Windows”). The encoding software has been designed to interface with C and Java. The encoder software has been tested on Windows 2000, Windows XP and Windows Vista. A separate User Guide is available for those users wishing to access the software encoder from Microsoft Office applications as well as Microsoft Visual Basic and Microsoft Visual C++ applications.

This User Guide is for C and Java users and describes:

- The relevant terminology and data requirements for using the USPS4CB encoding software module (Chapter 1);
- How and where to obtain the module package for USPS4CB (Chapter 2);
- How to extract the data from the module package (Chapter 2);
- How to compile, execute, and test USPS4CB in the C environment (Chapters 3 and 4);
- How to compile, execute, and test USPS4CB in a Java environment (Chapters 5 and 6); and
- The Return Codes that indicate whether the Intelligent Mail barcode character string was created successfully, and if not, the probable cause of the error Appendix A).

1.1. System Definition

The USPS4CB encoding software module interfaces with the software a mailer uses to print addresses, other human readable contents, and barcode fonts onto mail pieces.

The USPS4CB encoding software module does the following:

- Performs error checking;
- Combines a 20-digit Tracking Code, which uniquely identifies the mail piece, with an 11-digit Routing Code, which identifies the destination of the mail piece, into a single string;

- Encodes that string into a 65-character Intelligent Mail barcode string;
- Provides the 65-character Intelligent Mail barcode string; and
- Provides a Return Code to indicate whether the 65-character Intelligent Mail barcode string was successfully generated, or the reason why it was not successfully generated.

The mailer uses the Intelligent Mail barcode string to generate the actual barcode, which can be printed on labels or directly onto mail pieces.

1.2. Tracking Code Definition

The Tracking Code uniquely identifies the mail piece. It is constructed by combining the values in the following fields in this order:

The Tracking Code string (`TrackString`) contains 20 digits in the following form:
AASSSCCCCCCNNNNNNNNN

where:

AA is the 2-digit barcode Identifier. The second digit must be 0, 1, 2, 3, or 4.
SSS is the 3-digit Service Type Identifier code.
CCCCCC is the 6-digit Mailer Identifier.
NNNNNNNNN is the 9-digit Serial Number. Please note that the Mailer ID can be a 6-digit or 9-digit number based upon mailer's volume. Since the Mailer ID and Serial Number are interdependent, the Serial Number is a 6-digit or 9-digit number depending on the length of the Mailer ID. If the Mailer ID is 6 digits, the Serial Number must be 9 digits. If the Mailer ID is 9 digits, there are 6 digits available for the Serial Number.

The scope, classification, and description of the Tracking Code are described further in the USPS publication, USPS-B-3200. This document is available at USPS's Rapid Information Bulletin Board System (RIBBS) website at: http://ribbs.usps.gov/OneCodeSOLUTION/.

1.3. Routing Code Definition

The Routing Code identifies the destination of the mail piece. The Routing Code must contain any one of the following:

- 5 digits representing a 5-digit ZIP Code,
- 9 digits representing a ZIP + 4,
- 11 digits representing a ZIP + 4 plus a Delivery Point Code, or
- no digits representing an unknown Routing code.

The ZIP Code must never be padded with zeroes, spaces or nulls that are not part of the "valid" ZIP code. Therefore, the resulting string can be 5, 9, 11, or 0 digits in length and can have one of the following forms:

- ZZZZZ (5 digit string),
- ZZZZZZZZZ (9 digit string),
- ZZZZZZZZZZZ (11 digit string), or
- (no Routing Code).

The Routing Code can have one of 4 lengths. The sample Installation Verification Programs (IVP) provided for C show how the Routing Code parameter can be handled as a variable length string.

The scope, classification, and description of the Routing Code are described further in the USPS publication, USPS-B-3200. This document is available at USPS's Rapid Information Bulletin Board System (RIBBS) website at: http://ribbs.usps.gov/OneCodeSOLUTION/.

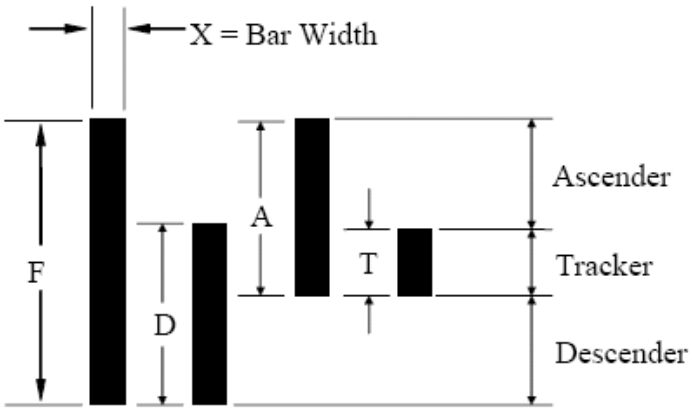
1.4. Intelligent Mail Barcode Font Definition

The Intelligent Mail barcode consists of a barcode string of 65 characters. Each bar can be printed in one of four states where each bar is one of A, D, T, or F. The Intelligent Mail barcode is based on a tracker with ascenders and descenders. The four possible states are:

- **T**racker (neither ascender nor descender),
- **F**ull (both ascender and descender),
- **A**scender only, and
- **D**escender.

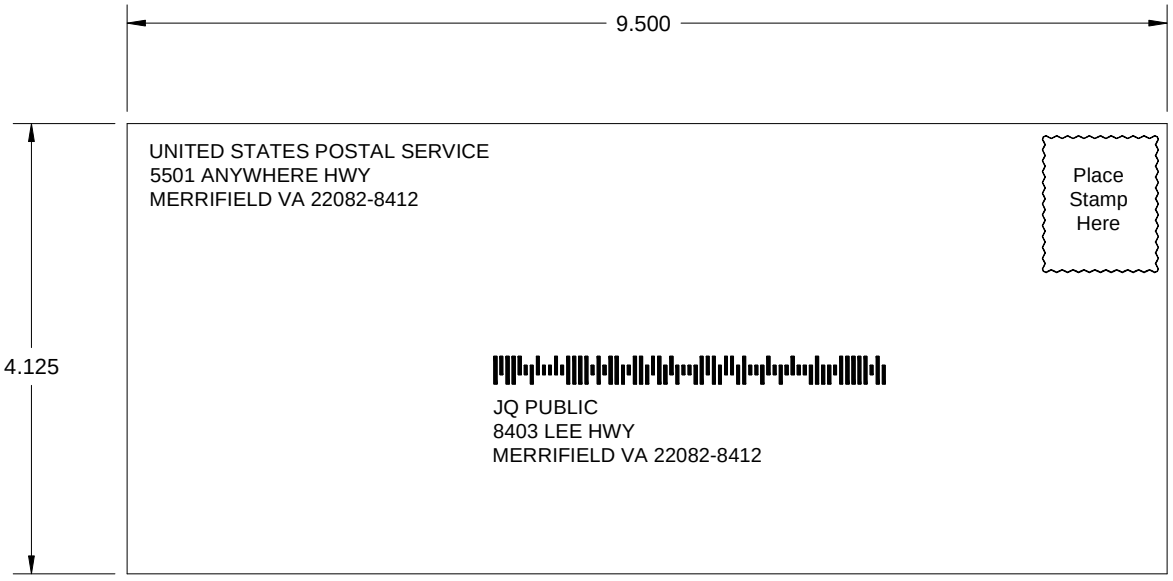
These states appear in Figure 1 and in the sample envelope in Figure 2.

Figure 1. Intelligent Mail Barcode Bar Types and Elements



Bar Type	Elements
T – Tracker	Tracker
D – Descender	Tracker, Descender
A – Ascender	Tracker, Ascender
F – Full Height	Tracker, Ascender, Descender

Figure 2. Envelope with Intelligent Mail Barcode



1.5. Return Codes Definition

The Return Code indicates whether the 65-character Intelligent Mail barcode string was successfully generated, or the reason why it was not successfully generated. Appendix A lists and describes the Return Codes.

1.6. Intelligent Mail Barcode Printing Guidelines

Fonts for the Intelligent Mail barcode are available for the following production printing environments:

- AFP (MVS, AS/400, VM, VSE),
- HP PCL3
- PostScript,
- Xerox Metacode, and
- TrueType.

A *Font Users Guide* and details regarding the installation and proper use of the Intelligent Mail barcode fonts is available at USPS's Rapid Information Bulletin Board System (RIBBS) website at: http://ribbs.usps.gov/OneCodeSOLUTION/.

2.0. Obtaining and Extracting the Encoding Software

A Windows dll file version of the Intelligent Mail barcode encoding software is provided for calling programs written in C. It is distributed in the form of a standard Zip file.

This chapter describes how to obtain the encoding software, and how the directory structure appears when the Zip file is expanded by any standard unzip utility.

2.1. Naming

The internal name for Intelligent Mail barcode encoding software is identified as USPS4CB. It is referenced as `usps4cb.dll` in the directories.

2.2. Obtaining Intelligent Mail Barcode Encoding Software Package

The Intelligent Mail barcode encoding software package is distributed as a standard Zip file, which you can download from the U.S. Postal Service website at: http://ribbs.usps.gov/OneCodeSOLUTION/ .

2.3. Extracting Intelligent Mail Barcode Encoding Software

Table 1. Directory structure for Intelligent Mail Barcode Encoding Software

DirectoryFile Name	Description of Contents
uspsEncoderMsWindows32-1.2.0.zip	Name of the distribution's zip file
/uspsEncoderMsWindows32-1.2.0	Root directory for the distribution
readme.txt	Text file description of the directory contents
/ivp	Installation Verification program
civp.c	Installation Verification Program for C
/modules	Directory for modules
/c	Directory for C
libusps4cb.a	Library file for linking C programs to usps4cb.dll
usps4cb.dll	Intelligent Mail Encoder Windows dll
/java	Directory for Java files
USPS4CB.java	Sample Java program
usps4cb.jar	Java file containing gov.usps.USPS4CB.class
IVP.java	Installation Verification Programs for Java
usps4cb.dll	Intelligent Mail Encoder Windows dll
/userGuide	
uspsEncoderMsWindows32-1.2.0.pdf	pdf version of User Guide

The directories within the expanded Zip file have the following contents:

/uspsEncoderMsWindows32-1.2.0/ivp

This directory contains one file:

- civp.c: the Installation Verification Program (IVP) for the Windows environment in the form of C source code. The IVP program was developed and tested using the GNU gcc compiler for Windows. This compiler is available within the MinGW distribution. MinGW is a collection of freely available and freely distributable Windows specific header files and import libraries combined with the GNU toolsets that allows for the production of native Windows programs that do not rely on any 3rd-party C runtime DLLs.

/uspsEncoderMsWindows32-1.2.0/modules/c

This directory contains two files:

1. A “lib” file called libusps4cb.a, to be used when linking a C program with the usps4cb.dll encoder module.
2. A “dll” called usps4cb.dll, compiled with gcc, having internal entry point “USPS4CB”, which can be called from a C program.

/uspsEncoderMsWindows32-1.2.0/modules/java

This directory contains four files:

1. `USPS4CB.java,` a sample Java program that loads and calls `usps4cb.dll,`
2. `usps4cb.jar`, contains compiled instances of gov.usps.USPS4CB.java and of IVP.java that can be used for a quick test and to import gov.usps.USPS4CB into a program,
3. `IVP.java`, the Java Installation Verification Program, and
4. `usps4cb.dll,` the Java based MS Windows dll file version of the encoder.

/uspsEncoderMsWindows32-1.2.0/userGuide

This directory contains the *User Guide* for the Intelligent Mail barcode encoding software for Windows for C and Java in PDF format.

3.0. Using the Encoding Software from C

Once the installation package is downloaded from the USPS RIBBS web site and unzipped, the next tasks are to:

1. Obtain the Installation Verification Program (IVP),
2. Obtain the USPS4CB dll module, usps4cb.dll,
3. Compile and execute the IVP, and
4. Evaluate results.

The sections below provide step-by-step instructions for performing each of these tasks.

Section 3.3 describes the calling parameters for C, and Section 4.0 provides a complete sample installation verification program for C.

3.1. Obtain the Installation Verification Program (IVP)

The Installation Verification Program (IVP) is a standard C source file, civp.c. The IVP verifies that the installation procedure is correctly accomplished. It contains eight tests with valid input data and three tests with intentionally bad data to show some of the more common errors and the encoder's response to them. The last line of the IVP reports the results of the test, and confirms that the encoder is ready to perform. The following procedure describes the IVP installation and testing procedures.

Step 1. **Create** a directory for the IVP test, for example: `/ivptest`.

Step 2. **Install** `usps4cb.dll and libusps4cb.a` in the Windows directory, `/ivptest`. Any copying of these modules from one machine to another must be in binary mode.

Step 3. **Install** civp.c in the Windows directory, `/ivptest`.

Step 4. **View** the C source file with an editor to ensure it was successfully installed.

3.2. Compile and Execute the IVP Program for C.

Step 1. **Compile and Link**

Since the gcc compiler was used to compile the object module, the example shown here will use the same compiler for the IVP program:

```
gcc  -o  civp.exe civp.c –L./  -lusps4cb
```

Step 2. Execute

To execute the module, just enter the executable module name on a command line:

```
civp.exe
```

The IVP program runs a set of 11 internally coded test cases and writes the results to the output stream. The inputs and outputs for each test case, along with a success or failure message, are also written to the output stream. At the end of the run, the final lines written should be, assuming a successful test:

```
End USPS4CB IVP C Run
11 Test Cases were successful
0 Test Cases failed
Exiting with Return Code 0
```

Note: The last line of each test case indicates whether the test was successful. In the example that follows, all of the test cases are successful, even though three test cases (5, 6, and 7) have errors. Success does not indicate that the test case has a Return Code of 0, but that the test case generated the expected Return Code.

This sample is for C.

```
 Start USPS4CB IVP Run for C
 --------------
 Test Case # 1
 -------------
  Inputs are:
  Tracking Code: 53379777234994544928
  Routing Code:  51135759461
  Calling the USPS4CB encoder...
  Outputs are:
  Return Code:  0
   Encoded Intelligent Mail barcode:
AFDTDAFFDFTDADTDDFTTFDTATATFFFDFTTFFFTFDDTDAAFATDFTFDFDTTTDTTFDA
  Test Case #   1 was Successful
  --------------
  Test Case #    2
  --------------
  Inputs are:
  Tracking Code: 53055494689272602879
  Routing Code:  13765583689
  Calling the USPS4CB encoder...
  Outputs are:
  Return Code:  0
  Encoded Intelligent Mail barcode:
 AFDADAFTAFDTDFTFTFDAAFDDTAFDFDTTFADATAATAAAADDDFAAAAATDADAFADFTTT
  Test Case #   2 was Successful
  --------------
  Test Case #    3
  --------------
```

```
 Inputs are:
 Tracking Code: 40120111574675115924
 Routing Code:  62176609110
 Calling the USPS4CB encoder...
 Outputs are:
 Return Code:  0
 Encoded Intelligent Mail barcode:
DDAFFFDAFTFDFFAAATTTDDFFTFADDFAFTTDAAAAAADFTTFDTAFDDTDDADATDAFAFF
 Test Case #   3 was Successful
 --------------
 Test Case #    4
 --------------
 Inputs are:
 Tracking Code: 82205455868913559972
 Routing Code:  54765515722
 Calling the USPS4CB encoder...
 Outputs are:
 Return Code:  0
 Encoded Intelligent Mail barcode:
TTTADTFFADAFDTTFDTAADATFFFADFFTDDFFFATDADAATAAADDATFTAADFADTADADD
 Test Case #   4 was Successful
 --------------
 Test Case #    5
 --------------
 Inputs are:
 Tracking Code: 57379777234994544928
 Routing Code:  51135759461
 Calling the USPS4CB encoder...
 Outputs are:
 Return Code: 11
 Encoded Intelligent Mail barcode:
 Test Case #   5 was Successful
 --------------
 Test Case #    6
 --------------
 Inputs are:
 Tracking Code: 53055494689272602879
 Routing Code:  137655B3689
 Calling the USPS4CB encoder...
 Outputs are:
 Return Code: 13
 Encoded Intelligent Mail barcode:
 Test Case #   6 was Successful
 --------------
 Test Case #    7
 --------------
 Inputs are:
 Tracking Code: 4012X111574675115924
 Routing Code:  62176609110
 Calling the USPS4CB encoder...
 Outputs are:
 Return Code: 10
 Encoded Intelligent Mail barcode:
 Test Case #   7 was Successful
 --------------
```

```
 Test Case #    8
 --------------
 Inputs are:
 Tracking Code: 01234567094987654321
 Routing Code:
 Calling the USPS4CB encoder...
 Outputs are:
 Return Code:  0
 Encoded Intelligent Mail barcode:
ATTFATTDTTADTAATTDTDTATTDAFDDFADFDFTFFFFFTATFAAAATDFFTDAADFTFDTDT
 Test Case #   8 was Successful
 --------------
 Test Case #    9
 --------------
 Inputs are:
 Tracking Code: 01234567094987654321
 Routing Code:  01234
 Calling the USPS4CB encoder...
 Outputs are:
 Return Code:  0
 Encoded Intelligent Mail barcode:
DTTAFADDTTFTDTFTFDTDDADADAFADFATDDFTAAAFDTTADFAAATDFDTDFADDDTDFFT
 Test Case #   9 was Successful
 --------------
 Test Case #   10
 --------------
 Inputs are:
 Tracking Code: 01234567094987654321
 Routing Code:  012345678
 Calling the USPS4CB encoder...
 Outputs are:
 Return Code:  0
 Encoded Intelligent Mail barcode:
ADFTTAFDTTTTFATTADTAAATFTFTATDAAAFDDADATATDTDTTDFDTDATADADTDFFTFA
 Test Case #  10 was Successful
 --------------
 Test Case #   11
 --------------
 Inputs are:
 Tracking Code: 01234567094987654321
 Routing Code:  01234567891
 Calling the USPS4CB encoder...
 Outputs are:
 Return Code:  0
 Encoded Intelligent Mail barcode:
AADTFFDFTDADTAADAATFDTDDAAADDTDTTDAFADADDDTFFFDDTTTADFAAADFTDAADA
 Test Case #  11 was Successful
 --------------
 End USPS4CB IVP C Run
 11 Test Cases were successful
  0 Test Cases failed
 Exiting with Return Code 0
```

3.3. Use on a MS Windows System with C

The following describes the calling procedures and the parameters involved for C. Three parameters are required:

- The Tracking Code (`TrackString`) input parameter,
- The Routing Code (`RouteString`) input parameter, and
- The Intelligent Mail barcode (`BarString`) output parameter.

This section describes the requirements for their use with USPS4CB.

3.3.1 Parameters

The USPS4CB module uses the character string parameters shown in Table 2.

Table 2. Parameter names and information

Code name	Parameter name	String size	String length	Input or output	Further information
Tracking Code	`TrackString`	20 bytes	21 digits	Input	Sections 1.2, 3.3.1.1
Routing Code	`RouteString`	11 bytes	12 digits	Input	Sections 1.3, 3.3.1.2
Intelligent Mail barcode	`BarString`	65 bytes	66 characters	Output	Sections 1.4, 3.3.1.3

The length of each parameter must be exactly equal to the number of characters or digits for that parameter, plus 1 for a null terminator. Please consult the IVP source to see how the parameter should be declared for each language.

See Chapter 1 for brief definitions of the Tracking Code, Routing Code, and the Intelligent Mail barcode. Additional information for these parameters is contained in the USPS publication, USPS-B-3200. See Appendix A for a list of Return Codes.

3.3.1.1 Tracking Code (TrackString) Parameter

The Tracking Code (`TrackString`) is an input parameter. It should be declared as a fixed-length string whose length is 21 bytes (20 data bytes, plus null terminator). The Tracking Code is checked to ensure all characters are digits with a value from 0 to 9. Any failing character will cause an error return with Return Code 10. The second character in the Tracking Code must be the number 0, 1, 2, 3, or 4. If this is not true, it will cause an error return with Return Code 11. If this condition is detected, the user should examine the contents of the Tracking Code parameter to ensure only valid characters are present.

3.3.1.2 Routing Code (RouteString) Parameter

The Routing Code (`RouteString`) is an input parameter. It should be declared as a character string whose length is 12 bytes (11 data bytes plus null terminator). The Routing Code may consist of 0, 5, 9, or 11 digits, with 0 corresponding to the case where there is no Routing Code. Valid layouts for this field would be as follows:

```
N                    0  Routing Code digits, null character
ddddd N              5  Routing Code digits, null character
ddddddddd N          9  Routing Code digits, null character
dddddddddddN         11 Routing Code digits, null character
```

Scanning from the left, when a character is found, it must be a digit whose value is from 0 to 9. Any character found in the scan, which is not a digit from 0 to 9 results in an error return with Return Code 13. The Return Code of 12 signifies that the length of the Routing Code is neither 0, 5, 9, nor 11.

3.3.1.3 The Intelligent Mail Barcode (BarString) parameter

The Intelligent Mail Barcode (`BarString`) is an output parameter. It should be declared as a fixed-length character string whose length is 66 bytes (65 data bytes plus null terminator). When a call is made to USPS4CB with valid input parameters, USPS4CB returns 65 encoded four-state symbols of A, D, F, or T in the parameter.

> Note: On every call, when setting the values for the input parameters, `BarString` should be populated with 65 blanks. This is recommended, since `BarString` is not altered when an error return is taken, and residual data might survive from a preceding call. The IVP programs all show this action.

3.3.1.4 Return Code

On return from USPS4CB, the Return Code is always set. A Return Code of 0 means no errors were detected and `BarString` will contain 65 encoded four state symbols of A, D, F, or T. If the Return Code is not 0, the `BarString` is not altered, and nothing is returned in `BarString`. This means that the calling program must always check the Return Code before deciding to use the contents of the `BarString` output parameter. The IVP programs show how to check the Return Code value for C, or Java.

See Chapter 4 for examples of the Return Code for C and Chapter 6.0 for examples for Java. Return Code Definitions are presented in Appendix A.

3.4. Sample Calling Sequence

Consult the IVP source code provided for C or for Java to see how the calling sequence is set up and how the results are processed. For each language, there are 11 internally coded test cases arranged in a table, with expected results for each set of input parameters. Three test cases have intentional parameter errors, to generate error codes 11, 13, and 10. These are deemed successful when the encoder detects the errors correctly and returns the appropriate error code.

4.0. Sample C Installation Verification Program (CIVP)

The following is the sample C installation verification program (CIVP).

```
/************************************************************************
**                                                                     *
**  CIVP - 'C' Installation Verification Program to call               *
**         U.S. Postal Service Intelligent Mail Barcode Encoder V1R2M0 *
**         TPG Technology Consulting Limited                           *
**         Source code compiles on Windows/XP with:                    *
**         MinGW gcc Version 3.4.2 or                                  *
**         Microsoft Visual C/C++ 6.0                                  *
**         March 3, 2008                                               *
**                                                                     *
************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

const int ivptab_size = 11;     /* num of entries in test case table */
      int j,success_count,fail_count ;

const
struct testcase_list
       {
       char      TestNum[6];
       char      TrackStr[20];
       char      RouteStr[11];
       char      ExpectBar[65];
       int       ExpectRC;
       } testcase[11] =
{"000001",               /* Test Case 1: Valid Track & Route RC=0 */
 "53379777234994544928",
 "51135759461",
 "DAFDTDAFFDFTDADTDDFTTFDTATATFFFDFTTFFFTFDDTDAAFATDFTFDFDTTTDTTFDA",
 0,
 "000002",               /* Test Case 2: Valid Track & Route RC=0 */
 "53055494689272602879",
 "13765583689",
 "AFDADAFTAFDTDFTFTFDAAFDDTAFDFDTTFADATAATAAAADDDFAAAAATDADAFADFTTT",
 0,
 "000003",               /* Test Case 3: Valid Track & Route RC=0 */
 "40120111574675115924",
 "62176609110",
 "DDAFFFDAFTFDFFAAATTTDDFFTFADDFAFTTDAAAAAADFTTFDTAFDDTDDADATDAFAFF",
 0,
 "000004",               /* Test Case 4: Valid Track & Route RC=0 */
 "82205455868913559972",
 "54765515722",
 "TTTADTFFADAFDTTFDTAADATFFFADFFTDDFFFATDADAATAAADDATFTAADFADTADADD",
 0,
 "000005",              /* Test Case 5: 2nd byte Track inv. RC =11*/
 "57379777234994544928",
 "51135759461",
 "                                                                ",
 11,
 "000006",             /* Test Case 6: invalid char Route  RC=13  */
 "53055494689272602879",
 "137655B3689",
```

```
 "                                                                  ",
 13,
 "000007",              /* Test Case 7: invalid char Track  RC=10  */
 "4012X111574675115924",
 "62176609110",
 "                                                                  ",
 10,
 "000008",              /* Test Case 8: Valid Route 0 bytes  RC=0  */
 "01234567094987654321",
 "",
 "ATTFATTDTTADTAATTDTDTATTDAFDDFADFDFTFFFFFTATFAAAATDFFTDAADFTFDTDT",
 0,
 "000009",              /* Test Case 9: Valid Route 5 bytes  RC=0  */
 "01234567094987654321",
 "01234",
 "DTTAFADDTTFTDTFTFDTDDADADAFADFATDDFTAAAFDTTADFAAATDFDTDFADDDTDFFT",
 0,
 "000010",              /* Test Case 10: Valid Route 9 bytes RC=0  */
 "01234567094987654321",
 "012345678",
 "ADFTTAFDTTTTFATTADTAAATFTFTATDAAAFDDADATATDTDTTDFDTDATADADTDFFTFA",
 0,
 "000011",              /* Test Case 11: Valid Route 11 bytes RC=0 */
 "01234567094987654321",
 "01234567891",
 "AADTFFDFTDADTAADAATFDTDDAAADDTDTTDAFADADDDTFFFDDTTTADFAAADFTDAADA",
 0 };



char TrackString[21];     /* Input parameter Track String + 1 null*/
char RouteString[12];     /* Input parameter Route String + 1 null*/
char BarString[66];       /* Output parameter Bar String + 1 null */


int RetCode;              /* Return code from the usps4cb encoder */

__declspec (dllimport) int USPS4CB( char *TrackPtr, char *RoutePtr, char *BarPtr);

int main(void)
{

printf("Start USPS4CB IVP C Run\n");

j = success_count = fail_count = 0;

while (j < ivptab_size )
  {
   printf("--------------\n");
   printf("Test Case # %i\n",j+1);
   printf("--------------\n");

   memcpy(TrackString,testcase[j].TrackStr,20);
   TrackString[20] = '\0';

   memcpy(RouteString,testcase[j].RouteStr,11);
   RouteString[11] = '\0';

   memset(BarString, ' ', sizeof(BarString));   /* Prime blanks!  */
   BarString[65] = '\0';                        /* Set null       */
```

```
    printf("Inputs are:\n");
    printf("Tracking Code: %s\n",TrackString);
    printf("Routing Code: %s\n",RouteString);

    printf("Calling the USPS4CB encoder...\n");

    /***************************/
    /* Call the usps4cb encoder */
    /***************************/

    RetCode =  USPS4CB(TrackString,RouteString,BarString);

    printf("Outputs are:\n");                  /* Print the outputs */
    printf("Return Code: %i\n",RetCode);
    printf("Intelligent Mail Barcode: %s\n",BarString);

                                              /* Check the outputs  */
    if ((memcmp(BarString,testcase[j].ExpectBar,65) == 0) &
        (RetCode == testcase[j].ExpectRC) )
       {                                      /* Declare success    */
        printf("Test Case %i was Successful\n",j+1);
        success_count++;
       }
       else
          {                                   /* Declare failure    */
          printf("Test Case %i has Failed\n",j+1);
          fail_count++;
          }

    j++;
   }

   printf("--------------\n");
   printf("End USPS4CB IVP C Run\n");           /* Final summary     */
   printf("%i Test Cases were successful\n",success_count);
   printf("%i Test Cases failed\n",fail_count);
   if ((success_count == ivptab_size) && (fail_count == 0))
      {
       printf("Exiting with Return Code 0\n");
       exit(0);                         /* Overall success set RC=0 */
      }
      else
         {
          printf("Exiting with Return Code 8\n");
          exit(8);         /* At least 1 test case failed set RC=8 */
         }

}
```

5.0. Using the Java Encoder Software

Chapter 3 describes how to download the installation package and the directory structure to use on the workstation. The next tasks are to:

1. Download the Java distribution package.
2. Compile the two Java source files.
3. Execute the Java distribution package.
4. Evaluate results.

Section 5.2 below provides step-by-step instructions for performing each of these tasks.

Chapter 6 includes a sample Java program (`USPS4CB.java`).

5.1. Process Description

A Windows dll-file version of the USPS Encoder is provided for use with Java. The encoding module, `usps4cb.c`, is compiled and linked using the gcc GNU C Compiler to produce a Windows DLL-file, `usps4cb.dll`, with an entry point of `jencode`.

A Java program passes the Tracking Code (`TrackString`) and Routing Code (`RouteString`) input parameters as standard Unicode Java strings, and receives a results string, `BarString`, also in Unicode, in the form of `rc.barcode`.

Where:

`rc`	The two-digit Return Code
`.`	period used to separate the Return Code and barcode string
`barcode`	The 65-character Intelligent Mail barcode (`BarString`), if USPS4CB encoder successfully created it. If the Intelligent Mail barcode (`BarString`) was not successfully created, USPS4CB encoder returns a single space character instead of the 65-character string.

5.2. Installing Files from the Java Distribution Package

Step 1. Unzip the distribution file into a directory. The Java distribution consists of four files:

`USPS4CB.java`
`usps4cb.jar`
`IVP.java`
`usps4cb.dll`

Step 2. Compile the two Java-source files as follows:

Copy USPS4CB.java to a directory structure of gov/usps/ to then invoke the Java compiler as follows:

```
javac  gov/usps/USPS4CB.java
javac  IVP.java
```

Step 3. Execute. Call the two Java programs as follows to test that the distribution works properly. Note that the library usps4cb.so must be available in the Java class-path.

```
java   -cp  .  gov.usps.USPS4CB
java   -cp  .  IVP
```

The distribution Zip-file includes `usps4cb.jar`. This file contains compiled instances of `USPS4CB.java`, `IVP.java`, and a third sample program, `AUspsTest.java`. The `usps4cb.jar` file is a self-contained file that is Java executable and works on 32-bit MS Windows. It also has been tested on MS Windows 2000, MS XP and MS Vista.

Call these compiled instances for a test of the distribution as follows:

```
java  -cp  usps4cb.jar   gov.usps.USPS4CB

java  -cp  usps4cb.jar   IVP

java  -cp  usps4cb.jar   mytest.AUspsTest
```

`usps4cb.jar` can also be used to import `gov.usps.USPS4CB` into a Java program and as a run-time class-path to find and load `usps4cb.dll`.

5.3. About the Java Native Interface (JNI) and Encoder Use

The USPS4CB encoder is written in the C programming language. To interface Java to a C-based module the Java Native Interface Application Programming Interface, or JNI API, is used.

JNI is platform dependent. Each platform vendor supplies C source in the form of included C header files to allow compilation of the C source code in a manner that allows a Java program to load and call the compiled C module. In the case of the USPS4CB encoder, the C code, `usps4cb.c`, has been compiled to produce a Windows DLL-file `usps4cb.dll`

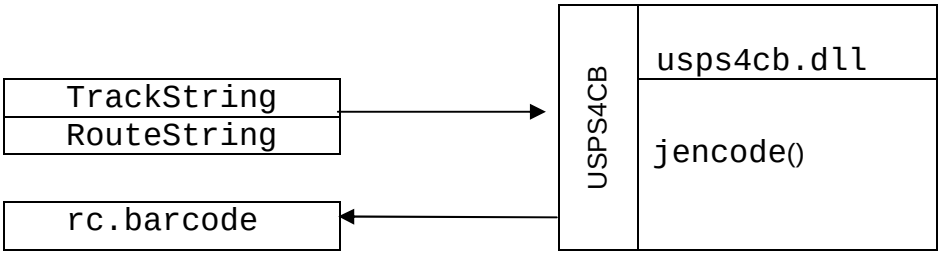
JNI requires that the name of the invoking Java object be defined when the DLL is generated. After the DLL has been compiled, it can be invoked only by a Java object whose name matches the one provided in the DLL-file. The Java object name declared when the `usps4cb.dll` was created is `gov.usps.USPS4CB`. Therefore, in order to use

the supplied usps4cb.dll file, the invoking Java object must be named gov.usps.USPS4CB.

The Java object that loads and invokes the jencode function, the entry point in usps4cb.dll, must be named USPS4CB.java for this reason. This is also why the installation verification program, IVP.java, uses USPS4CB.java to invoke the encoder routine.

When integrating the USPS4CB encoder into your environment, you may use the USPS4CB.java, in the same manner used by the IVP.java program, or replace it with custom code. If custom code is used, the object that invokes the usps4cb.dll must be named USPS4CB.

The USPS4CB.java loads usps4cb.dll and calls an entry-point called jencode to obtain the encoded Intelligent Mail barcode string in response to the Tracking Code (TrackString) and Routing Code (RouteString) input parameters.



See the sample Java code, USPS4CB.java and IVP.java, for detailed reference information. What follows is a short description of relevant portions of the code. USPS4CB.java has a main method and can run as a console application for use from the console. Its purpose, however, is to act as a Java class to load and call the C language compiled DLL usps4cb.dll, and to serve as a front end, a Java Bean, to a calling Java program.

The following is an excerpt from the included USPS4CB.java source that shows the relevant portions of the code.

Code	Comments
`USPS4CB usps = gov.usps.USPS4CB.getInstance();`	define USPS4CB
`usps.setTrack( UPPSTrackNo );`	set the tracking code
`usps.setRoute( USPSRouteNo );`	set the routing code
`sr = usps.getBarCode();`	obtain the resulting Intelligent Mail barcode

These four statements capture the key part of the code, namely:

1. Define the object USPS4CB

2. Set the Tracking Code.
3. Set the Routing Code.
4. Get the answer, `getBarcode(),` the encoded Intelligent Mail barcode string.

A successful call returns a string of 68-chracters long configured as follows:

`00.FDDTAFDFDTTFDDDFAAAAATDADAFADFTTTADATAATAAAADDDFAAAAATDADAFADFTTT`

where
`00` is a Return Code for a successful call, and

`FDDTAFDFDTTFDDDFAAAAATDADAFADFTTTADATAATAAAADDDFAAAAATDADAFADFTTT` contains the 65 characters which, when printed using the Intelligent Mail barcode font, will result in the Intelligent Mail barcode.

An unsuccessful call returns a string of 4 characters long configured as follows:

`nn.b`

Where:
`nn` is a numeric Return Code other than `00,`
"**.**" a dot character, separates the Return Code from
`b,` a space character.

See section 1.2 for a description of the Tracking Code parameter. See section 1.3 for a description of the Routing Code parameter. See Appendix A for a list of Return Codes and definitions.

6.0. Sample Java Programs

6.1. Sample Java Program to Invoke the Shared Object Module

USPS4CB.java is the sample code to load and invoke the C library shared object module usps4cb.dll. USPS4CB.java can be run standalone but its main function is to be used by the sample application IVP.java, which calls it and runs eleven self-contained test cases. The following are the comments extracted from the sample program, USPS4CB.java.

```
package gov.usps;

import java.util.*;
import java.io.*;
import java.io.IOException;
import java.net.*;
import java.util.jar.*;

/**
 * @author     TPG Technology Consulting (TPG)
 * @created    January 06, 2006
 * @updated    February    11, 2008
 *
 *          This Class is designed for use by a java program to load
 *          and call the USPS Intelligent Mail barcode
 *          Encoder and available as a compiled and linked library
 *          'usps4cb.so' for Linux/Unix and usps4cb.dll for MS Windows.
 *
 *          A java program would use gov.usps.USPS4CB as follows:
 *
 *          gov.usps.USPS4CB usps = gov.usps.USPS4CB.getInstance();
 *          usps.setTrack( UPPSTrackNo );    set the tracking code
 *          usps.setRoute( USPSRouteNo );     set the routing code
 *          sr = usps.getBarCode(); obtain the resulting Intelligent Mail barcode
 *
 *          Note that this Class is an example of how to load and call
 *          the corresponding C-library. You may want to code your own class
 *          to load and call the C-library provided the class is
 *          named also gov.usps.USPS4CB.
 *
 *          This Class name limitation is imposed by Java's JNI interface
 *          which forces a predefined name in the C-library.
 *          gov.usps.USPS4CB is the defined Class name in the distributed
 *          platform dependent C-libraries, e.g. USPS Intelligent Mail
 *          barcode Encoder.
 *
 *          The method USPS4CB.getInstance() is included and recommended
 *          in place of 'new' to create an instance of USPS4CB.
 *          The getInstance() method ensures the object is created once and
```

```
*       returns same reference to subsequent calls.
*       The getInstance() method prevents multiple object instances
*       given that the native C-library Encoder is not thread safe;
*       it is not re-entrant.
*
*       You can run this program standalone; it has one
*       self-contained test-case and it displays the result values to
*       the console as shown here:
*
*       gov.usps.USPS4CB usps = gov.usps.USPS4CB.getInstance();
*       usps.setTrack("01234567094987654321");
*       usps.setRoute("01234567891");
*       String result = usps.getBarCode();
*
*       The above code sequence would detect if the platform is
*       MS Windows and if it is loads usps4cb.dll. Alternatively,
*       it loads usps4cb.so included for the corresponding Mac,
*       Unix, Linux or other platforms. Each distribution contains
*       the library corresponding to the subject platform.
*       For convenience the MS Windows C-library, usps4cb.dll,
*       is included to facilitate cross-platform testing.
*
*       The usps4cb.jar file is included in the distribution and
*       can be used to import the class in your applications and to
*       invoke USPS4CB for testing by entering:
*
*       java -jar  usps4cb.jar  which is equivalent to
*       java -cp   usps4cb.jar  gov.usps.USPS4CB
*
*       The program looks for the platform dependent library,
*       usps4cb.so or usps4cb.dll, in the class-path to load it.
*       If it does not find it, it attempts to load it from the
*       jar-file. It is preferable that this library be placed
*       in the class-path of gov.usps.USPS4CB.class
*
*       A method, setLibrary( String ), is provided to
*       allow specific setting of the C-library should it be named
*       differently than usps4cb.so and/or reside on a path other
*       than the class-path.
*
*       The String parameter must have an absolute path followed
*       by the library name. The example above is equivalent to:
*
*       gov.usps.USPS4CB usps = gov.usps.USPS4CB.getInstance();
*       usps.setLibrary("/home/aDirectory/usps4cb.so");   load library
*       usps.setTrack( UPPSTrackNo );           set the tracking code
*       usps.setRoute( USPSRouteNo );            set the routing code
*       sr = usps.getBarCode();   obtain the resulting Intelligent Mail barcode
*
```

```
*          To call this class by an external program, call it as shown in
*          the following example:
*
*  package mytest;
*  import gov.usps.USPS4CB;
*
*  class AUspsTest {
*     public static void main(String[] args) throws Exception {
*
*         gov.usps.USPS4CB usps = gov.usps.USPS4CB.getInstance();
*         usps.setTrack("82205455868913559972");
*         usps.setRoute("54765515722");
*         String result = usps.getBarCode();
*
*         String[] parseResult = result.split("\\.");
*
*         System.out.println("\ntrackNo:\t"      + usps.getTrack());
*         System.out.println("routeNo:\t"        + usps.getRoute());
*
*         System.out.println("ReturnCode:\t"     + parseResult[0]);
*         System.out.println("EncodedBarCode:\t" + parseResult[1]);
*     }
*  }
*
*
*         To compile it, run:
*
*         javac -classptath .:usps4cb.jar mytest/AUspsTest.java
*
*         To invoke it, run
*
*         java  -cp  .:usps4cb.jar  mytest.AUspsTest
*
*         To call the compiled class in distributionFile.jar, run:
*
*         java -cp     usps4cb.jar  mytest.AUspsTest
*
*
*********************************************************************************/
```

6.2. Java Installation Verification Program IVP.java

A second Java Program is an example of calling the USPS C routine, `usps4cb.dll`, through the supplied `USPS4CB.java` class. The following are the comments extracted from the sample program; IVP.java:

```
import java.util.*;
import java.io.*;
import java.text.SimpleDateFormat;

import gov.usps.USPS4CB;

/**
 *@author     TPG Technology Consulting (TPG)
 *@created    January  06, 2006
 *@updated    February 10, 2008
 *
 *          This program is an example of calling the USPS Intelligent
 *          Mail barcode Encoder C-routine, as compiled for selected
 *          platforms in the form of a library such as usps4cb.so
 *          loaded and called through the supplied USPS4CB.java class.
 *
 *          See the source for USPS4CB.java file which is included in
 *          the download and serves also as an example to load and call
 *          the Encoder C-library.
 *
 *          The key lines are found below under the zTest() function.
 *          They are:
 *
 *          USPS4CB usps = gov.usps.USPS4CB.getInstance();  define USPS4CB
 *          usps.setTrack( UPPSTrackNo );           set the tracking code
 *          usps.setRoute( USPSRouteNo );            set the routing code
 *          sr = usps.getBarCode();   obtain the resulting Intelligent Mail barcode
 *
 *          The program runs 11-self-contained test cases and
 *          prints to the console the expected and obtained values.
 *
 *          Several command-line switches are provided:
 *
 *          -l n The [-l|-L] is used for forcing a loop-n-times
 *              This switch allows the IVP program to loop n-times
 *              calling the 11-self-contained test cases or processing
 *              the data from an input file; it defaults to -l 1
 *
 *          -f fileName switch reads the referenced file and
 *              processes included test cases.
 *
 *          -p 1 is the default; for a value other than "1" it suppresses
 *              printing the results of a test-case, eg: -p 0
```

```
 *
 *          -e 1 is the default; for a value other than "1" it suppresses
 *              calling the usps-encoder, eg. -e 0
 *
 *          -h switch prints a program usage line.
 *
*****************************************************************************/
```

----------END OF SAMPLE PROGRAMS ----------

Appendix A. Return Code Definitions

The encode function performs error checking and combines the two input strings in Tracking Code and Routing Code into a single string that is encoded into a 65-character Intelligent Mail barcode string. If valid input strings have been received, a Return Code of zero (0) is provided. However, if an error is detected a non-zero Return Code is issued. The following table details all of the possible Return Code values.

Preprocessor Definition	Return Code	Description
USPS_FSB_ENCODER_API_SUCCESS	0	Successful completion
USPS_FSB_ENCODER_API_SELFTEST_FAILED	1	Internal self-test failed
USPS_FSB_ENCODER_API_BAR_STRING_IS_NULL	2	Output barcode string is null
USPS_FSB_ENCODER_API_BYTE_CONVERSION_FAILED	3	Encoder byte conversion failed
USPS_FSB_ENCODER_API_RETRIEVE_TABLE_FAILED	4	Encoder retrieve table failed
USPS_FSB_ENCODER_API_CODEWORD_CONVERSION_FAILED	5	Encoder codeword conversion failed
USPS_FSB_ENCODER_API_CHARACTER_RANGE_ERROR	6	Encoder character range error
USPS_FSB_ENCODER_API_TRACK_STRING_IS_NULL	7	Input Tracking Code is null
USPS_FSB_ENCODER_API_ROUTE_STRING_IS_NULL	8	Input Routing Code is null
USPS_FSB_ENCODER_API_TRACK_STRING_BAD_LENGTH	9	Input Tracking Code must have 20 digits
USPS_FSB_ENCODER_API_TRACK_STRING_HAS_INVALID_DATA	10	Input Tracking Code must contain digits 0–9
USPS_FSB_ENCODER_API_TRACK_STRING_HAS_INVALID_DIGIT2	11	Input Tracking Code second digit must contain digits 0–4
USPS_FSB_ENCODER_API_ROUTE_STRING_BAD_LENGTH	12	Input Routing Code must be 0, 5, 9, or 11 digits in length
USPS_FSB_ENCODER_API_ROUTE_STRING_HAS_INVALID_DATA	13	Input Routing Code must contain digits 0–9

End of Document