

Table of Contents

Section 1: Introduction and Overview	3
<i>USPS® Address Matching System Developer's Kit.....</i>	<i>3</i>
<i>Address Matching System Technical Support.....</i>	<i>3</i>
<i>Installation Procedures for Windows (32 Bit).....</i>	<i>4</i>
<i>Installation Procedures for SUN UNIX (32 Bit).....</i>	<i>6</i>
<i>Installation Procedures for SUN UNIX (64 Bit).....</i>	<i>8</i>
<i>Installation Procedures for AIX UNIX.....</i>	<i>10</i>
<i>Installation Procedures for LINUX (32 Bit).....</i>	<i>12</i>
<i>Installation Procedures for LINUX (64 Bit).....</i>	<i>14</i>
Section 2: Coding Requirements	16
<i>Thread Safety.....</i>	<i>16</i>
<i>Process Safety.....</i>	<i>16</i>
<i>Stop Processing / False Positive Events.....</i>	<i>16</i>
<i>Function Call Order.....</i>	<i>17</i>
Section 3: API Functions.....	18
<i>Open the Address Matching System with Special Parameters.....</i>	<i>19</i>
<i>Address Inquiry.....</i>	<i>22</i>
<i>Address Sort Key.....</i>	<i>28</i>
<i>9-digit Inquiry.....</i>	<i>30</i>
<i>11-digit Inquiry.....</i>	<i>32</i>
<i>Address Standardization.....</i>	<i>34</i>
<i>Close the Address Matching System.....</i>	<i>36</i>
<i>Read City/State File By Key.....</i>	<i>37</i>
<i>Read City/State File Next.....</i>	<i>38</i>
<i>Read ZIP+4 File By Key.....</i>	<i>40</i>
<i>Read ZIP+4 File Next.....</i>	<i>41</i>
<i>Read ZIP+4 File Previous.....</i>	<i>43</i>
<i>Get ZIP Codes from a City/State.....</i>	<i>45</i>
<i>Terminate Active Address Inquiry.....</i>	<i>47</i>
<i>Get Date of ZIP+4 Database.....</i>	<i>48</i>
<i>Get AMS Data Expiration.....</i>	<i>50</i>
<i>Get AMS Library Expiration.....</i>	<i>52</i>

<i>Get API Code Version</i>	54
<i>Multiple Response Stack</i>	55
<i>Get Last Error</i>	57
<i>Get Environment</i>	59
<i>Retrieving the LACS^{Link®} Security Key</i>	60
<i>Checking for LACS^{Link} functionality</i>	63
<i>Disabling the LACS^{Link®} Security Key</i>	66
<i>SUITE^{LINK™} Database Date</i>	69
<i>SUITE^{LINK™} Error Code</i>	71
<i>SUITE^{LINK™} Error Message</i>	72
<i>SUITE^{LINK™} Query</i>	74
<i>Abbreviated Street Address Query</i>	76
Section 4: Footnote Flags	78
Section 5: Record Types	81
Section 6: Return Codes	82
Appendix A: Interface Definition	83
Appendix B: GDEV Application	94
Appendix C: DPV®	95
<i>Error Values</i>	95
<i>Error Codes</i>	95
<i>Database Tables</i>	96
<i>Database Table Options</i>	96
<i>Data Types</i>	97
<i>Interface Overview</i>	98
<i>Notes</i>	102
<i>DPV® Sample</i>	103

Section 1: Introduction and Overview

The USPS® *Address Matching System Application Programming Interface User Guide* is the primary reference document for the USPS National Customer Support Center's Address Matching System product. The guide contains installation instructions for each platform as well as function descriptions.

The USPS® Address Matching System (AMS) is an application programming interface (API). As such, this guide should be used when the user wants to interface an application with the Address Matching System.

USPS® Address Matching System Developer's Kit

The USPS Address Matching System Developer's Kit contains the following:

- API library(s) for each specific computer platform
- Interface definition file (ZIP4.H)
- Test utility (SAMPLE.EXE)
- Test utility source code
- User documentation

The test utility can be used to ensure that the Address Matching System and data files have been installed correctly and to provide access to our matching logic, which displays the standardized address returned by the matching engine. This enables you to verify the accuracy of the ZIP+4 results returned from your product.

The AMS software, including, but not limited to, .DLLs, shared objects and static objects all expire and cease functionality based on USPS Coding Accuracy Support System (CASS™) guidelines. The AMS software expires July 31st each year. The AMS data expires 105 days from the release date of the DVD, which is the 15th day of each month.

During your development cycle and subsequent updates of your software, you should compile your software with the AMS library. The AMS library will handle any necessary interface with the DPV® library, the Suite^{Link}® library and the KeyManager library.

Address Matching System Technical Support

If there are any questions regarding the Address Matching System API, please call the USPS' National Customer Support Center, Address Matching System Technical Support at 1-877-640-0724. Hours of operation are 7am to 5pm Monday through Friday CST.

Installation Procedures for Windows (32 Bit)

1. Create a directory on your hard drive in which to store the API files.

Ex: MD C:\AMS

2. Copy the Address Matching System files to your hard drive.

The AMS Product is distributed on DVD. The AMS files are located in the corresponding directory below:

```
[DVD] \<PRODUCT TYPE>\dev_kits\w32\
```

All of the files in this directory are encrypted and must be unencrypted before use.

There are two (2) utility programs on the DVD that will unencrypt files.

- a. GDEV – See Appendix B for description and use
- b. dev_w32.exe located in the dev_kits directory

Ex: DEV_W32 CUST_ID OUTPUT_PATH PRODUCT_FILE

- i. OUTPUT_PATH is the directory created in step 1.
- ii. PRODUCT_FILE is the file from the list in step 6. This should not include any directory paths.

The installation program must be executed from within the DVD directory. This step needs to be performed once for each file listed in the file description in step 6.

Note: A customer ID (CUST_ID) should be obtained from Address Matching System Technical Support. The customer ID must be entered in uppercase letters. The customer ID provided by Address Matching System Technical Support will change each month. We do not recommend hard-coding the customer ID into an install program. For program installation, you may obtain a unique customer ID from Address Matching System Technical Support. This unique customer ID will not change for the duration of the AMS API license unless otherwise specified.

3. Run SAMPLE.EXE to test AMS.

Select the option to manually enter the paths.

4. Use SAMPLE.C as an example to create your own API application.
5. Refer to Section 3, API Functions, to test other API function commands.
6. The following is an explanation of the API files for W32:

- | | |
|-----------------|--|
| a. ZIP4_W32.DLL | ZIP4 dynamic-link library |
| b. ZIP4_W32.LIB | Stub library to link with the user application |
| c. ZIP4.H | Interface header file |
| d. Z4CONFIG.DAT | File location file |
| e. Z4CXLOG.DAT | Date time file |

- | | | |
|----|-------------|----------------------|
| f. | SAMPLE.C | Sample C source file |
| g. | SAMPLE.H | Sample header file |
| h. | SAMPLE.EXE | Sample executable |
| i. | KEYMGR3.DLL | Key manager dll |

Special Notes for Windows (32 Bit)

The Windows 32-bit version of the Address Matching System DLL was built with all export functions having the ‘_cdecl’ calling convention, which has caused problems with some programming languages that do not support this convention. To provide access to the address matching routines in the DLL for non C and C++ languages, the DLL also contains a set of routines with the proper DLL calling convention ‘_stdcall.’ These routines have separate names from the original routines to preserve linkage with existing programs, and the new names are a concatenation of the original function name and ‘STD,’ which implies the _stdcall calling convention, e.g.,

_cdecl function name	_stdcall function name
z4opencfg()	z4opencfgSTD()
z4adring()	z4adringSTD()
z4close()	z4closeSTD()

All of the _stdcall functions map directly to the original functions, so there is no loss in functionality. All existing functions have an associated _stdcall version, and all future additions to the DLL will contain both a _cdecl version and a _stdcall version.

Installation Procedures for SUN UNIX (32 Bit)

1. Create a directory on your hard drive in which to store the API files.

Ex: `mkdir /usr/src/ams`

2. Copy the Address Matching System files to your hard drive.

The AMS product is distributed on DVD. The AMS files are located in the corresponding directory below:

[DVD] /<PRODUCT TYPE>/dev_kits/sun/

All of the files in this directory are encrypted and must be unencrypted before use.

There are two (2) utility programs on the DVD that will unencrypt the files.

- a. GDEV – See Appendix B for description and use.
- b. dev_sun.exe is located in the dev_kits directory

Ex: `DEV_SUN.EXE CUST_ID OUTPUT_PATH PRODUCT_FILE`

- i. OUTPUT_PATH is the directory created in step 1.
- ii. PRODUCT_FILE is a file from the list in step 6. This should not include any directory paths.

The installation program must be executed from within the DVD directory. This step needs to be performed once for each file listed in the file description in step 6 on the next page.

Note: *A customer ID (CUST_ID) should be obtained from Address Matching System Technical Support. The customer ID must be entered in uppercase letters.*

The customer ID provided by Address Matching System Technical Support will change each month. We do not recommend hard-coding the customer ID into an install program. For program installation, you may obtain a unique customer ID from Address Matching System Technical Support. This unique customer ID will not change for the duration of the AMS API license unless otherwise specified.

3. Run SAMPLESH and SAMPLEST to test the Address Matching System.
 - a. CHMOD on SAMPLESH and SAMPLEST to rwx.
 - b. CHMOD on Z4CXLOG.DAT to rw.
 - c. Select the option to manually enter the paths
4. Use SAMPLE.C as an example to create your own API application.
5. Refer to Section 3, API Functions, to test other API function calls.
6. The following is an explanation of the API files for SUN UNIX:
 - a. LIBZ4SUN.SO ZIP4 shared library
 - b. ZIP4_SUN.A Static link library; not recommended
 - c. ZIP4.H Interface header file
 - d. Z4CONFIG.DAT File location file
 - e. Z4CXLOG.DAT Date time file

- f. SAMPLE.C Sample C source file
- g. SAMPLE.H Sample header file
- h. SAMPLESH Sample executable linked with LIBZ4SUN.SO
- i. SAMPLEST Sample executable built with ZIP4_SUN.A
- j. LIBKEYMGR.SO.3 Key manager shared library

Special Notes for SUN UNIX (32 Bit)

The Address Matching System DVD uses the ISO9660 file-system format, which stores file names in uppercase letters with a version control number appended to the end. The API requires that the DVD file names appear in lowercase letters without the version number. Some versions of UNIX will automatically accommodate file name conversion during the mount process, but some require the user to specify the conversion explicitly with the options of the “mount” command. Please see the **man** pages on mount for more information on these options.

The Address Matching System SUN API Developer’s Kit contains both a static-link and a shared library. The static-link library is provided for compatibility with older programs written before the shared library was available. The USPS does not recommend use of the static-link library because logic changes are often made to the API, and the user would have to re-link the executable file with the AMS static-link library every time there is an update. Also, in compliance with CASS rules, the API code is set to expire at the end of the current CASS cycle, each August. If this date is reached without re-linking with a newer API, a user’s application will stop functioning.

To avoid these problems the USPS recommends using the AMS shared library so that user applications can gain immediate access to any logic changes simply by installing the new shared library. User applications do not need to be re-linked when a new shared library is provided on DVD updates.

Installation Procedures for SUN UNIX (64 Bit)

1. Create a directory on your hard drive in which to store the API files.
Ex. `mkdir /usr/src/ams`
2. Copy the Address Matching System files to your hard drive.

The AMS product is distributed on DVD. The AMS files are located in the corresponding directory below:

```
[DVD] /<PRODUCT TYPE>/dev_kits/sun/
```

All of the files in this directory are encrypted and must be unencrypted before use.

There are two (2) utility programs on the DVD that will unencrypt the files.

- a. GDEV – See Appendix B for description and use.
- b. `dev_s64.exe` is located in the `dev_kits` directory

```
Ex: DEV_S64.EXE CUST_ID OUTPUT_PATH PRODUCT_FILE
```

- i. `OUTPUT_PATH` is the directory created in step 1.
- ii. `PRODUCT_FILE` is a file from the list in step 6. This should not include any directory paths.

The installation program must be executed from within the DVD directory. This step needs to be performed once for each file listed in the file description in step 6 on the next page.

Note: A customer ID (`CUST_ID`) should be obtained from Address Matching System Technical Support. The customer ID must be entered in uppercase letters.

The customer ID provided by Address Matching System Technical Support will change each month. We do not recommend hard-coding the customer ID into an install program. For program installation, you may obtain a unique customer ID from Address Matching System Technical Support. This unique customer ID will not change for the duration of the AMS API license unless otherwise specified.

3. Run `SAMPLESH` to test the Address Matching System.
 - a. `CHMOD` on `SAMPLESH` to `rw`.
 - b. `CHMOD` on `Z4CXLOG.DAT` to `rw`.
 - c. Select the option to manually enter the paths
4. Use `SAMPLE.C` as an example to create your own API application.
5. Refer to Section 3, API Functions, to test other API function calls.
6. The following is an explanation of the API files for SUN UNIX:
 - a. `LIBZ4SUN64.SO` ZIP4 shared library
 - b. `ZIP4.H` Interface header file
 - c. `Z4CONFIG.DAT` File location file
 - d. `Z4CXLOG.DAT` Date time file
 - e. `SAMPLE.C` Sample C source file
 - f. `SAMPLE.H` Sample header file
 - g. `SAMPLESH` Sample executable linked with `LIBZ4SUN.SO`

- h. LIBKEYMGR.SO.3 Key manager shared library

Special Notes for SUN UNIX (64 Bit)

The Address Matching System DVD uses the ISO9660 file-system format, which stores file names in uppercase letters with a version control number appended to the end. The API requires that the DVD file names appear in lowercase letters without the version number. Some versions of UNIX will automatically accommodate file name conversion during the mount process, but some require the user to specify the conversion explicitly with the options of the “mount” command. Please see the **man** pages on mount for more information on these options.

The Address Matching System S64 API Developer’s Kit contains a shared library. In compliance with CASS rules, the API code is set to expire at the end of the current CASS cycle, each August. If this date is reached without replacing the shared library, a user’s application will stop functioning.

Installation Procedures for AIX UNIX

- 1 Create a directory on your hard drive in which to store the API files.

Ex. `mkdir /usr/src/ams`

- 2 Copy the Address Matching System files to your hard drive.

The AMS product is distributed on DVD. The AMS files are located in the corresponding directory below:

[DVD] /<PRODUCT TYPE>/dev_kits/aix/

All of the files in this directory are encrypted and must be unencrypted before use.

There are two (2) utility programs on the DVD that will unencrypt the files.

- a. GDEV – See Appendix B for description and use.
- b. `dev_aix.exe` located in the `dev_kits` directory

Ex. `DEV_AIX.EXE CUST_ID OUTPUT_PATH PRODUCT_FILE`

- i. `OUTPUT_PATH` is the directory created in step 1.
- ii. `PRODUCT_FILE` is a file from the list in step 6. This should not include any directory paths

The installation program must be executed from within the DVD directory. This step needs to be performed once for each file listed in the file description in step 6 on the next page.

Note: *A customer ID (CUST_ID) should be obtained from Address Matching System Technical Support. The customer ID must be entered in uppercase letters.*

The customer ID provided by Address Matching System Technical Support will change each month. We do not recommend hard-coding the customer ID into an install program. For program installation, you may obtain a unique customer ID from Address Matching System Technical Support. This unique customer ID will not change for the duration of the AMS API license unless otherwise specified.

- 3 Run `SAMPLEST` to test Address Matching System.
 - a. `CHMOD` on `SAMPLEST` to `rwx`.
 - b. `CHMOD` on `Z4CXLOG.DAT` to `rw`.
 - c. Select the option to manually enter the paths.
- 4 Use `SAMPLE.C` as an example to create your own API application.
- 5 Refer to Section 3, API Functions, to test other API function commands.
- 6 The following is an explanation of the API files for AIX UNIX:
 - a. `ZIP4_AIX.A` Static-link library
 - b. `ZIP4.H` Interface header file
 - c. `Z4CONFIG.DAT` File location file [Deprecated]
 - d. `Z4CXLOG.DAT` Date time file
 - e. `SAMPLE.C` Sample C source file
 - f. `SAMPLE.H` Sample header file

- g. SAMPLEST Sample executable built with ZIP4_AIX.A

Special Notes for AIX UNIX

The Address Matching System DVD uses the ISO9660 file-system format, which stores file names in uppercase letters with a version control number appended to the end. However, the API requires that the DVD file names appear in lowercase letters without the version number. Some versions of UNIX will automatically accommodate file-name conversion during the mount process, but some require the user to specify the conversion explicitly with the options of the “mount” command. Please see the **man** pages on mount for more information on these options.

Installation Procedures for LINUX (32 Bit)

1. Create a directory on your hard drive in which to store the API files.
Ex. `mkdir /usr/src/ams`
2. Copy the Address Matching System files to your hard drive.

The AMS product is distributed on DVD. The AMS files are located in the corresponding directory below:

```
[DVD] /<PRODUCT TYPE>/dev_kits/lnx/
```

All of the files in this directory are encrypted and must be unencrypted before use.

There are two (2) utility programs on the DVD that will unencrypt the files.

- a. GDEV – See Appendix B for description and use
- b. `dev_lnx.exe` located in the `dev_kits` directory

```
Ex. DEV_LNX.EXE CUST_ID OUTPUT_PATH PRODUCT_FILE
```

- i. OUTPUT_PATH is the directory created in step 1.
- ii. PRODUCT_FILE is a file from the list in step 6. This should not include any directory paths.

The installation program must be executed from within the DVD directory. This step needs to be performed once for each file listed in the file description in step 6 on the next page. Following initial installation, the only files that need to be installed with subsequent DVD updates are the header files and libraries. A batch file is recommended to simplify this install process.

Note: *A customer ID (CUST_ID) should be obtained from Address Matching System Technical Support. The customer ID must be entered in uppercase letters.*

The customer ID provided by Address Matching System Technical Support will change each month. We do not recommend hard-coding the customer ID into an install program. For program installation, you may obtain a unique customer ID from Address Matching System Technical Support. This unique customer ID will not change for the duration of the AMS API license unless otherwise specified.

3. Run SAMPLESH and SAMPLEST to test Address Matching System.
 - a. CHMOD on SAMPLESH and SAMPLEST to `rw`.
 - b. CHMOD on Z4CXLOG.DAT to `rw`.
 - c. Select the option to manually enter the paths
4. Use SAMPLE.C as an example to create your own API application.
5. Refer to Section 3, API Functions, to test other API function commands.
6. The following is an explanation of the API files for LNX:
 - a. LIBZ4LNX.SO ZIP4 shared library
 - b. ZIP4_LNX.A Static link library; not recommended
 - c. ZIP4.H Interface header file
 - d. Z4CONFIG.DAT File location file
 - e. Z4CXLOG.DAT Date time file

- f. SAMPLE.C Sample C source file
- g. SAMPLE.H Sample header file
- h. SAMPLESH Sample executable linked with LIBZ4LNX.SO
- i. SAMPLEST Sample executable built with ZIP4_LNX.A
- j. LIBKEYMGR.SO.3 Key manager shared library

Special Notes for LINUX (32 Bit)

The Address Matching System DVD uses the ISO9660 file-system format, which stores file names in uppercase letters with a version control number appended to the end. However, the API requires that the DVD file names appear in lowercase letters without the version number. Some versions of UNIX will automatically accommodate file-name conversion during the mount process, but some require the user to specify the conversion explicitly with the options of the “mount” command. Please see the **man** pages on mount for more information on these options.

The Address Matching System LINUX API Developer’s Kit contains both a static-link and a shared library. The static-link library is provided for compatibility with older programs written before the shared library was available. The USPS does not recommend use of the static-link library because logic changes are often made to the API, and the user would have to re-link the executable files with the AMS staticlink library every time there is an update. Also, in compliance with CASS rules, the API code is set to expire at the end of the current CASS cycle, each August. If this date is reached without re-linking with a newer API, a user’s application will stop functioning.

To avoid these problems, the USPS recommends using the AMS shared library so that user applications can gain immediate access to any logic changes simply by installing the new shared library. User applications do not need to be re-linked when a new shared library is provided on DVD updates.

Installation Procedures for LINUX (64 Bit)

1. Create a directory on your hard drive in which to store the API files.
Ex. `mkdir /usr/src/ams`
2. Copy the Address Matching System files to your hard drive.

The AMS product is distributed on DVD. The AMS files are located in the corresponding directory below:

```
[DVD] /<PRODUCT TYPE>/dev_kits/164/
```

All of the files in this directory are encrypted and must be unencrypted before use.

There are two (2) utility programs on the DVD that will unencrypt the files.

- a. GDEV – See Appendix B for description and use
- b. `dev_164.exe` located in the `dev_kits` directory

```
Ex. DEV_L64.EXE CUST_ID OUTPUT_PATH PRODUCT_FILE
```

- i. OUTPUT_PATH is the directory created in step 1.
- ii. PRODUCT_FILE is a file from the list in step 6. This should not include any directory paths.

The installation program must be executed from within the DVD directory. This step needs to be performed once for each file listed in the file description in step 6 on the next page. Following initial installation, the only files that need to be installed with subsequent DVD updates are the header files and libraries. A batch file is recommended to simplify this install process.

Note: *A customer ID (CUST_ID) should be obtained from Address Matching System Technical Support. The customer ID must be entered in uppercase letters.*

The customer ID provided by Address Matching System Technical Support will change each month. We do not recommend hard-coding the customer ID into an install program. For program installation, you may obtain a unique customer ID from Address Matching System Technical Support. This unique customer ID will not change for the duration of the AMS API license unless otherwise specified.

3. Run SAMPLESH and SAMPLEST to test Address Matching System.
 - a. CHMOD on SAMPLESH and SAMPLEST to `rxw`.
 - b. CHMOD on Z4CXLOG.DAT to `rw`.
 - c. Select the option to manually enter the paths
4. Use SAMPLE.C as an example to create your own API application.
5. Refer to Section 3, API Functions, to test other API function commands.
6. The following is an explanation of the API files for LNX:
 - a. LIBZ4LNX64.SO ZIP4 shared library
 - b. ZIP4_LNX64.A Static link library; not recommended
 - c. ZIP4.H Interface header file
 - d. Z4CONFIG.DAT File location file
 - e. Z4CXLOG.DAT Date time file

- f. SAMPLE.C Sample C source file
- g. SAMPLE.H Sample header file
- h. SAMPLESH Sample executable linked with LIBZ4LNX.SO
- i. SAMPLEST Sample executable built with ZIP4_LNX.A
- j. LIBKEYMGR.SO.3 Key manager shared library

Special Notes for LINUX (64 Bit)

The Address Matching System DVD uses the ISO9660 file-system format, which stores file names in uppercase letters with a version control number appended to the end. However, the API requires that the DVD file names appear in lowercase letters without the version number. Some versions of UNIX will automatically accommodate file-name conversion during the mount process, but some require the user to specify the conversion explicitly with the options of the “mount” command. Please see the **man** pages on mount for more information on these options.

The Address Matching System LINUX 64 API Developer’s Kit contains both a static-link and a shared library. The static-link library is provided for compatibility with older programs written before the shared library was available. The USPS does not recommend use of the static-link library because logic changes are often made to the API, and the user would have to re-link the executable files with the AMS static-link library every time there is an update. Also, in compliance with CASS rules, the API code is set to expire at the end of the current CASS cycle, each August. If this date is reached without re-linking with a newer API, a user’s application will stop functioning.

To avoid these problems, the USPS recommends using the AMS shared library so that user applications can gain immediate access to any logic changes simply by installing the new shared library. User applications do not need to be re-linked when a new shared library is provided on DVD updates.

Section 2: Coding Requirements

Thread Safety

The Address Matching System library is not thread safe and its function calls must be protected if it is to be used in a multi-threaded application.

Your software will need to implement controls to ensure that the input is queued and submitted to the AMS library one at a time.

Process Safety

The Address Matching System library can be used in multiple processes, but each process must have a z4cxlog.dat file that is dedicated for its sole use.

The AMS library performs write operations to the z4cxlog.dat file and it is possible that the file can become corrupt if it is used by multiple processes at the same time.

Stop Processing / False Positive Events

Your software is required to handle false positive events which cause the AMS library to shut down (stop processing).

This requirement is not unique to the AMS library, it is a requirement that has been placed upon all address matching software that uses the USPS® DPV® and/or LACS^{Link}® data products.

False positive detection is a security measure embedded in the DPV and LACS^{Link} sub-systems of the AMS library. This security measure is designed to prevent the artificial creation of an address list by detecting when a submitted address appears to have been constructed artificially and not obtained legitimately. This should be a very rare occurrence, but when this security measure is tripped the AMS library will shut down and it will no longer process addresses until the reporting requirement has been completed

The following steps provide an overview of how to comply with this requirement:

1. Determine when a false positive event occurs.
Immediately after all `z4adrinq()` function calls your software will need to check the DPV and LACS^{Link} sub-systems to see if a false positive event occurred.
DPV sub-system : call `z4DpvGetCode(HSF_DPV)` and check for the value 'Y'
LACS^{Link} sub-system : check the `llk_ind` variable of the `ZIP4_PARM` structure and check for the value 'F'
2. If a false positive event was not detected, skip the following steps.
3. Gather the required information.
This includes the submitted address that tripped the false positive and information on the customer that submitted the address.

This information must be written to a file in a specific format. For details see pages 12 and 13 in the document http://ribbs.usps.gov/dpv/documents/tech_guides/DPV_LPR.PDF
4. Depending on which sub-system reported the event, call the appropriate function to get the shut-down key.
DPV sub-system : `z4DpvGetKey()`
LACS^{Link} sub-system : `z4LLkGetKey()`
5. Send an email to the DSF2STOP@USPS.COM email address informing them that your software has hit a false positive address. Include the information from step 3 (as a file attachment) and step 4 above in this email.

This email will be evaluated by the USPS licensing dept. and they will provide further guidance after reviewing the information.

6. You will be provided a re-activation key after step 5 has been completed and approved.
7. You will need to pass the re-activation key(s) in as a parameter to the `z4DpvSetKey()` or the `z4LlkSetKey()` function depending upon which sub-system reported the event.
7. Call the `z4opencfg()` function to open the AMS library.

In addition, it is also required that the following text be included in all end-user documentation to describe this error.

AMS, DPV, LACSLink and SuiteLink API processing was terminated due to the detection of what is determined to be an artificially created address. No address beyond this point has been validated and/or processed. In accordance with the Agreement between the Licensor and the Licensee, AMS, DPV, LACSLink and SuiteLink API shall be used to validate legitimately obtained addresses only, and shall not be used for the purpose of artificially creating address lists. The written agreement between the Licensee and the End User shall also include this same restriction against using AMS, DPV, LACSLink and SuiteLink API to artificially create address lists. Continuing use of AMS, DPV, LACSLink and SuiteLink API requires compliance with all terms of the Agreement. If you believe this address was identified in error, please contact your Vendor.

Function Call Order

The AMS library provides your software with the ability to process addresses in compliance with CASS™ requirements. In order to accurately comply with these requirements it is recommended that you use the following steps when submitting addresses to the AMS library.

1. Initially submit the address to the `z4adring()` function call.
2. If a multiple response is returned, then call the `z4DpvResolveMultiResp()` function to attempt to resolve it to a single match.
3. If a default response is returned, then call the `z4SLNKQuery()` function to attempt to resolve it to an exact match.
4. Finally, if you wish to obtain an abbreviated version of the street name, then call the `z4ABSQuery()` function.

Note: Step 4 is required when your software is processing a CASS test, but it is optional at all other times.

Example code snippet for the steps above:

```
z4adring(pZip4);  
  
if(pZip4->retcc == Z4_MULTIPLE)  
    z4DpvResolveMultiResp(pZip4);  
  
if(pZip4->retcc == Z4_DEFAULT)  
    z4SLNKQuery(pZip4);  
  
if(pZip4->retcc > Z4_MULTIPLE)  
    z4ABSQuery(pZip4, pAbbr);
```

Section 3: API Functions

The following functions are used to perform inquiries on addresses and 9-digit ZIP Codes:

- `z4opencfg()` Open the Address Matching System with Special Parameters
- `z4adrinq()` Address Inquiry
- `z4adrkey()` Address Sort Key
- `z4xrfinq()` 9-digit Inquiry
- `z4xrfinq11()` 11-digit Inquiry
- `z4adrstd()` Address Standardization
- `z4close()` Close the Address Matching System
- `z4ctyget()` Read City/State File by Key
- `z4ctynxt()` Read City/State File Next
- `z4adrget()` Read ZIP+4 File by Key
- `z4adrnxt()` Read ZIP+4 File Next
- `z4adrprv()` Read ZIP+4 File Previous
- `z4getzip()` Get a ZIP Code range for a City/St
- `z4abort()` Terminate Active Address Inquiry
- `z4date()` Get Date of ZIP+4 Database
- `z4GetDataExpireDays()` Get AMS Data Expiration
- `z4GetCodeExpireDays()` Get AMS Library Expiration
- `z4ver()` Get the Version of the API code
- `z4scroll()` Multiple Response Stack
- `z4geterror()` Get Last Error
- `z4getenv()` Get Environment
- `z4LLkGetKey()` Retrieving the LACS^{Link} Security Key
- `z4LLkIsDisabled()` Checking for LACS^{Link} Functionality
- `z4LLkSetKey()` Disabling the LACS^{Link} Security Key
- `z4SLNKGetDate()` Provides the date associated with a table
- `z4SLNKGetError()` Provides the error integer status of Suite^{Link}
- `z4SLNKGetErrorMsg()` Provides the error string status of Suite^{Link}
- `z4SLNKQuery()` Performs a Suite^{Link} lookup
- `z4ABSQuery()` Performs an abbreviated street address lookup

Open the Address Matching System with Special Parameters

The `z4opencfg()` function opens the Address Matching System for application use. This function must be called before attempting to use any of the inquiry functions. During system opening, the Address Matching System allocates memory buffers and file handles for disk I/O. The function returns a code summarizing the results of the open operation.

Note: *The DPV® and LACS^{Link}® components are no longer optional and must always be enabled. While the Z4OPEN_PARM still contains the `llkflag` and `dpvflag` variables, they no longer provide any functionality and are ignored by the AMS library. eLOT® is available through the USPS AMS API, but it is turned off by default. To enable eLOT® processing, you must first call `z4opencfg()` and set the `elotflag` variable to 'Y'. You must also use the `CONFIG_PARM` to specify the paths to the AMS database.*

Syntax

```
#include "zip4.h"
int z4opencfg(Z4OPEN_PARM* openparm);
```

Input

`openparm` A pointer to a Z4OPEN_PARM structure where the output will be stored.

If a field in the Z4OPEN_PARM is not used, then it must be initialized to NULL/zero (see example code).

```
typedef struct
{
    char          rsvd1[50];
    short         status;
    char*         fname;

    CONFIG_PARM  config;

    char          ewsflag;
    char          elotflag;
    char          llkflag;
    char          dpvflag;
    char          systemflag;
    char          rtsw[16];
    char          dpvtypeflag;
    char          stelkflag;
    char          abrstflag;
    char          rsvd2[492];
}Z4OPEN_PARM;
```

Field Definitions:

rsvd1 Reserved for future use.
 status See "Output" section.
 fname Pointer to a string that contains the full path and filename for a custom config file. [Deprecated]
 config Embedded structure for setting the path names to the AMS database. (Not used if *fname* is set)
 ewsflag Set to 'Y' to activate EWS processing
 elotflag Set to 'Y' to activate eLOT processing.
 llkflag Usage has been discontinued
 dpvflag Usage has been discontinued
 systemflag Set to 'Y' to de-activate the auto-generation of the security file.
 rsvd2 Reserved for future use.
 rtsw Reserved for future use.
 dpvtypeflag Reserved for future use
 stelnkflag Set to 'Y' to activate Suite^{Link} processing
 abrstflag Set to 'Y' to activate abbreviated street name processing

Output

Z4OPEN_PARM.status will be set to 1, 2 or 9 to indicate which value was used for the configuration file.

<u>Name</u>	<u>Value</u>	<u>Meaning</u>
Z4_FNAME	1	Used the value pointed to by the <i>fname</i> character pointer
Z4_CONFIG	2	Used the values pointed to by the CONFIG_PARM structure
Z4_SEARCH	9	Searched for a file named z4config.dat

Return

- 1 The USPS Address Matching System is already open
- 0 The USPS Address Matching System opened successfully
- 1 The USPS Address Matching System is not in sync
- 2 The USPS Address Matching System has expired
- 4 The USPS Address Matching System failed to open DPV
- 5 The USPS Address Matching System failed to open DPV
- 7 The USPS Address Matching System failed to open LACS^{Link}
- 13 The USPS Address Matching System failed to open Suite^{Link}
- 17 The USPS Address Matching System failed to open Abbreviated street name

Note: See the DPV User Guide for specific information on DPV errors.

Example

```

#include <stdio.h>
#include "zip4.h"

void main(void)
{
    Z4OPEN_PARM openparm;
    int rtn=0;

    memset(&openparm, 0, sizeof(openparm));

    /*Open with the paths embedded in the CONFIG_PARM structure*/
    openparm.config.address1 = "c:\\amsdata\\";
    openparm.config.addrindex = "c:\\amsdata\\";
    openparm.config.cdrom = "d:\\";
    openparm.config.citystate = "c:\\amsdata\\";
    openparm.config.crossref = "c:\\amsdata\\";
    openparm.config.system = "c:\\amsdata\\";
    openparm.config.elot = "c:\\elotdata\\";
    openparm.config.elotindex = "c:\\elotdta\\";
    openparm.config.llkpath = "c:\\llkdata\\";
    openparm.config.dpvpath = "c:\\dpvdata\\";
    openparm.config.fnsnpath = "c:\\amsdata\\";
    openparm.config.stelnkpath = "c:\\slkdata\\";
    openparm.config.abrstpath = "c:\\abrstdata\\";

    /*Turn eLOT processing on*/
    openparm.elotflag = 'Y';

    rtn = z4opencfg(&openparm);

    if(rtn==0)
        printf("\nSuccess opening the USPS Address Matching System.");
    else
        printf("\nError opening the USPS Address Matching System.");

    /*close the USPS Address Matching System*/
    z4close();
}

```

Address Inquiry

The `z4adrinq()` function commands the Address Matching System to perform an address inquiry using firm name (optional), address, and city/state/ZIP information. Before performing this function, the input address information must be copied into the corresponding input fields outlined below. Note that the City, State, and ZIP fields may be placed either within the `parm. ictyi` field or copied to the `parm. ictyi`, `parm. stai`, and `parm. izipc` fields, respectively. Following the address inquiry, the `parm. retcc` field contains a response code summarizing the inquiry results. If an address response was found, standardized address information will be located in the output fields described below.

Syntax

```
#include "zip4.h"
int z4adrinq(ZIP4_PARM* parm);
```

Input

`parm` A pointer to a `ZIP4_PARM` structure that provides the input and where the output will be stored.

The following fields must be initialized before calling the `z4adrinq()` function. If a field is not used, it must be initialized to zero.

<code>parm.iadl1</code>	Street Address
<code>parm.iadl2</code>	Firm Name
<code>parm.iadl3</code>	Secondary Address
<code>parm.iprurb</code>	Puerto Rican Urbanization Name
<code>parm.ictyi</code>	City or City/State/ZIP
<code>parm.istai</code>	State or empty
<code>parm.izipc</code>	ZIP or empty
<code>parm.iddpv11</code>	Future Use

Output

parm.retcc	Response Code
<code>Z4_INVADDR</code>	10 — Invalid input address (i.e., contained a dual address)
<code>Z4_INVZIP</code>	11 — Invalid input 5-digit ZIP Code
<code>Z4_INVSTATE</code>	12 — Invalid input state abbreviation code
<code>Z4_INVCITY</code>	13 — Invalid input city name
<code>Z4_NOTFND</code>	21 — No match found using input address
<code>Z4_MULTIPLE</code>	22 — Multiple responses were found and more specific information is required to select a single or default response
<code>Z4_SINGLE</code>	31 — A single address was found
<code>Z4_DEFAULT</code>	32 — An address was found, but a more specific address could be found with more information

parm.foot

parm.foot.a = "A"
 parm.foot.b = "B"
 parm.foot.c = "C"
 parm.foot.d = "D"
 parm.foot.e = "E"
 parm.foot.f = "F"
 parm.foot.g = "G"
 parm.foot.h = "H"
 parm.foot.i = "I"
 parm.foot.j = "J"
 parm.foot.k = "K"
 parm.foot.l = "L"
 parm.foot.m = "M"
 parm.foot.n = "N"
 parm.foot.o = "O"
 parm.foot.p = "P"
 parm.foot.q = "Q"
 parm.foot.r = "R"
 parm.foot.s = "S"
 parm.foot.t = "T"
 parm.foot.u = "U"
 parm.foot.v = "V"
 parm.foot.w = "W"
 parm.foot.x = "X"
 parm.foot.y = "Y"
 parm.foot.z = "Z"

parm.stelnkfoot = "00"
 = "A "
 = ""

Footnotes

ZIP Code Corrected
 City/State Corrected
 Invalid City/State/ZIP
 No ZIP+4 Code Assigned
 ZIP Code Assigned with a Multiple Response
 Address Not Found
 All or Part of the Firm Line Used For Address Line
 Missing Secondary Number
 Insufficient/Incorrect Data
 PO Box Dual Address
 Non-PO Box Dual Address
 Address Component Changed
 Street Name Changed
 Address Standardized
 Multiple response can be broken using the lowest +4
 Better Address Exists
 Unique ZIP Code Match
 No Match due to EWS
 Incorrect Secondary Number
 Multiple response due to Magnet Street Syndrome
 Unofficial Post Office Name
 Unverifiable City/State
 Small Town Default
 Unique ZIP Code Default
 Military Match
 ZIP Move Match

Suite^{Link} no match
 Suite^{Link} match
 Suite^{Link} did not attempt a lookup

Return Address	Description
parm.dad11	Standardized Output Address
parm.dad12	Standardized Output Firm Name
parm.dad13	Standardized Secondary Address
parm.dprurb	Standardized Puerto Rican Urbanization Name
parm.dctya	Standardized Output City
parm.dstaa	Standardized Output State
parm.dlast	Standardized Output City, State, and ZIP
parm.dctys	Main Post Office Output City
parm.dstas	Main Post Office Output State
parm.abcty	Abbreviated Output City
parm.zipc	5-digit ZIP Code
parm.addon	4-digit Add-on Code
parm.cris	4-digit Carrier Route Code
parm.county	3-digit County Code
parm.dpbc	2-digit Delivery Point Barcode and 1-digit Check Digit
parm.mpnum	Matched Primary Number
parm.msnum	Matched Secondary Number
parm.auto_zone_ind	Carrier Route Rate Sort Indicator (Y or N)
parm.elot_num	Enhanced Line of Travel (eLOT) number
parm.elot_code	eLOT Ascending/Descending Flag (A/D)
parm.llk_rc	LACS ^{Link} Return Code
parm.llk_ind	LACS ^{Link} Indicator
parm.respn	Number of Response Records Returned
parm.retcc	Lookup Status
parm.adrkey	Address Database Key (Binary field)
parm.misc	unused input data

Parsed Input	Description
ppnum	Primary Number
psnum	Secondary Number
psnum2	Second or Right Secondary Number
prote	Rural Route Number
punit	Secondary Number Unit
punit2	Secondary or Right Secondary Number Unit
ppre1	First or Left Pre-direction
ppre2	Second or Right Pre-direction
psuf1	First or Left Suffix
psuf2	Second or Right Suffix
ppst1	First or Left Post-direction
ppst2	Second or Right Post-direction
ppnam	Primary Name
mpnum	Matched Primary Number
msnum	Matched Secondary Number
pmb	PMB Unit Designator
pmbnum	PMB Number

Return

- 0 - The USPS Address Matching System resident
- 1 - The USPS Address Matching System issued a system error
- 2 - The USPS Address Matching System not ready
- 3 - DVD has expired

Additional Information About Z4ADRINQ()

If parm.retcc is Z4_INVADDR, Z4_INVZIP, Z4_INVSTATE, Z4_INVCITY, Z4_NOTFND, or Z4_MULTIPLE, then the return address fields will contain the input address. If the input address is unambiguously a rural route, highway contract, PO Box, or general delivery address, then the return fields will contain the normalized version of the input address.

If parm.retcc is Z4_MULTIPLE, then parm.foot, parm.respn, and parm.stack are also returned by the system. The parm.zipc and/or parm.cris fields may contain data if all records in the stack have the same ZIP Code and/or carrier route ID.

If parm.retcc is Z4_SINGLE or Z4_DEFAULT, then all fields in the returned data section are returned by the Address Matching System. The first record in the parm.stack structure will contain the ZIP+4 record to which the system matched. This record may be used to access the individual fields from the matched record, such as primary name, suffix, post-directional, etc.

Example

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "zip4.h"

ZIP4_PARM parm;

int main(int argc, char** argv)
{
    Z4OPEN_PARM openparm;
    memset (&openparm, 0, sizeof(openparm));
    /* ... Populate openparm ... */
    /* open the USPS Address Matching System */
    if (z4opencfg(&openparm) != 0)
    {
        printf("The USPS Address Matching System failed to open");
        /* Always call z4close() even on open failure */
        z4close();
        exit(5);
    }

    /* load input address parameters */
    memset(&parm, 0, sizeof(parm));
    strcpy(parm.iadl2, "ACME TOOL AND DIE"); /* Firm line */
    strcpy(parm.iadl3, "STE 200" ); /* Secondary or extra line*/
    strcpy(parm.iadl1, "323 S 152ND ST" ); /* Primary address line */
    strcpy(parm.iprurb, "" ); /* Puerto Rico specific */
    strcpy(parm.ictyi, "OMAHA, NE 68154" ); /* City, State, ZIP */

    /* request address inquiry */
    z4adrinq(&parm);

    /* if a response found (either single or default) */
    if (parm.retcc==Z4_SINGLE || parm.retcc==Z4_DEFAULT)
    {
        printf("Found response.\n");
        printf("Name: %s\n", parm.dadl2);
        printf("S Addr: %s\n", parm.dadl3);
        printf("Addr: %s\n", parm.dadl1);
        printf("PRUrb: %s\n", parm.dprurb);
        printf("City: %s\n", parm.dctya);
        printf("ST: %s\n", parm.dstaa);
        printf("ZIP: %s\n", parm.zipc);
        printf("Addon: %s\n", parm.addon);
        printf("DPBC: %s\n", parm.dpbc);
        printf("Pre Dir: %s\n", parm.stack[0].pre_dir);
        printf("Str Name: %s\n", parm.stack[0].str_name);
        printf("Suffix: %s\n", parm.stack[0].suffix);
        printf("Post Dir: %s\n", parm.stack[0].post_dir);
        printf("Lacs Ind: %c\n", parm.stack[0].lacs_status);
    }

    /* close The USPS Address Matching System */
    z4close();

    exit(0)
}

```

Address Sort Key

The `z4adrkey()` function creates a sort key for an address. This function can be used in batch processes to sort an input file in the order that addresses are contained on the Address Matching System data files. However, the function does not sort your file; it produces a key field to assist your software in sortation. Sorting an input file usually produces a dramatic increase in processing throughput.

Syntax

```
#include "zip4.h"
int z4adrkey(ZIP4_PARM* parm);
```

Input

`parm` A pointer to a `ZIP4_PARM` structure that provides the input and where the output will be stored.

The following fields must be initialized before calling the `z4adrkey()` function.

<code>parm.iadl1</code>	Street Address
<code>parm.iadl2</code>	Firm Name
<code>parm.iprurb</code>	Puerto Rican Urbanization Name
<code>parm.ictyi</code>	City or City/ State/ ZIP
<code>parm.istai</code>	State or empty
<code>parm.izipc</code>	ZIP or empty

Output

`parm.adrkey` Address Sort Key

Note: *The contents and length of the address sort key are subject to change at any time. The key contains binary data and should be used in its entirety for the sort process.*

Return

- 0 - The USPS Address Matching System resident
- 1 - The USPS Address Matching System issued a system error
- 2 - The USPS Address Matching System not ready

Example

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "zip4.h"

ZIP4_PARM parm;

int main(int argc, char** argv)
{
    int i;

    Z4OPEN_PARM openparm;

    memset (&openparm, 0, sizeof(openparm));

    /* ... Populate openparm ... */

    /* open the USPS Address Matching System */
    if (z4opencfg(&openparm) != 0)
    {
        printf("The USPS Address Matching System failed to open");

        /* Always call z4close() even on open failure */
        z4close();

        exit(5);
    }

    /* load input address parameters */
    memset(&parm, 0, sizeof(parm));

    strcpy(parm.iadl2, "ACME TOOL AND DIE");/* Firm line          */
    strcpy(parm.iadl3, "STE 200"           );/* Secondary/extra line */
    strcpy(parm.iadl1, "323 S 152ND ST"    );/* Primary address line */
    strcpy(parm.iprurb, ""                 );/* Puerto Rico specific */
    strcpy(parm.ictyi, "OMAHA, NE 68154   );/* City, State, ZIP    */

    /* request address sort key */
    z4adrkey(&parm);

    /* print the address sort key in hex */
    for(i=0; i<sizeof(parm.adrkey); i++)
        printf("%02X", parm.adrkey[i]);

    printf("\n");

    /* close The USPS Address Matching System */
    z4close();
    exit(0);
}

```

9-digit Inquiry

The `z4xrfinq()` (9-digit Inquiry) function commands the Address Matching System to perform an address inquiry using an input 9-digit ZIP Code. Before using this function, the input 9-digit ZIP Code must be copied into the `parm.iadl1` field outlined below. Following the 9-digit inquiry, the `parm.retcc` field displays a return code summarizing the result of the inquiry. If an address response was found, standardized address information can be found in the output fields described in the Address Inquiry function description (page 22).

Note: *This function only returns matches to address records, not to specific addresses. Address records generally contain a range of possible addresses.*

To find a match to a specific address using only the ZIP Code, you will need to use the 11-Digit Inquiry function (page 32).

Syntax

```
#include "zip4.h"
int z4xrfinq(ZIP4_PARM* parm);
```

Input

`parm` A pointer to a `ZIP4_PARM` structure to provide the input and where the output will be stored.

The following field must be initialized before calling the `z4xrfinq()` function:

`parm.iadl1` 9-digit ZIP Code.

Note: *Return Code 22 denotes multiple responses. The address fields contain the first of a stack of ten possible responses (or matches).*

Output

parm.retcc	Response code
Z4_SINGLE	A single address was found
Z4_DEFAULT	A default address was found, but more specific addresses exist
Z4_NOTFND	No match found; considered a not found address
Z4_MULTIPLE	Multiple responses were found

Refer to the Address Inquiry function description for other output fields (page 22).

Return

- 0 - The USPS Address Matching System resident
- 1 - The USPS Address Matching System issued a system error
- 2 - The USPS Address Matching System not ready
- 3 - The USPS Address Matching System has expired

Example

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "zip4.h"

ZIP4_PARM parm;
int main(int argc, char** argv)
{
    Z4OPEN_PARM openparm;

    memset (&openparm, 0, sizeof(openparm));

    /* ... Populate openparm ... */

    /* open the USPS Address Matching System */
    if (z4opencfg(&openparm) != 0)
    {
        printf("The USPS Address Matching System failed to open");

        /* Always call z4close() even on open failure */
        z4close();

        exit(5);
    }

    /* load input 9-digit ZIP parameter */
    memset(&parm, 0, sizeof(parm));
    strcpy(parm.iadl1, "681642815");

    /* request address inquiry */
    z4xrfinq(&parm);

    /* if a response found (either single or default) */
    if(parm.retcc == Z4_SINGLE || parm.retcc == Z4_DEFAULT)
    {
        printf("Found response.\n");
        printf("Name:      %s\n", parm.dadl2);
        printf("Addr:      %s\n", parm.dadl1);
        printf("PRUrb:     %s\n", parm.dprurb);
        printf("City:      %s\n", parm.dctya);
        printf("ST: %s\n", parm.dstaa);
        printf("ZIP:       %s\n", parm.zipc);
        printf("Addon:     %s\n", parm.addon);
        printf("DPBC:     %s\n", parm.dpbc);
    }

    /* close The USPS Address Matching System */
    z4close();

    exit(0);
}

```

11-digit Inquiry

The `z4xrfinq11()` (11-digit Inquiry) function commands the Address Matching System to perform an address inquiry using an input 11-digit ZIP Code. Before using this function, the input 11-digit ZIP Code must be copied into the `parm.iad11` field outlined below. Following the 11-digit inquiry, the `parm.retcc` field displays a return code summarizing the result of the inquiry. If an address response was found, standardized address information can be found in the output fields described in the Address Inquiry function description (page 22).

Syntax

```
#include "zip4.h"
int z4xrfinq11(ZIP4_PARM* parm);
```

Input

`parm` A pointer to a `ZIP4_PARM` structure to provide the input and where the output will be stored.

The following field must be initialized before calling the `z4xrfinq11()` function:

`parm.iad11` 11-digit ZIP Code

Note *Return Code 22 denotes multiple responses. The address fields contain the first of a stack of ten possible responses (or matches). It is recommended that the first address in the output fields not be used as a mailing address because it is not an exact match.*

Output

parm.retcc	Response code
Z4_SINGLE	A single address was found
Z4_DEFAULT	A default address was found, but more specific addresses exist
Z4_NOTFND	No match found; considered a not found address
Z4_MULTIPLE	Multiple responses were found

Refer to the Address Inquiry function description for other output fields (page 22).

Return

- 0 - The USPS Address Matching System resident
- 1 - The USPS Address Matching System issued a system error
- 2 - The USPS Address Matching System not ready
- 3 - The USPS Address Matching System has expired

Example

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "zip4.h"

ZIP4_PARM parm;

int main(int argc, char** argv)
{
    Z4OPEN_PARM openparm;

    memset (&openparm, 0, sizeof(openparm));

    /* ... Populate openparm ... */

    /* open the USPS Address Matching System */
    if (z4opencfg(&openparm) != 0)
    {
        printf("The USPS Address Matching System failed to open");

        /* Always call z4close() even on open failure */
        z4close();

        exit(5);
    }

    /* load input 11-digit ZIP parameter */
    memset(&parm, 0, sizeof(parm));

    strcpy(parm.iadl1, "68164281527");

    /* request address inquiry */
    z4xrfinq11(&parm);

    /* if a response found (either single or default) */
    if(parm.retcc == Z4_SINGLE || parm.retcc == Z4_DEFAULT)
    {
        printf("Found response.\n");
        printf("Name:      %s\n", parm.dadl2);
        printf("Addr:      %s\n", parm.dadl1);
        printf("PRUrb:    %s\n", parm.dprurb);
        printf("City:     %s\n", parm.dctya);
        printf("ST: %s\n", parm.dstaa);
        printf("ZIP:      %s\n", parm.zipc);
        printf("Addon:    %s\n", parm.addon);
        printf("DPBC:    %s\n", parm.dpbc);
    }
    /* close The USPS Address Matching System */
    z4close();

    exit(0);
}

```

Address Standardization

The `z4adrstd()` (Address Standardization) function instructs the Address Matching System to standardize an address. This function can be used when a `Z4_MULTIPLE` response is returned from the `z4adring()` function. Use this function to standardize an address from the stack, but use it with caution. The index parameter is relative to zero and must be in increments of ten for each `z4scroll()` function called. Therefore, the index will have a value between zero and `parm.respn` minus one. Do not use the offset into the current stack of ten records.

When this function is called, the record corresponding to the index value is moved to the first position on the stack (offset zero). If components from the `ADDR_REC` structure are needed for the current record that was processed through `z4adrstd()`, they may be retrieved from the first stack record. Do not use the modulus 10 of the index (`index % 10`) to retrieve the `ADDR_REC` components from the stack.

Note: *This function should only be used when an operator is reviewing the multiple responses returned and selecting the record to be standardized. Please be advised that using this function in an unattended (batch) mode may result in inaccurate matches and possible failure to CASS certify.*

Syntax

```
#include "zip4.h"
int z4adrstd(ZIP4_PARM* parm, int index)
```

Input

<code>parm</code>	Pointer to the unmodified parameter list from the previous call to <code>z4adring()</code> .
<code>index</code>	Index of stack record to standardize address (refer to the description above). This must be less than <code>parm.respn</code> .

Output

<code>parm.dadl1</code>	Standardized Street Address
<code>parm.dadl2</code>	Standardized Firm Name
<code>parm.dprurb</code>	Standardized Puerto Rican Urbanization Name
<code>parm.dlast</code>	Standardized City/State/ZIP

Return

- 0 - Success
- 1 - Failure (i.e., invalid index parameter)
- 2 - The USPS Address Matching System not ready

Example

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "zip4.h"
ZIP4_PARM parm;

int main(int argc, char** argv)
{
    Z4OPEN_PARM openparm;
    memset(&openparm, 0, sizeof(openparm));

    /* ... Populate openparm ... */

    /* open the USPS Address Matching System */
    if (z4opencfg(&openparm) != 0)
    {
        printf("The USPS Address Matching System failed to open");
        /* Always call z4close() even on open failure */
        z4close();
        exit(5);
    }
    /* load input address parameters */
    memset(&parm, 0, sizeof(parm));

    strcpy(parm.iadl2, "ACME TOOL AND DIE"); /* Firm line */
    strcpy(parm.iadl3, ""); /* Secondary or extra line */
    strcpy(parm.iadl1, "1336 CHATMAN"); /* Primary address line */
    strcpy(parm.iprurb, ""); /* Puerto Rico specific */
    strcpy(parm.ictyi, "CORDOVA TN 38018"); /* City, State, ZIP */

    /* request address inquiry */
    z4adrinq(&parm);

    /* standardize second address */
    z4adrstd(&parm, 1);

    /* display address */
    printf("Found response.\n");
    printf("Name: %s\n", parm.dadl2);
    printf("Addr: %s\n", parm.dadl1);
    printf("PRUrb: %s\n", parm.dprurb);
    printf("City: %s\n", parm.dctya);
    printf("ST: %s\n", parm.dstaa);
    printf("ZIP: %s\n", parm.zipc);
    printf("Addon: %s\n", parm.addon);
    printf("DPBC: %s\n", parm.dpbc);

    /* close The USPS Address Matching System */
    z4close();

    exit(0);
}

```

Close the Address Matching System

The `z4close()` function closes the Address Matching System and is called when address inquiries have been completed and the interface is no longer needed. During execution of this function, memory buffers and file handles allocated during the `z4opencfg()` function are de-allocated and closed.

Note: *The `z4close()` function call must be called after all calls to the `z4opencfg()` function call – regardless if `z4opencfg()` succeeded or failed.*

Syntax

```
#include "zip4.h"
int z4close(void);
```

Input

None

Output

None

Return

- 0 - The USPS Address Matching System closed
- 1 - The USPS Address Matching System not resident
- 2 - The USPS Address Matching System not ready

Example

```
#include <stdio.h>
#include "zip4.h"

void main(void)
{
    /* close The USPS Address Matching System */
    if(z4close() == 0)
        printf("The USPS Address Matching System closed.\n");
    else
        printf("Error closing the USPS Address Matching System.\n");
}
```

Read City/State File By Key

The `z4ctyget()` (Read City/State File By Key) function initiates a read of the City/State File. A specific ZIP Code can be selected as a starting point in a read of the City/State File. To read subsequent records, the Read City/State File Next function is used. For documentation on the City/State File, please refer to the *Address Information System Products Technical Guide*, which is available from the USPS National Customer Support Center's Customer Support Department at 800-238-3150. It is also available on the Internet at <http://ribbs.usps.gov/files/addressing/pubs>

Syntax

```
#include "zip4.h"
int z4ctyget(CITY_REC* cityrec, char* zipcode);
```

Input

`cityrec` Pointer to an empty CITY_REC structure where the output will be stored.
`zipcode` Pointer to a char array containing a 5 digit ZIP, or "00000", as the starting point.

Output

The `cityrec` argument will be populated with the city information for the provided 5 digit ZIP code.

Return

0 - Success
 1 - Failure
 2 - The USPS Address Matching System not ready

Example

See example code for "Read City/State File Next" (Page 38).

Read City/State File Next

The `z4ctynxt ()` (Read City/State File Next) function reads subsequent records of the City/State File. It can only be used after the `z4ctyget ()` function has been called.

Note: *Multiple calls to `z4ctynxt ()` can not be mixed with calls to other Address Matching System functions. This function is designed to be called after a `z4ctyget ()` or previous `z4ctynxt ()` function call. The results of the `z4ctynxt ()` are undefined if it is called after any other AMS function call.*

Syntax

```
#include "zip4.h"  
int z4ctynxt(CITY_REC* cityrec);
```

Input

`cityrec` Pointer to an empty CITY_REC structure where the output will be stored.

Output

The `cityrec` argument will be populated with the next city record in the database.

Return

0 - Success
1 - Failure
2 - The USPS Address Matching System not ready

Example

```

#include <stdio.h>
#include <stdlib.h>
#include "zip4.h"

CITY_REC city;

int main(int argc, char** argv)
{
    int i;

    Z4OPEN_PARM openparm;

    memset (&openparm, 0, sizeof(openparm));

    /* ... Populate openparm ... */

    /* open the USPS Address Matching System */
    if (z4opencfg(&openparm) != 0)
    {
        printf("The USPS Address Matching System failed to open");

        /* Always call z4close() even on open failure */
        z4close();

        exit(5);
    }

    /* read first city */
    z4ctyget(&city, "00000");

    /* read 10 more cities */
    for(i=0; i<10 && z4ctynext(&city) == 0; ++i)
    {
        printf("%s %-28.28s %s %s\n" city.zip_code,
              city.city_name,
              city.state_abbrev,
              city.finance);
    }

    /* close The USPS Address Matching System */
    z4close();
    exit(0);
}

```

Read ZIP+4 File By Key

The `z4adrget ()` (Read ZIP+4 File by Key) function is used to read the ZIP+4 File. For documentation on the ZIP+4 File, please refer to the Address Information Products Technical Guide, which is available from the USPS National Customer Support Center's Customer Support Department at 800-238-3150. It is also available on the Internet at <http://ribbs.usps.gov/files/addressing/pubs>. A specific postal finance number can be selected as a starting point in a read of the ZIP+4 File. To read subsequent records, the `z4adrnxt()` function is used. To read previous records, the `z4adrprv` function is used.

Syntax

```
#include "zip4.h"
int z4adrget(ADDR_REC* addrrec, char* finance);
```

Input

`addrrec` A pointer to an empty ADDR_REC structure.
`finance` A pointer to a char array containing the starting finance number or "000000"

Output

The `addrrec` argument will be populated with the first address for the finance number provided.

Return

0 - Success
 1 - Failure
 2 - The USPS Address Matching System not ready

Example

See example code for "Read ZIP+4 File Next" (page 41)

Read ZIP+4 File Next

The `z4adrnxt ()` (Read ZIP+4 File Next) function reads subsequent records of the ZIP+4 File. It can only be used after the `z4adrget ()` function has been called.

Note: *Multiple calls to `z4adrnxt ()` can not be mixed with calls to other Address Matching System functions. This function is designed to follow a `z4adrget ()`, `z4adrprv ()` or another `z4adrnxt ()` function call. The results of `z4adrnxt ()` are undefined if it is called after any other AMS function*

Syntax

```
#include "zip4.h"
int z4adrnxt(ADDR_REC* addrrec);
```

Input

`addrrec` A pointer to an empty `ADDR_REC` structure where the output will be stored.

Output

The `addrrec` argument will be populated with the next address in the database.

Return

- 0 - Success
- 1 - Failure
- 2 - The USPS Address Matching System not ready

Example

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "zip4.h"

CITY_REC city;
ADDR_REC addr;

int main(int argc, char** argv)
{
    Z4OPEN_PARM openparm;

    memset (&openparm, 0, sizeof(openparm));

    /* ... Populate openparm ... */

    /* open the USPS Address Matching System */
    if (z4opencfg(&openparm) != 0)
    {
        printf("The USPS Address Matching System failed to open");

        /* Always call z4close() even on open failure */
        z4close();

        exit(5);
    }

    /* read a city */
    z4ctyget(&city, "00000");

    /* read first address record for this city */
    z4adrget(&addr, city.finance);

    /* read remaining adrs for this finance number */
    while(z4adrnxt(&addr) == 0)
    {
        /* check if finance number has changed */
        if (memcmp(addr.finance, city.finance, 6) != 0)
            break;

        /* Code to process the current address record. */
    }

    /* close The USPS Address Matching System */
    z4close();

    exit(0);
}

```

Read ZIP+4 File Previous

The `z4adrprv()` (Read ZIP+4 File Previous) function reads prior records of the ZIP+4 File within a ZIP code. It can only be used after the `z4adrget()` function has been called.

Note: *Multiple calls to `z4adrprv()` can not be mixed with calls to other Address Matching System functions. This function is designed to follow a `z4adrget()`, `z4adrnxt()` or another `z4adrprv()` function call. The results of `z4adrprv()` are undefined if it is called after any other AMS function*

Syntax

```
#include "zip4.h"
int z4adrprv(ADDR_REC* addrrec);
```

Input

`addrrec` A pointer to an empty `ADDR_REC` structure where the output will be stored.

Output

The `addrrec` argument will be populated with the previous address in the database.

Return

0 - Success
1 - Failure
2 - The USPS Address Matching System not ready

Example

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "zip4.h"

CITY_REC city;
ADDR_REC addr;

int main(int argc, char** argv)
{
    Z4OPEN_PARM openparm;

    memset (&openparm, 0, sizeof(openparm));

    /* ... Populate openparm ... */

    /* open the USPS Address Matching System */
    if (z4opencfg(&openparm) != 0)
    {
        printf("The USPS Address Matching System failed to open");

        /* Always call z4close() even on open failure */
        z4close();

        exit(5);
    }

    /* read a city */
    z4ctyget(&city, "00000");

    /* read first address record for this city */
    z4adrget(&addr, city.finance);

    /* read previous addrs for this finance number */
    while(z4adrprv(&addr) == 0)
    {
        /* check if finance number has changed */
        if (memcmp(addr.finance, city.finance, 6) != 0)
            break;

        /* Code to process the current address record. */
    }

    /* close The USPS Address Matching System */
    z4close();

    exit(0);
}

```

Get ZIP Codes from a City/State

The `z4getzip()` (Get ZIP Codes) from a City/State function retrieves a range of ZIP Codes for a city or state and returns the valid high and the low values for the input city/state. The standardized form of the input city/state as well as the finance number is also returned.

Note: All ZIP Codes within the range are not necessarily valid.

Syntax

```
#include "zip4.h"
int z4getzip(GET_ZIPCODE_STRUCT* parm);
```

Input

`parm` A pointer to a `GET_ZIPCODE_STRUCT` structure to provide the input and where the output will be stored.

The requested city and state must be populated before calling the function.

`parm.input_cityst` Input city/state to lookup

Output

<code>parm.output_cityst</code>	Standardized city/state
<code>parm.low_zipcode</code>	Low ZIP Code value
<code>parm.high_zipcode</code>	High ZIP Code value
<code>parm.finance_num</code>	Finance number

Return

0 - Success

1 - Failure

Example

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "zip4.h"
GET_ZIPCODE_STRUCT parm;

int main(int argc, char** argv)
{
    Z4OPEN_PARM openparm;

    int result;

    memset (&openparm, 0, sizeof(openparm));

    /* ... Populate openparm ... */

    /* open the USPS Address Matching System */
    if (z4opencfg(&openparm) != 0)
    {
        printf("The USPS Address Matching System failed to open");
        /* Always call z4close() even on open failure */
        z4close();

        exit(5);
    }

    /* read a city */
    strcpy(parm.input_cityst, "MEMPHIS TN");
    result=z4getzip(&parm);

    /* Display the ZIP codes found */
    if(result == 0)
    {
        printf("CITY FOUND:      %s\n",parm.output_cityst);
        printf("LOW ZIP:          %s\n",parm.low_zipcode);
        printf("HIGH ZIP: %s\n",parm.high_zipcode);
        printf("FINANCE:          %s\n",parm.finance_num);
    }

    /* close The USPS Address Matching System */
    z4close();

    exit(0);
}

```

Terminate Active Address Inquiry

The `z4abort()` (Terminate Active Address Inquiry) function terminates an active address inquiry and is useful in real-time applications where each inquiry must be completed within a specified period of time. This function would normally be called from within a timer interrupt handler. The `z4adring()` call in progress is terminated by the function call.

Syntax

```
#include "zip4.h"  
int z4abort(void);
```

Input

None

Output

None

Return

None

Get Date of ZIP+4 Database

The `z4date()` (Get Date of ZIP+4 Database) function returns the date of the ZIP+4 database and prints the date for PS Form 3553 (CASS certificate). The date is returned as an 8-byte character string in the “YYYYMMDD” format.

Syntax

```
#include "zip4.h"
int z4date(char* date);
```

Input

`date` A pointer to a char array that will be modified to contain the date of the database. The char array must be at least nine (9) bytes in length.

Output

The date of the ZIP+4 database. This field must be at least nine (9) bytes in length.

Return

- 0 - Success
- 1 - Failure
- 2 - The USPS Address Matching System not ready

Example

```

#include <stdio.h>
#include <stdlib.h>
#include "zip4.h"

char date[9];

int main(int argc, char** argv)
{
    Z4OPEN_PARM openparm;

    memset (&openparm, 0, sizeof(openparm));

    /* ... Populate openparm ... */

    /* open the USPS Address Matching System */
    if (z4opencfg(&openparm) != 0)
    {
        printf("The USPS Address Matching System failed to open");

        /* Always call z4close() even on open failure */
        z4close();

        exit(5);
    }

    /* get release date */
    z4date(date);
    printf("Release date: %s\n", date);

    /* close The USPS Address Matching System */
    z4close();

    exit(0);
}

```

Get AMS Data Expiration

The `z4GetDataExpireDays()` (Get AMS Data Expiration) function instructs the Address Matching System to return the number of days until the AMS database expires. Because the function can be used periodically to check the number of days remaining until database expiration, it is strongly recommended that you integrate this function into your software.

Note: This function replaces the `z4expire()` function.

Syntax

```
#include "zip4.h"  
int z4GetDataExpireDays(void);
```

Input

None

Output

None

Return

-1 The AMS database has expired. Otherwise, the number of days until the AMS database expires.

Example

```

#include <stdio.h>
#include <stdlib.h>
#include "zip4.h"

int main(int argc, char** argv)
{
    Z4OPEN_PARM openparm;

    int days = 0;

    memset (&openparm, 0, sizeof(openparm));

    /* ... Populate openparm ... */

    /* open the USPS Address Matching System */
    if (z4opencfg(&openparm) != 0)
    {
        printf("The USPS Address Matching System failed to open");

        /* Always call z4close() even on open failure */
        z4close();
        exit(5);
    }

    /* get number of days until database expiration */
    days = z4GetDataExpireDays();

    if (days == -1)
        printf("AMS database has already expired.\n");
    else
        printf("%d days until AMS database expiration.\n", days);

    /* close The USPS Address Matching System */
    z4close();
}

```

Get AMS Library Expiration

The `z4GetCodeExpireDays()` (Get AMS Library Expiration) function instructs the Address Matching System to return the number of days until the AMS library expires. Because the function can be used periodically to check the number of days remaining until library expiration, it is strongly recommended that you integrate this function into your software.

Syntax

```
#include "zip4.h"  
int z4GetCodeExpireDays(void);
```

Input

None

Output

None

Return

-1 – The AMS library has expired. Otherwise, the number of days until the AMS library expires.

Example

```

#include <stdio.h>
#include <stdlib.h>
#include "zip4.h"

int main(int argc, char** argv)
{
    Z4OPEN_PARM openparm;

    int days = 0;

    memset (&openparm, 0, sizeof(openparm));

    /* ... Populate openparm ... */

    /* open the USPS Address Matching System */
    if (z4opencfg(&openparm) != 0)
    {
        printf("The USPS Address Matching System failed to open");

        /* Always call z4close() even on open failure */
        z4close();

        exit(5);
    }
    /* get number of days until database expiration */
    days = z4GetCodeExpireDays();

    if (days == -1)
        printf("AMS library has already expired.\n");
    else
        printf("%d days until AMS library expiration.\n", days);

    /* close The USPS Address Matching System */
    z4close();
}

```

Get API Code Version

The `z4ver()` (Get API Code Version) function commands the program to retrieve the version string of the API code. This string is in compliance with the CASS requirements for address matching software version information and may be used when generating a PS Form 3553 for mailing discounts.

Note: *Most functions require you to call `z4opencfg()` first to initialize the AMS system. This function does not require the AMS system to be open.*

Syntax

```
#include "zip4.h"
int z4ver(char* ver);
```

Input

`ver` A pointer to a char array where the output will be stored.

Output

The `ver` argument will be populated with the version string.

Return

0 - Success

Example

```
#include <stdio.h>
#include "zip4.h"
void main(void)
{
    char version[32];

    /* get the Address Matching System version */
    z4ver(version) ;

    printf("The Address Matching System version is %s\n", version) ;
    exit (0);
}
```

Multiple Response Stack

Scroll the Stack of Address Records

The `z4scroll()` (Scroll the Stack of Address Records) function commands the Address Matching System to access additional stacks of ten address records each. The function is related to the `z4adrinq()` and `z4xrfinq()` functions, which return up to ten records when the `Z4_MULTIPLE` or `Z4_DEFAULT` return codes are set. When the `parm.respn` field contains a number greater than ten, your program can use this function to obtain additional stacks of ten address records (up to the number of records specified in the `parm.respn` return field). This function may only be called immediately after a call to the `z4adrinq()` or `z4xrfinq()` functions.

Syntax

```
#include "zip4.h"
int z4scroll(ZIP4_PARM* parm);
```

Input

`parm` A pointer to the unmodified `ZIP4_PARM` structure that was returned from a previous `z4adrinq()` call.

Output

The `parm.stack` field will be updated to contain the next ten records (fewer records may be returned if less than ten records remain).

Return

- 0 - Success
- 1 - The USPS Address Matching System not installed
- 2 - The USPS Address Matching System not open
- 3 - Stack access not allowed

Example

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "zip4.h"

ZIP4_PARM parm;

int main(int argc, char** argv)
{
    Z4OPEN_PARM openparm;

    int i = 0;

    memset (&openparm, 0, sizeof(openparm));

    /* ... Populate openparm ... */

    /* open the USPS Address Matching System */
    if (z4opencfg(&openparm) != 0)
    {
        printf("The USPS Address Matching System failed to open");

        /* Always call z4close() even on open failure */
        z4close();

        exit(5);
    }
    /* Create parameter list and call the USPS Address Matching System */
    memset(&parm, 0, sizeof(parm));

    strcpy(parm.iadl1, "1336 CHATMAN" );
    strcpy(parm.ictyi, "CORDOVA TN" );

    z4adrinq(&parm);

    /*process all addresses returned by The USPS Address Matching System */
    for(i=0; i<parm.respn; i++)
    {
        /* check if stack needs to be refreshed */
        if (i != 0 && (i% 10) == 0)
        {
            if(z4scroll(&parm))
                break;
        }
    }
    /* examine each address returned by The USPS Address Matching System */
    ...
}
/* close The USPS Address Matching System */
z4close();

exit(0);
}

```

Get Last Error

The `z4geterror()` (Get Last Error) function retrieves the last error that was encountered after a failed `z4opencfg()` function call.

Syntax

```
#include "zip4.h"
int z4geterror(Z4_ERROR* pError);
```

Input

`pErrorPointer` to an empty `Z4_ERROR` structure where the output will be stored.

Output

`pError` will be populated with the last error that was encountered.

Return

#defines for the *iErrorCode* values and their meanings:

<code>EROR_FILE_OPEN</code>	1	Error opening a file
<code>ERROR_FILE_READ</code>	2	Error reading a file
<code>ERROR_FILE_WRITE</code>	3	Error writing to a file
<code>ERROR_FILE_FIND</code>	4	Error finding a file
<code>ERROR_FILE_EXPIRE</code>	5	AMS library has expired
<code>ERROR_FILE_SYNC</code>	6	AMS Database files are out of sync
<code>ERROR_SECURITY</code>	7	AMS Security error

Example

```

#include <stdio.h>
#include <string.h>
#include "zip4.h"

int main(int argc, char** argv)
{
    Z4_ERROR errorparm;
    Z4_ENV envparm;
    Z4OPEN_PARM openparm;
    memset(&errorparm, 0, sizeof(Z4_ERROR) );
    memset(&envparm, 0, sizeof(Z4_ENV) );
    memset(&openparm, 0, sizeof(Z4OPEN) PARM));

    /* ... Populate openparm ... */
    /* open the USPS Address Matching System */
    if(z4opencfg(&openparm) != 0)
    {
        printf("The USPS Address Matching System failed to open");
        z4getenv(&envparm);
        z4geterror(&errorparm);

        /* Detailed Error Information */
        printf("\n\nDETAILED ERROR INFORMATION\n");
        printf("-----\n");
        printf("Error Message:  %s\n", errorparm.strErrorMessage);
        printf("File Name:      %s\n", errorparm.strFileName);
        printf("Diagnostics:    %s\n", errorparm.strDiagnostics);

        /* Detailed Environment Information */
        printf("\n\nDETAILED ENVIRONMENT INFORMATION\n");
        printf("-----\n");
        printf("Configuration File: %s\n", envparm.strConfigFile);
        printf("Address1:           %s\n", envparm.address1);
        printf("AddrIndex:          %s\n", envparm.addrindex);
        printf("CityState:          %s\n", envparm.citystate);
        printf("CrossRef:           %s\n", envparm.crossref);
        printf("System:             %s\n", envparm.system);
        printf("eLOT:               %s\n", envparm.elot);
        printf("eLOTIndex:          %s\n", envparm.elotindex);
        printf("EWS Path:           %s\n", envparm.ewspath);
        printf("eLOT Flag:          %s\n", envparm.elotflag);
    }
    else
    {
        printf("The USPS Address Matching System opened successfully\n");
    }
    return 0;
}

```

Get Environment

The `z4getenv()` (Get Environment) function retrieves the environment for the Address Matching System.

Syntax

```
#include "zip4.h"  
int z4getenv(Z4_ENV* pEnv);
```

Input

`pEnv` Pointer to an empty `Z4_ENV` structure where the output will be stored.

Output

`pEnv` will be populated with the environment for the Address Matching System.

Return

0 – Success

Example

See example code for “Get Last Error” (page 57)

Retrieving the LACS^{Link}® Security Key

The `z4LLkGetKey()` function returns the stop processing security key used to disable LACS^{Link}®. A stop processing security key is an alphanumeric character string that is randomly generated when a LACS^{Link}® security violation occurs.

You may call the `z4LLkGetKey()` function after a LACS^{Link}® security violation occurs. In order to identify a LACS^{Link}® security violation, a return value of 7 (seven) is given after making an open call. At that point you may call `z4LLkGetKey()` to retrieve the randomly generated stop processing security key.

The stop processing security key returned from `z4LLkGetKey()` will be used to generate the corresponding enable security key you need for `z4LLkSetKey()`. You can obtain an enable security key from a customer care representative in exchange for the stop processing security key given to you by `z4LLkGetKey()`.

Note: During a `z4LLkGetKey()` call OS resources are allocate so a call to `z4close()` must be made in order to free the resources.

Syntax

```
#include "zip4.h"
const char* Z4FUNC z4LLkGetKey(void);
```

Input

None

Output

None

Return

`const char*` - pointer to a null terminated alphanumeric character string

Example

```

#include <stdio.h>
#include "zip4.h"

void main( void)
{
    Z4OPEN_PARM OpenParm;
    char szKey[32] = {0};
    int iReturn = 0;

    memset(&openparm, 0, sizeof(Z4OPEN_PARM));

    /* Setting up paths */
    OpenParm.config.address1 = "c:\\amsdata\\";
    OpenParm.config.addrindex = "c:\\amsdata\\";
    OpenParm.config.cdrom = "d:\\";

    OpenParm.config.citystate = "c:\\amsdata\\";
    OpenParm.config.crossref = "c:\\amsdata\\";
    OpenParm.config.system = "c:\\amsdata\\";
    OpenParm.config.llkpath = "c:\\llkdata\\";
    OpenParm.config.dpvpath = "c:\\dpvdata\\";

    /* open the USPS Address Matching System */
    iReturn = z4opencfg(&OpenParm);

    /* success */
    if( iReturn == 0 )
    {
        printf("\nThe USPS Address Matching System Opened
        Successfully.");
    }
    /* LACSLink security violation */
    else if( iReturn == 7 )
    {
        const char* szCode = z4LLkGetKey();

        /* display error message an security code */
        printf("\nLACSLink has been disabled.");
        printf("\n\nSecurity code: %s", szCode);
        printf("\nTo enable LACSLink contact customer support with the
        security");
        printf("\ncode above to receive the security key you need to
        enable LACSLink\n");

        /* prompt for security key */
        printf("\nEnter security key w/o formatting characters: ");
        gets(szKey);
    }
}

```

```
    /* verify security key */
    if( z4LkSetKey(szKey) )
    {
        /* inform user of success */
        printf("\nThe key %s is valid and LACSLink is enabled.",
szKey);
    }
    else
    {
        /* inform user of failure */
        printf("\nThe key %s is invalid and LACSLink is disabled.",
szKey);
    }
}
/* other errors */
else
{
    printf("\nError Opening the USPS Address Matching System.");
}

/* close the USPS Address Matching System */
z4close();
}
```

Checking for LACS^{Link} functionality

The `z4LLkIsDisabled()` identifies when LACS^{Link} functionality is enable/disabled. When the return value from `z4LLkIsDisabled()` is TRUE (non-zero) LACS^{Link} is disabled otherwise LACS^{Link} is enabled.

Before LACS^{Link} can be enabled a system open call must be made with the `Z4OPEN_PARM.llkflag` set to 'Y', and the `Z4OPEN_PARM.config.llkpath` containing the path to the LACS^{Link} data files. After the open call you may check the state of LACS^{Link} via `z4LLkIsDisabled()`.

Note: During a `z4LLkIsDisabled()` call OS resources may be allocate so a call to `z4close()` must be made in order to free the resources.

Syntax

```
#include "zip4.h"
int Z4FUNC z4LLkIsDisabled(void);
```

Input

None

Output

None

Return

TRUE - LACS^{Link} is disabled
FALSE - LACS^{Link} is enabled

Example

```

#include <stdio.h>
#include "zip4.h"

void main( void)
{
    Z4OPEN_PARM OpenParm;
    char szKey[32] = {0};
    int iReturn = 0;

    memset(&openparm, 0, sizeof(Z4OPEN_PARM));

    /* Setting up paths */
    OpenParm.config.address1      = "c:\\amsdata\\";
    OpenParm.config.addrindex    = "c:\\amsdata\\";
    OpenParm.config.cdrom        = "d:\\";
    OpenParm.config.citystate    = "c:\\amsdata\\";
    OpenParm.config.crossref     = "c:\\amsdata\\";
    OpenParm.config.system       = "c:\\amsdata\\";
    OpenParm.config.llkpath      = "c:\\llkdata\\";
    OpenParm.config.dpvpath      = "c:\\dpvdata\\";

    /* open the USPS Address Matching System */
    iReturn = z4opencfg(&OpenParm);

    /* success */
    if( iReturn == 0 )
    {
        printf("\nThe USPS Address Matching System Opened Successfully.");
    }
    /* LACSLink security violation */
    else if( iReturn == 7 )
    {
        const char* szCode = z4LLkGetCode();

        /* display error message an security code */
        printf("\nLACSLink has been disabled.");
        printf("\n\nSecurity code: %s", szCode);
        printf("\nTo enable LACSLink contact customer support with the
security");
        printf("\ncode above to receive the security key you need to
enable LACSLink\n");

        /* prompt for security key */
        printf("\nEnter security key w/o formatting characters: ");
        gets(szKey);
    }
}

```

```
    /* verify security key */
    if( z4LlkSetKey(szKey) )
    {

        /* inform user of success */
        printf("\nThe key %s is valid and LACSLink is enabled.", szKey);
    }
    else
    {
        /* inform user of failure */
        printf("\nThe key %s is invalid and LACSLink is disabled.",
szKey);
    }
}
/* other errors */
else
{
    printf("\nError Opening the USPS Address Matching System.");
}

/* close the USPS Address Matching System */
z4close();
}
```

Disabling the LACS^{Link}® Security Key

The `z4LLkSetKey()` function verifies the stop processing security key used for enabling LACS^{Link}® after a LACS^{Link}® security violation. A security key is an alphanumeric character string given to you by a customer care representative in exchange for the security code given to you by `z4LLkGetCode()`.

Make a call to `z4LLkSetKey()` after a LACS^{Link}® security violation. In order to identify a LACS^{Link}® security violation, a return value of 7 (seven) is given after making an open call. At that point you may call `z4LLkSetKey()` with the security key provided to you by some customer care representative.

A status of TRUE (non-zero) is returned to identify success (LACS^{Link}® is enabled) otherwise failure occurred (LACS^{Link}® is disabled).

Note: `z4LLkSetKey()` must be called after a `z4openCfg()` function call. Even if the `z4openCfg()` function call fails to open AMS, it has put AMS in a state to be able to accept the key information.

Since this process causes AMS to allocate OS resources, the `z4close()` function call must be called in order to allow AMS to free those resources.

Syntax

```
#include "zip4.h"
int Z4FUNC z4LLkSetKey(const char* szKey);
```

Input

`szKey` A pointer to a null terminated alphanumeric character string

Output

None

Return

TRUE - The key update is successful and LACS^{Link} is enabled
 FALSE - The key update failed and LACS^{Link} is disabled

Example

```

#include <stdio.h>
#include "zip4.h"
void main( void)
{
    Z4OPEN_PARM OpenParm;
    char szKey[32]    = {0};
    int iReturn = 0;

    memset(&OpenParm, 0, sizeof(Z4OPEN_PARM));

    /* Setting up paths */
    OpenParm.config.address1      = "c:\\amsdata\\";
    OpenParm.config.addrindex    = "c:\\amsdata\\";
    OpenParm.config.ctystate     = "c:\\amsdata\\";
    OpenParm.config.crossref     = "c:\\amsdata\\";
    OpenParm.config.system       = "c:\\bin\\";
    OpenParm.config.llkpath      = "c:\\llkdata\\";
    OpenParm.config.dpvpath      = "c:\\dpvdata\\";

    /* open the USPS Address Matching System */
    iReturn = z4opencfg(&OpenParm);

    /* success */
    if( iReturn == 0 )
    {
        printf("\nThe USPS Address Matching System Opened Successfully.");
    }
    /* LACSLink security violation */
    else if( iReturn == 7 )
    {
        const char* szCode = z4LLkGetCode();

        /* display error message an security code */
        printf("\nLACSLink has been disabled.");
        printf("\n\nSecurity code: %s", szCode);
        printf("\nTo enable LACSLink contact customer support with the
security");
        printf("\ncode above to receive the security key you need to
enable LACSLink\n");

        /* prompt for security key */
        printf("\nEnter security key w/o formatting characters: ");
        gets(szKey);
    }
}

```

```
/* verify security key */
if( z4LkSetKey(szKey) )
{
    /* inform user of success */
    printf("\nThe key %s is valid and LACSLink is enabled.",
szKey);
}
else
{
    /* inform user of failure */
    printf("\nThe key %s is invalid and LACSLink is disabled.",
szKey);
}
}
/* all other errors */
else
{
    printf("\nError Opening the USPS Address Matching System.");
}

/* close the USPS Address Matching System */
z4close();
}
```

SUITE^{LINK™} Database Date

The `z4SLNKGetDate()` function returns the date of the Suite^{Link™} database. The date is returned as an 8-byte character string in the YYYYMMDD format.

Syntax

```
#include "zip4.h"
const char* Z4FUNC z4SLNKGetDate(int IID);
```

Notes: *The date string is null terminated and always returned unless the SUITE^{Link™} library is not loaded correctly or the database is not found. The return value is zero/null when the date string is not returned.*

Input

`IID` A numerical value identifying a data table or -1 for entire database.

Output

None

Return

A pointer to a NULL terminated char array that contains the date associated with the table(s) in the database. Format: YYYYMMDD.

Example

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "zip4.h"

ZIP4_PARM parm;

int main(int argc, char** argv)
{
    Z4OPEN_PARM openparm;
    memset (&openparm, 0, sizeof(openparm));
    /* ... Populate openparm ... */
    /* open the USPS Address Matching System */
    if (z4opencfg(&openparm) != 0)
    {
        printf("The USPS Address Matching System failed to open");
        /* Always call z4close() even on open failure */
        z4close();
        exit(5);
    }
}
```

```

/* get database date */
printf("SuiteLink database date: %s\n", z4SLNKGetDate (-1) );

/* load input address parameters */
memset(&parm, 0, sizeof(parm));
strcpy(parm.iadl2, "ACME TOOL AND DIE");/* Firm line */
strcpy(parm.iadl3, "" );/* Secondary or extra line*/
strcpy(parm.iadl1, "323 S 152ND ST" );/* Primary address line */
strcpy(parm.iprurb, "" );/* Puerto Rico specific */
strcpy(parm.ictyi, "OMAHA, NE 68154 ");/* City, State, ZIP */

/* request address inquiry */
z4adrinq(&parm);

/* request SuiteLink inquiry */
z4SLNKQuery(&parm);

/* close The USPS Address Matching System */
z4close();
exit(0);
}

```

SUITE^{LINK™} Error Code

The z4SLNKGetError() function retrieves the last error encountered during Suite^{Link™} processing.

Syntax

```
#include "zip4.h"  
long Z4FUNC z4SLNKGetError(void);
```

Notes: This interface does not return AMS errors, it only returns errors pertaining strictly to SUITE^{Link™}. The purpose of this function is to aid in debugging and logging issues.

Input

None

Output

None

Return

Integer value identifying the last error

Example

See example code for Suite^{Link} Error Message (page 72)

SUITE^{LINK™} Error Message

The `z4SLNKGetErrorMsg()` function retrieves the last error encountered during a Suite^{Link™} lookup.

Syntax

```
#include "zip4.h"
const char* Z4FUNC z4SLNKGetErrorMsg(void);
```

Notes: This is a null terminated ASCII string and it may not be formatted enough for user feedback.

Input

None

Output

None

Return

A pointer to a Null terminated char array containing a text description of the last error.

Example

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "zip4.h"

ZIP4_PARM parm;

int main(int argc, char** argv)
{
    Z4OPEN_PARM openparm;
    memset (&openparm, 0, sizeof(openparm));
    /* ... Populate openparm ... */
    /* open the USPS Address Matching System */
    if (z4opencfg(&openparm) != 0)
    {
        printf("The USPS Address Matching System failed to open");

        /* SuiteLink Error */
        printf("Error %d:          %s\n", z4SLNKGetError(),
              z4SLNKGetErrorMsg() );

        /* Always call z4close() even on open failure */
        z4close();
        exit(5);
    }

    /* load input address parameters */
```

```

memset(&parm, 0, sizeof(parm));
strcpy(parm.iadl2, "ACME TOOL AND DIE");/* Firm line */
strcpy(parm.iadl3, "" );/* Secondary or extra line*/
strcpy(parm.iadl1, "323 S 152ND ST" );/* Primary address line */
strcpy(parm.iprurb, "" );/* Puerto Rico specific */
strcpy(parm.ictyi, "OMAHA, NE 68154 ");/* City, State, ZIP */

/* request address inquiry */
z4adrinq(&parm);

/* request SuiteLink inquiry */
z4SLNKQuery(&parm);

/* if a response found (either single or default) */
if(parm.retcc==Z4_SINGLE || parm.retcc==Z4_DEFAULT)
{
    printf("Found response.\n");
    printf("Name: %s\n", parm.dadl2);
    printf("S Addr: %s\n", parm.dadl3);
    printf("Addr: %s\n", parm.dadl1);
    printf("PRUrb: %s\n", parm.dprurb);
    printf("City: %s\n", parm.dctya);
    printf("ST: %s\n", parm.dstaa);
    printf("ZIP: %s\n", parm.zipc);
    printf("Addon: %s\n", parm.addon);
    printf("DPBC: %s\n", parm.dpbc);
    printf("Pre Dir: %s\n", parm.stack[0].pre_dir);{
    printf("Str Name: %s\n", parm.stack[0].str_name);
    printf("Suffix: %s\n", parm.stack[0].suffix);
    printf("Post Dir: %s\n", parm.stack[0].post_dir);
    printf("Lacs Ind: %c\n", parm.stack[0].lacs_status);
    }
}

/* close The USPS Address Matching System */
z4close();

exit(0)
}

```

SUITE^{LINK™} Query

The `z4SLNKQuery()` function should be used to correct missing secondary information when a call from `z4adring()` returns a default (32) response.

Syntax

```
#include "zip4.h"
int Z4FUNC z4SLNKQuery(ZIP4_PARM* pZip4);
```

Notes: If successful the `parm.dad11` address line will contain the secondary information found during the query.

The `parm.stefoot` will contain one of the following values to identify the status of the query.

“A “ - Confirmed entire address
 “00” - Could not confirm address
 “” - Address was not submitted for confirmation

Input

`pZip4` A pointer to a `ZIP4_PARM` structure that contains the address to perform the query on. The contents of the structure will be altered to contain the secondary information for the input address.

Output

None

Return

TRUE - The address was confirmed

FALSE - The address was not confirmed.

Example

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "zip4.h"

ZIP4_PARM parm;

int main(int argc, char** argv)
{
    Z4OPEN_PARM openparm;
    memset (&openparm, 0, sizeof(openparm));
    /* ... Populate openparm ... */
    /* open the USPS Address Matching System */
    if (z4opencfg(&openparm) != 0)
    {
        printf("The USPS Address Matching System failed to open");
        /* Always call z4close() even on open failure */
        z4close();
        exit(5);
    }

    /* load input address parameters */
```

```

memset(&parm, 0, sizeof(parm));
strcpy(parm.iadl2, "ACME TOOL AND DIE");/* Firm line */
strcpy(parm.iadl3, "" );/* Secondary or extra line*/
strcpy(parm.iadl1, "323 S 152ND ST" );/* Primary address line */
strcpy(parm.iprurb, "" );/* Puerto Rico specific */
strcpy(parm.ictyi, "OMAHA, NE 68154" );/* City, State, ZIP */

/* request address inquiry */
z4adrinq(&parm);

/* request SuiteLink inquiry */
z4SLNKQuery(&parm);

/* if a response found (either single or default) */
if(parm.retcc==Z4_SINGLE || parm.retcc==Z4_DEFAULT)
{
    printf("Found response.\n");
    printf("Name: %s\n", parm.dadl2);
    printf("S Addr: %s\n", parm.dadl3);
    printf("Addr: %s\n", parm.dadl1);
    printf("PRUrb: %s\n", parm.dprurb);
    printf("City: %s\n", parm.dctya);
    printf("ST: %s\n", parm.dstaa);
    printf("ZIP: %s\n", parm.zipc);
    printf("Addon: %s\n", parm.addon);
    printf("DPBC: %s\n", parm.dpbc);
    printf("Pre Dir: %s\n", parm.stack[0].pre_dir);{
    printf("Str Name: %s\n", parm.stack[0].str_name);
    printf("Suffix: %s\n", parm.stack[0].suffix);
    printf("Post Dir: %s\n", parm.stack[0].post_dir);
    printf("Lacs Ind: %c\n", parm.stack[0].lacs_status);
}
}

/* close The USPS Address Matching System */
z4close();

exit(0)
}

```

Abbreviated Street Address Query

The `z4ABSQuery()` function performs an abbreviated address lookup on a returned `z4adrinq()` address. This optional call identifies a street address line exceeding thirty (30) characters and returns a thirty (30) character or less abbreviation.

Syntax

```
#include "zip4.h"
int Z4FUNC z4ABSQuery (ZIP4_PARM* pZip4, TAbbrSt* pAbbrSt );
```

Input

`pZip4` A pointer to the `ZIP4_PARM` that was used during the preceding `z4adrinq()` call.
`pAbbrSt` A pointer to an empty `TAbbrSt` structure where the output will be stored.

Output

`pAbbrSt` will be populated with the abbreviated street information.

Return

`TRUE` – A successful lookup was completed.
`FALSE` – An abbreviated address lookup was unsuccessful.

Example

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "zip4.h"

ZIP4_PARM parm;
TAbbrSt            pStreet[1];

int main(int argc, char** argv)
{
    Z4OPEN_PARM openparm;
    memset (&openparm, 0, sizeof(openparm));
    /* ... Populate openparm ... */
    /* open the USPS Address Matching System */
    if (z4opencfg(&openparm) != 0)
    {
        printf("The USPS Address Matching System failed to open");
        /* Always call z4close() even on open failure */
        z4close();
        exit(5);
    }

    /* load input address parameters */
    memset(&parm, 0, sizeof(parm));
    strcpy(parm.iadl2, "ACME TOOL AND DIE" );/* Firm line */
    strcpy(parm.iadl3, "UNIT 1" );/* Secondary or extra line*/
    strcpy(parm.iadl1, "100 TYNGSBOROUGH BUSINESS PK DR"); /* Primary */
    strcpy(parm.iprurb, "" );/* Puerto Rico specific */
    strcpy(parm.ictyi, "TYNGSBORO, MA 01879");/* City, State, ZIP */
```

```

/* request address inquiry */
z4adrinq(&parm);

/* request abbreviated inquiry */
z4ABSQuery(&parm, pStreet);

/* if a response found (either single or default) */
if(parm.retcc==Z4_SINGLE || parm.retcc==Z4_DEFAULT)
{
    printf("Found response.\n");
    printf("Name:          %s\n", parm.dad12);
    printf("S Addr:         %s\n", parm.dad13);
    printf("Addr:           %s\n", pStreet->szAddress);
    printf("PRUrb:          %s\n", parm.dprurb);
    printf("City:           %s\n", parm.dctya);
    printf("ST:             %s\n", parm.dstaa);
    printf("ZIP:            %s\n", parm.zipc);
    printf("Addon:          %s\n", parm.addon);
    printf("DPBC:           %s\n", parm.dpbc);
    printf("Pre Dir:        %s\n", parm.stack[0].pre_dir);{
    printf("Str Name: %s\n", parm.stack[0].str_name);
    printf("Suffix:         %s\n", parm.stack[0].suffix);
    printf("Post Dir: %s\n", parm.stack[0].post_dir);
    printf("Lacs Ind: %c\n", parm.stack[0].lacs_status);
}
}

/* close The USPS Address Matching System */
z4close();

exit(0)
}

```

Section 4: Footnote Flags

A ZIP CODE CORRECTED

The address was found to have a different 5-digit ZIP Code than given in the submitted list. The correct ZIP Code is shown in the output address.

B CITY / STATE SPELLING CORRECTED

The spelling of the city name and/or state abbreviation in the submitted address was found to be different than the standard spelling. The standard spelling of the city name and state abbreviation are shown in the output address.

C INVALID CITY / STATE / ZIP

The ZIP Code in the submitted address could not be found because neither a valid city, state, nor valid 5-digit ZIP Code was present. It is also recommended that the requestor check the submitted address for accuracy.

D NO ZIP+4 ASSIGNED

This is a record listed by the United States Postal Service on the national ZIP+4 file as a non-deliverable location. It is recommended that the requestor verify the accuracy of the submitted address.

E ZIP CODE ASSIGNED FOR MULTIPLE RESPONSE

Multiple records were returned, but each shares the same 5-digit ZIP Code.

F ADDRESS COULD NOT BE FOUND IN THE NATIONAL DIRECTORY FILE DATABASE

The address, exactly as submitted, could not be found in the city, state, or ZIP Code provided. It is also recommended that the requestor check the submitted address for accuracy. For example, the street address line may be abbreviated excessively and may not be fully recognizable.

G INFORMATION IN FIRM LINE USED FOR MATCHING

Information in the firm line was determined to be a part of the address. It was moved out of the firm line and incorporated into the address line.

H MISSING SECONDARY NUMBER

ZIP+4 information indicates this address is a building. The address as submitted does not contain an apartment/suite number. It is recommended that the requestor check the submitted address and add the missing apartment or suite number to ensure the correct Delivery Point Barcode (DPBC).

I INSUFFICIENT / INCORRECT ADDRESS DATA

More than one ZIP+4 Code was found to satisfy the address as submitted. The submitted address did not contain sufficiently complete or correct data to determine a single ZIP+4 Code. It is recommended that the requestor check the address for accuracy and completeness. For example, firm name, or institution name, doctor's name, suite number, apartment number, box number, floor number, etc. may be missing or incorrect. Also pre-directional or post-directional indicators (North = N, South = S, East = E, West = W, etc.) and/or street suffixes (Street = ST, Avenue = AVE, Road = RD, Circle = CIR, etc.) may be missing or incorrect.

J DUAL ADDRESS

The input contained two addresses. For example: 123 MAIN ST PO BOX 99.

K MULTIPLE RESPONSE DUE TO CARDINAL RULE

CASS rule does not allow a match when the cardinal point of a directional changes more than 90%.

L ADDRESS COMPONENT CHANGED

An address component (i.e., directional or suffix only) was added, changed, or deleted in order to achieve a match.

M STREET NAME CHANGED

The spelling of the street name was changed in order to achieve a match.

N ADDRESS STANDARDIZED

The delivery address was standardized. For example, if STREET was in the delivery address, the system will return ST as its standard spelling.

O LOWEST +4 TIE-BREAKER

More than one ZIP+4 Code was found to satisfy the address as submitted. The lowest ZIP +4 addon may be used to break the tie between the records.

P BETTER ADDRESS EXISTS

The delivery address is matchable, but is known by another (preferred) name. For example, in New York, NY, AVENUE OF THE AMERICAS is also known as 6TH AVE. An inquiry using a delivery address of 55 AVE OF THE AMERICAS would be flagged with a Footnote Flag P.

Q UNIQUE ZIP CODE MATCH

Match to an address with a unique ZIP Code.

R NO MATCH DUE TO EWS

The delivery address is matchable, but the EWS file indicates that an exact match will be available soon.

S INCORRECT SECONDARY ADDRESS

The secondary information (i.e., floor, suite, apartment, or box number) does not match that on the national ZIP+4 file. This secondary information, although present on the input address, was not valid in the range found on the national ZIP+4 file.

T MULTIPLE RESPONSE DUE TO MAGNET STREET SYNDROME

The search resulted in a single response; however, the record matched was flagged as having magnet street syndrome. "Whenever an input address has a single suffix word or a single directional word as the street name, or whenever the ZIP+4 File records being matched to have a single suffix word or a single directional word as the street name field, then an exact match between the street, suffix and/or post-directional and the same components on the ZIP+4 File must occur before a match can be made. Adding, changing or deleting a component from the input address to obtain a match to a ZIP+4 record will be considered incorrect." Instead of returning a "no match" in this situation a multiple response is returned to allow access the candidate record.

U UNOFFICIAL POST OFFICE NAME

The city or post office name in the submitted address is not recognized by the United States Postal Service as an official last line name (preferred city name), and is not acceptable as an alternate name. This does denote an error and the preferred city name will be provided as output.

- V UNVERIFIABLE CITY / STATE**
The city and state in the submitted address could not be verified as corresponding to the given 5-digit ZIP Code. This comment does not necessarily denote an error; however, it is recommended that the requestor check the city and state in the submitted address for accuracy.
- W INVALID DELIVERY ADDRESS**
The input address record contains a delivery address other than a PO BOX, General Delivery, or Postmaster with a 5-digit ZIP Code that is identified as a “small town default.” The United States Postal Service does not provide street delivery for this ZIP Code. The United States Postal Service requires use of a PO BOX, General Delivery, or Postmaster for delivery within this ZIP Code.
- X UNIQUE ZIP CODE GENERATED**
Default match inside a unique ZIP Code.
- Y MILITARY MATCH**
Match made to a record with a military ZIP Code.
- Z MATCH MADE USING THE ZIPMOVE PRODUCT DATA**
The ZIPMOVE product shows which ZIP + 4 records have moved from one ZIP Code to another. If an input address matches to a ZIP + 4 record which the ZIPMOVE product indicates as having moved, the search is performed again in the new ZIP Code.

Section 5: Record Types

F FIRM

This is a match to a Firm Record, which is the finest level of match available for an address.

G GENERAL DELIVERY

This is a match to a General Delivery record.

H BUILDING / APARTMENT

This is a match to a Building or Apartment record.

P POST OFFICE BOX

This is a match to a Post Office Box.

R RURAL ROUTE or HIGHWAY CONTRACT

This is a match to either a Rural Route or a Highway Contract record, both of which may have associated Box Number ranges.

S STREET RECORD

This is a match to a Street record containing a valid primary number range.

Section 6: Return Codes

10 **INVALID DUAL ADDRESS**

Information presented could not be processed in current format. Corrective action is needed. Be sure that the address line components are correct. For example, the input address line may contain more than one delivery address.

11 **INVALID CITY/ST/ZIP**

The ZIP Code in the submitted address could not be found because neither a valid city, state, nor valid 5-digit ZIP Code was present. Corrective action is needed. It is also recommended that the requestor check the submitted address for accuracy.

12 **INVALID STATE**

The state in the submitted address is invalid. Corrective action is needed. It is also recommended that the requestor check the submitted address for accuracy.

13 **INVALID CITY**

The city in the submitted address is invalid. Corrective action is needed. It is also recommended that the requestor check the submitted address for accuracy.

21 **NOT FOUND**

The address, exactly as submitted, could not be found in the national ZIP+4 file. It is recommended that the requestor check the submitted address for accuracy. For example, the street address line may be abbreviated excessively and may not be fully recognizable.

22 **MULTIPLE RESPONSE**

More than one ZIP+4 Code was found to satisfy the address submitted. The submitted address did not contain sufficiently complete or correct data to determine a single ZIP+4 Code. It is recommended that the requestor check the address for accuracy and completeness. Address elements may be missing

31 **EXACT MATCH.**

Single response based on input information. No corrective action is needed since an exact match was found in the national ZIP+4 file.

32 **DEFAULT MATCH**

A match was made to a default record in the national ZIP+4 file. A more specific match may be available if a secondary number (i.e., apartment, suite, etc.) exists.

Appendix A: Interface Definition

```

#ifndef ZIP4_H                /* avoid redefinition */
#define ZIP4_H
/*****
/* This record describes an address record. The record format is the same as
/* the USPS ZIP+4 File. Please see the USPS Address Information Products
/* Technical Guide for information on this record.
/* NOTE: All 'char' array fields contain an extra byte (+1) for the null
/* terminator.
*****/
typedef struct
{
    char    detail_code;        /* copyright detail code */
    char    zip_code[5+1];      /* zip code */
    char    update_key[10+1];   /* update key number */
    char    action_code;       /* action code */
    char    rec_type;          /* record type */
    char    carr_rt[4+1];       /* carrier route */
    char    pre_dir[2+1];       /* pre-direction abbrev */
    char    str_name[28+1];     /* street name */
    char    suffix[4+1];       /* suffix abbrev */
    char    post_dir[2+1];      /* post-direction abbrev */
    char    prim_low[10+1];     /* primary low range */
    char    prim_high[10+1];    /* primary high range */
    char    prim_code;         /* primary even odd code */
    char    sec_name[40+1];     /* bldg or firm name */
    char    unit[4+1];         /* secondary abbreviation */
    char    sec_low[8+1];       /* secondary low range */
    char    sec_high[8+1];     /* secondary high range */
    char    sec_code;         /* secondary even odd code */
    char    addon_low[4+1];     /* add on low */
    char    addon_high[4+1];    /* add on high */
    char    base_alt_code;      /* base alternate code */
    char    lacs_status;       /* LACS converted status */
    char    finance[6+1];      /* finance code */
    char    state_abbrev[2+1];   /* state abbreviation (not filled) */
    char    county_no[3+1];     /* county number */
    char    congress_dist[2+1]; /* congressional district */
    char    municipality[6+1];  /* municip. city/state key (not filled) */
    char    urbanization[6+1];  /* urb. city/state key */
    char    last_line[6+1];     /* last line city/state key */
} ADDR_REC;

/* NOTE: The GovtBldgInd (Government Building Indicator) field is not
/* available in the ADDR_REC structure.

```

Appendix A: Interface Definition

```

/*****
/*   This record describes a city/state record. The record format is the same */
/*   as the USPS City State File. Please see the USPS Address Information   */
/*   Products Technical Guide for information on this record.                 */
/*   NOTE: All 'char' array fields contain an extra byte (+1) for the null   */
/*   terminator.                                                              */
/*****/
typedef struct
{
    char  detail_code;                /* copyright detail code          */
    char  zip_code[5+1];              /* zip code                       */
    char  city_key[6+1];              /* city/state key                 */
    char  zip_class_code;             /* zip classification code        */
                                        /* blank = non-unique zip        */
                                        /* M=APO/FPO military zip        */
                                        /* P=PO BOX zip                  */
                                        /* U=Unique zip                  */
    char  city_name[28+1];            /* city/state name               */
    char  city_abbrev[13+1];          /* city/state name abbrev        */
    char  facility_cd;                /* facility code                  */
                                        /* A=Airport mail facility       */
                                        /* B=Branch                       */
                                        /* C=Community post office       */
                                        /* D=Area distrib. center        */
                                        /* E=Sect. center facility       */
                                        /* F=General distrib. center    */
                                        /* G=General mail facility       */
                                        /* K=Bulk mail center            */
                                        /* M=Money order unit           */
                                        /* N=Non-postal name             */
                                        /* community name,               */
                                        /* former postal facility,       */
                                        /* or place name                 */
                                        /* P=Post office                 */
                                        /* S=Station                     */
                                        /* U=Urbanization                */
    char  mailing_name_ind; /* mailing name indicator         */
                                        /* Y=Mailing name                 */
                                        /* N=Non-mailing name            */
    char  last_line_num[6+1];         /* preferred last line key       */
    char  last_line_name[28+1];       /* preferred city name           */
}

```

Appendix A: Interface Definition

```

char  city_delv_ind;          /* city delivery indicator          */
                                /* Y=Office has city delivery */
                                /* carrier routes                */
                                /* N=Office does not have city*/
                                /* delivery carrier routes      */
char  auto_zone_ind;        /* automated zone indicator        */
                                /* A=CR Sort Rates Apply         */
                                /* Merge Allowed                 */
                                /* B=CR Sort Rates Apply         */
                                /* Merge Not Allowed            */
                                /* C=CR Sort Rates Do Not Apply  */
                                /* Merge Allowed                 */
                                /* D=CR Sort Rates Do Not Apply  */
                                /* Merge Not Allowed            */
char  unique_zip_ind;      /* unique zip name indicator       */
                                /* Y=Unique zip name            */
                                /* blank=not applicable         */
char  finance[6+1];        /* finance code                    */
char  state_abbrev[2+1];   /* state abbreviation              */
char  county_no[3+1];     /* county number                   */
char  county_name[25+1];   /* county name                     */
} CITY_REC;

/*****
/* Parameter list for z4adring() and z4xrfinq() calls. Reserved fields are
/* for future use, do not access these fields. Size of this record can not
/* be changed.
/* NOTE: Only fields containing +1 in the length are null terminated.
*****/
typedef struct
{
    char  rsvd0[4];          /* reserve fore future use        */
    char  iadl1[50+1];      /* input delivery address         */
    char  iadl2[50+1];      /* input firm name                */
    char  ictyi[50+1];      /* input city                     */
    char  istai[2+1]; /* input state                    */
    char  izipc[10+1];      /* input ZIP+4 code              */
    char  iprurb[28+1];     /* input urbanization name       */
    char  iadl3[50+1];      /* input second address line     */
    char  iddpv1[12+1];    /* reserved for future use       */
    char  rsvd1[85];        /* reserved for future use       */

    char  dadl3[50+1];      /* standardized 2nd delivery address*/
    char  dadl1[50+1];      /* standardized delivery address */
    char  dadl2[50+1];      /* standardized firm name        */
    char  dlast[50+1];     /* standardized city/state/zip*/
    char  dprurb[28+1];    /* output PR urbanization name*/
    char  dctys[28+1];     /* main post office city        */
    char  dstas[2+1]; /* main post office state        */
    char  dctya[28+1];     /* standardized city            */
    char  abcty[13+1];     /* standardized city abbreviation */
    char  dstaa[2+1]; /* standardized state            */
    char  zipc[5+1];       /* 5-digit zip code             */
    char  addon[4+1]; /* ZIP+4 addon code             */
    char  dpbc[3+1];       /* delivery point bar code      */
    char  cris[4+1];       /* carrier route                 */
    char  county[3+1];     /* FIPS county code             */
    short respn;          /* number of returned responses */
    char  retcc;           /* return code                   */
    char  adrkey[12]; /* address key (for indexing) */
    char  auto_zone_ind; /* A, B, C or D                 */
    char  elot_num[4+1]; /* eLOT Number                   */
}

```

Appendix A: Interface Definition

```

char  elot_code;          /* eLOT Ascending/Descending Flag */
char  llk_rc[2+1]; /* LACS Link Return Code */
char  llk_ind;           /* LACS Link Indicator */
char  misc[128+1];      /* line for unused input data */
char  rsvd2[20];        /* Reserved for Future Use */

                                /****** parsed input data***** */
char  ppnum[10+1];      /* Primary Number */
char  psnum[8+1]; /* Secondary Number */
char  prote[3+1]; /* Rural Route Number */
char  punit[4+1]; /* Secondary Number Unit */
char  ppre1[2+1]; /* First or Left Pre-direction*/
char  ppre2[2+1]; /* Second or Right Pre-direction */
char  psuf1[4+1]; /* First or Left Suffix */
char  psuf2[4+1]; /* Second or Right Suffix */
char  ppst1[2+1]; /* First or Left Post-direction */
char  ppst2[2+1]; /* Second or Right Post-direction */
char  ppnam[28+1];     /* Primary Name */
char  mpnum[10+1];    /* Matched primary number. */
char  msnum[8+1]; /* Matched secondary number */
char  pmb[3+1];      /* PMB Unit Designator */
char  pmbnum[8+1];   /* PMB Number */
char  mlev1;        /* Reserved Use */

char  footnotes[32+1]; /* Reserved for Future Use */
char  stelinkfoot[3+1]; /* suite link footnote */
char  punit2[4+1]; /* second or right Secondary Unit */
char  psnum2[8+1]; /* second or right secondary number */
char  rsvd3[10]; /* Reserved for future use */

struct {
                                /****** footnotes***** */
char  a; /* zip corrected */
char  b; /* city/state corrected */
char  c; /* invalid city/state/zip */
char  d; /* no zip assigned */
char  e; /* ZIP assigned for mult response */
char  f; /* no zip available */
char  g; /* part of firm moved to address */

```

Appendix A: Interface Definition

```

char h;          /* secondary number missing          */
char i;          /* insufficient/incorrect data        */
char j;          /* dual input                          */
char k;          /* reserved for future use"          */
char l;          /* del addr component add/del/chg     */
char m;          /* street name spelling changed       */
char n;          /* delivery addr was standardized     */
char o;          /* multi break tie with lowest +4    */
char p;          /* better delivery addr exists        */
char q;          /* Unique ZIP Code                    */
char r;          /* no match caused by EWS             */
char s;          /* invalid secondary number           */
char t;          /* magnet street                      */
char u;          /* unofficial PO name                 */
char v;          /* unverifiable city/state           */
char w;          /* small town default                 */
char x;          /* unique ZIP Code generated          */
char y;          /* Military Match                     */
char z;          /* ZIP Move Match                     */
char f0;         /* reserved for future use            */
char f1;         /* reserved for future use            */
char f2;         /* reserved for future use            */
char f3;         /* reserved for future use            */
char f4;         /* reserved for future use            */
char f5;         /* reserved for future use            */
} foot;

ADDR_REC stack[10]; /******record stack******/
char rsvd4[194];   /* reserved for future use          */
} ZIP4_PARM;

/*****
/* Parameter list for z4getzip()
/* NOTE: Only fields containing +1 in the length are null terminated.
*****/
typedef struct
{
    char input_cityst[50+1];
    char output_cityst[50+1];
    char low_zipcode[5+1];
    char high_zipcode[5+1];
    char finance_num[6+1];
} GET_ZIPCODE_STRUCT;

```

```

/*****
/*          ABBREVIATED STREET RECORD          */
/*
/*   Parameter list for z4ABSQuerySTD()        */
/*   NOTE:  Fields names with a leading "sz" are null terminated.  */
/*
/*****
typedef struct tagNationalDirectoryFileZip4DetailAbbreviated
{
    char  psDetailCode[1];          /* COPYRIGHT DETAIL CODE          */
    char  szZipcode[6];            /* ZIP CODE                        */
    char  szUpdateKey[11];        /* UPDATE KEY NUMBER              */
    char  psActionCode[1];        /* ACTION CODE                     */
    char  psRecordType[1];        /* RECORD TYPE                     */
    char  szCarrierRt[5];         /* CARRIER ROUTE                  */
    char  szPreDir[3];            /* PRE-DIRECTIONAL ABBREVIATED    */
    char  szStreetName[29];       /* STREET NAME                      */
    char  szSuffix[5];            /* SUFFIX ABBREVIATED             */
    char  szPostDir[3];          /* POST-DIRECTIONAL ABBREVIATED   */
    char  szPrimaryL[11];         /* PRIMARY LOW RANGE               */
    char  szPrimaryH[11];         /* PRIMARY HIGH RANGE              */
    char  psPrimarCode[1];        /* EVEN/ODD/BOTH CODE (PRIMARY NUMBER) */
    char  szFirm[41];            /* BUILDING/FIRM NAME              */
    char  szUnit[5];             /* UNIT DESIGNATOR ABBREVIATED    */
    char  szSecondaryL[9];        /* SECONDARY LOW RANGE             */
    char  szSecondaryH[9];        /* SECONDARY HIGH RANGE            */
    char  psSecondaryCode[1];     /* EVEN/ODD/BOTH CODE (SECONDARY NUMBER) */
    char  szAddonL[5];           /* ADD ON LOW RANGE                */
    char  szAddonH[5];           /* ADD ON HIGH RANGE               */
    char  psBaseAltCode[1];       /* BASE/ALTERNATE CODE             */
    char  psLACS[1];             /* LACS CONVERTED STATUS           */
    char  szFinance[7];          /* FINANCE NUMBER                  */
    char  szState[3];            /* STATE ABBREVIATED              (NOT FILLED) */
    char  szCountyNumber[4];      /* COUNTY NUMBER                   */
    char  szCongressDist[3];     /* CONGRESSIONAL DISTRICT         */
    char  szMunicipality[7];     /* MUNICIPALITY CITY/STATE KEY (NOT FILLED) */
    char  szUrbanization[7];     /* URBANIZATION CITY/STATE KEY    */
    char  szLastLineKey[7];      /* LAST LINE CITY/STATE KEY       */
    char  szAddress[51];         /* STANDARDIZED DELIVERY ADDRESS   */
} TAbbrSt, *TPAbbrSt;

/*****
/* Error Codes for the iErrorCode variable inside the Z4_ERROR structure */
/*****
#define ERROR_FILE_OPEN 1        /* Error opening a file          */
#define ERROR_FILE_READ 2       /* Error reading a file          */
#define ERROR_FILE_WRITE 3      /* Error writing to a file        */
#define ERROR_FILE_FIND 4       /* Error finding a file          */
#define ERROR_FILE_EXPIRE 5     /* AMS library has expired       */
#define ERROR_FILE_SYNC 6       /* AMS Database files out of sync */
#define ERROR_SECURITY 7        /* AMS Security Error            */

```

Appendix A: Interface Definition

```

#define FILE_ID_CONFIG 1 /* Configuration File */
#define FILE_ID_ZADRFLE 2 /* zadrfile.dat */
#define FILE_ID_ZADRFLENDX 3 /* zadrfile.idx */
#define FILE_ID_CTYSTATE 4
#define FILE_ID_CTYSTATENDX 5
#define FILE_ID_ZIP5FLE 6
#define FILE_ID_ZIP5FLENDX 7
#define FILE_ID_ZXREFDTL 8
#define FILE_ID_ELTRVFLE 9
#define FILE_ID_ELTRVFLENDX 10
#define FILE_ID_EWS 11
#define FILE_ID_SYSTEM 12
#define FILE_ID_LIBRARY 13
#define FILE_ID_KEYMANLIB 14
#define FILE_ID_DATABASE 15
#define FILE_ID_LLK 16
#define FILE_ID_DPV 17
#define FILE_ID_FNSN 18
#define FILE_ID_STELNK 19
#define FILE_ID_ABBRST 20

/*****
/* Parameter list for z4geterror() */
/* NOTE: Only fields containing +1 in the length are null terminated */
/*****/
typedef struct
{
    int iErrorCode; /* Error Code */
    char strErrorMessage[100+1]; /* Error Message */
    int iFileCode; /* File Code */
    char strFileName[26+1]; /* File Name */
    char strDiagnostics[300+1]; /* Diagnostic Message */
} Z4_ERROR;

/*****
/* Paramter list for z4getenv() */
/* NOTE: Only fields containing +1 in length are null terminated */
/*****/
typedef struct
{
    char strConfigFile[300+1];
    char address1[300+1]; /*Contains the full path of the ZADRFLE.DAT file */
    char addrindex[300+1]; /*Contains the full path of the ZADRFLE.NDX file */
    char cdrom[300+1]; /*Contains the drive letter of the CD-ROM drive that*/
    /*Contains the ZIP+4/carrier route data;may be blank*/
    char citystate[300+1]; /*Contains the full path of the following files: */
    /*CTYSTATE.DAT - CITYSTATE.NDX */
    /*ZIP5FLE.DAT - ZIP5FLE.NDX */
    char crossref[300+1]; /*Contains full path of the ZXREFDTL.DAT file */
    char system[300+1]; /*Contains the full path of the Z4CXLOG.DAT file */
    char elot[300+1]; /*Contains the full path of the eltrvfle.dat file */
    char elotindex[300+1]; /*Contains the full path of the eltrvfle.ndx file */
    char llkpath[300+1] /*Contains the full path of the LACS Link files */
    char ewspath[300+1]; /*Contains the full path of the ews.txt file */
    char fnsnpath[300+1]; /*Contains the full path of the fnsn.* files */
    char stelknpath[300+1]; /* Path to STELNK.* files */
    char abrstpath[300+1]; /* Path to ABBRST.* files */
    char rsvd1[1208]; /* reserved for future use */
    char stelknflag; /* STELNK flag (Y enables else disabled) */
    char abrstflag; /* ABBRST flag (Y enables else disabled) */
    char ewsflag; /* EWS flag (Y enables else disabled) */
    char elotflag; /* eLot flag (Y enables else disabled) */
    char llkflag; /* LACS Link flag (Y enables else disabled) */
    char dpvflag; /* DPV flag (Y enables else disabled) */
}

```

```
}Z4_ENV;
```

```

/*****
/* Parameter list for z4opencfg()
/* NOTE: Only fields containing +1 in the length are null terminated.
/*****
/* Use of this structure will replace a physical copy of the configuration
/* file on the hard drive
*/

typedef struct
{
    char *address1; /*Contains the full path of the ZADRFLE.DAT file */
    char *addrindex; /*Contains the full path of the ZADRFLE.NDX file */
    char *cdrom; /*Contains the drive letter of the CD-ROM drive that*/
                /*contains the ZIP+4/carrier route data;may be blank*/
    char *citystate; /*Contains the full path of the following files: */
                /*CTYSTATE.DAT - CTYSTATE.NDX */
                /*ZIP5FILE.DAT - ZIP5FLE.NDX */
    char *crossref; /*Contains the full path of the ZXREFDTL.DAT file */
    char *system; /*Contains the full path of the Z4CXLOG.DAT file */
    char *elot; /*Contains the full path of the ELTRVFLE.DAT file */
    char *elotindex; /*Contains the full path of the ELTRVFLE.ND file */
    char *llkpath; /*Contains the full path of the LACS Link files */
    char *ewspath; /*Contains the full path of the EWS.TXT file */
    char *dpvpath; /*Contains the full path of the dpv files */
    char *fnsnpath; /*Contains the full path of the fnsn.* files */
    char* stelkpath; /* Path to the suite link files */
    char* abrstpath; /* Path to the abbreviated street name files */
    char rsvd[116]; /* reserved for future use */
}CONFIG_PARM;

typedef struct
{
    char rsvd1[50]; /*reserved for future use */
    short status; /**1 - Used value point to by fname */
                /**2 - Used values in CONFIG_PARM */
                /**9 - No values found. Search for z4config.dat */
    char *fname; /**pointer to a NULL terminated string that */
                /*contains the full path and filename for a custom */
                /*config file. If fname contains a leading space */
                /*or NULL then it is ignored and the CONFIG_PARM */
                /*is evaluated for path names */
    CONFIG_PARM config; /*Contains the path name for the config file */
    char ewsflag; /**Y Enabled EWS else Disable EWS */
    char elotflag; /**Y Enables LOT else Disable eLOT */
    char llkflag; /**Y Enables LACS Link else disable LACS Link */
    char dpvflag; /**Y Enables DPV else disable DPV */
    char systemflag; /**Indicates open option */
    char rtsw[15+1]; /* Internal use */
    char dpvtypeflag; /* Future use */
    char stelkflag; /* Y Enables STELNK else Disable STELNK */
    char abrstflag; /* Y Enables ABRST else Disable ABRST */
    char rsvd2[492]; /* reserved for future use */
}Z4OPEN_PARM

```

```

/*****
/*Z4OPEN_PARM.status values for z4opencfg()
*****/

#define      Z4_FNAME      1 /* Used the value in fname as the path and filename */
#define      Z4_CONFIG     2 /* Used the paths in the CONFIG_PARM structure*/
#define      Z4_SEARCH     9 /* Used neither, searched for z4config.dat          */

/*****
/*      Return Codes for z4adrinq() and z4xrfinq() calls
*****/
#define      Z4_INVADDR    10 /* invalid address
#define      Z4_INVZIP     11 /* invalid ZIP Code
#define      Z4_INVSTATE   12 /* invalid state code
#define      Z4_INVCITY    13 /* invalid city
#define      Z4_NOTFND     21 /* address not found
#define      Z4_MULTIPLE   22 /* multiple response - no default
#define      Z4_SINGLE     31 /* single response - exact match
#define      Z4_DEFAULT    32 /* default response

/*****
/*      Function prototypes for the ZIP+4 retrieval engine.
*****/
#if defined(OS2_32)
#define Z4FUNC
#elif defined(WIN32)
#define Z4FUNC _cdecl
#elif defined(_WINDOWS) || defined(_WINDLL)
#define Z4FUNC __far __pascal __export
#elif defined(OS2)
#define Z4FUNC _far _pascal _loadds _export
#elif defined(_MAC)
#define Z4FUNC_
#elif defined(ANSI_STRICT) || defined(UNIX) || defined(I370)
#define Z4FUNC
#else
#define Z4FUNC _cdecl
#endif

int  Z4FUNC z4remove(void); /* terminate the retrieval engine
int  Z4FUNC z4open(void); /* open the retrieval engine for use
int  Z4FUNC z4opencfg(Z4OPEN_PARM *); /*open with custom parameters
int  Z4FUNC z4close(void); /* close the retrieval engine
int  Z4FUNC z4abort(void); /* abort the current inquiry
int  Z4FUNC z4adrinq(ZIP4_PARM *); /* address inquiry
int  Z4FUNC z4scroll(ZIP4_PARM *); /* address inquiry
int  Z4FUNC z4adrkey(ZIP4_PARM *); /* address key (for indexing)
int  Z4FUNC z4xrfinq(ZIP4_PARM *); /* nine digit cross reference inquiry
int  Z4FUNC z4xrfinq11(ZIP4_PARM*); /* eleven digit cross reference inquiry
int  Z4FUNC z4adrstd(ZIP4_PARM *, int); /* address standardization

```

Appendix A: Interface Definition

```
int     Z4FUNC  z4ctyget(CITY_REC *, void *);/* get first city for a state      */
int     Z4FUNC  z4ctynxt(CITY_REC *);      /* get next city for a state      */
int     Z4FUNC  z4adrget(ADDR_REC *, void *);/* get first address for a fin. no */
int     Z4FUNC  z4adrnxt(ADDR_REC *);      /* get next address for a fin. no */
int     Z4FUNC  z4adrprv(ADDR_REC *);      /* get previous addrss for a fin. no*/
int     Z4FUNC  z4date(char *);            /* get date of ZIP+4 database     */
int     Z4FUNC  z4GetDataExpireDays(void); /* number of days until data expire */
int     Z4FUNC  z4GetCodeExpireDays(void); /* number of days until code expire */
int     Z4FUNC  z4expire(void);            /* Deprecated. Use GetDataExpireDays() */
int     Z4FUNC  z4getzip(GET_ZIPCODE_STRUCT*);/* get zip code range for cityst */
int     Z4FUNC  z4ver(char *);             /* get the version of the API code */
int     Z4FUNC  z4geterror(Z4_ERROR *);/* get the last error msg and code */
int     Z4FUNC  z4getenv(Z4_ENV *);        /* get the environment for AMS     */
int     Z4FUNC  z4lline(ZIP4_PARM *, char *);/* Validate Last Line             */
const char* Z4FUNC  z4LLkGetKey(void);
int     Z4FUNC  z4LLkIsDisabled(void);
int     Z4FUNC  z4LLkSetKey(const char* szKey);

const char* Z4FUNC  z4SLNKGetDate(int);      /* Gets date associated with a table */
long      Z4FUNC  z4SLNKGetError(void);      /* Suite Link error code            */
const char* Z4FUNC  z4SLNKGetErrorMsg(void);/* Suite Link error message         */
int       Z4FUNC  z4SLNKQuery(ZIP4_PARM*);/* Performs a Suite Link lookup     */
int       Z4FUNC  z4ABSQuery(ZIP4_PARM*, TABbrSt*); /* Performs abbreviated
street address lookup */

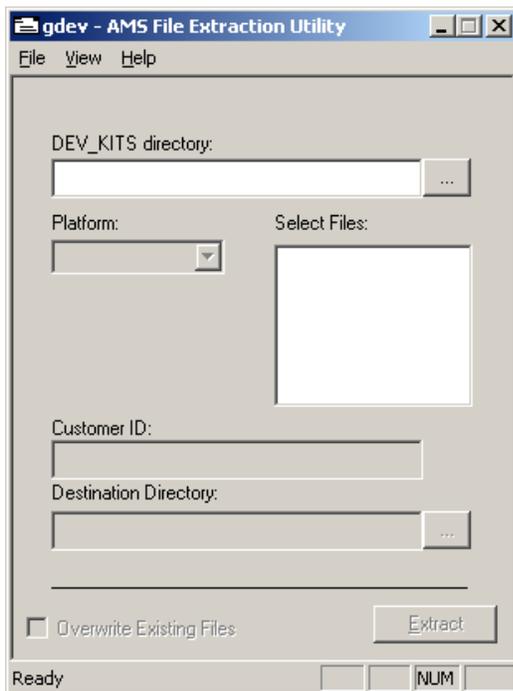
#endif /* ZIP4_H */
```

Appendix B: GDEV Application

GDEV is a GUI Windows application that provides the capability to unencrypt any of the developer kits from the AMS disc.

This application is located on the AMS disc in the dev_kits sub-directory. (gdev.exe)

If GDEV is launched directly from the disc it will automatically load information from that disc. Otherwise, GDEV will not display any information and you will have to manually select a DEV_KITS directory.



1. Select “File->Select DEV_KIT directory” to tell GDEV where the developer kits are located.
2. Select the “Platform” that you want to unencrypt.
3. Select the file(s) that you want to unencrypt.
4. Enter your customer ID.
5. Enter or Select the directory where the unencrypted files should be placed.
6. Click the [Extract] button.

Note: If the “Destination Directory” already contains the selected files then you must also select the “Overwrite Existing Files” checkbox or the unencryption process will fail.

Appendix C: DPV®

The United States Postal Service® (USPS®) has developed a new technology product that will help mailers validate the accuracy of their address information, right down to the physical delivery point. Mailers will be able to identify individual addresses within a mailing list that are potentially undeliverable-as-addressed due to an addressing deficiency. This new technology is now available through the current address matching API. Following is a layout and example usage of the DPV®/DSF2® interface made available through the address matching API.

Error Values

Constants used to identify errors returned from `z4DpvSetPath ()`:

```
#define ERROR_INVALID_AMS_STATUS      -1
#define ERROR_INVALID_DPV_HNDL      -2
#define ERROR_UNKNOWN_DPV_ID        -4
#define ERROR_LD_LIBRARY_FAIL        -5
#define ERROR_OPEN_DPVTBL_FAIL      -6
#define ERROR_INVALID_MATCH_LVL      -7
```

Error Codes

Constants used to identify dpv errors returned from `z4GetLastErrorCode ()`:

```
#define SUCCESSFUL_DPV                0    ( NO ERROR )
#define UNDEFINED_DPV                 1    ( UNKNOWN ERROR)
#define INVALID_HND_DPV               2    ( INVALID HANDLE )
#define INVALID_ID_DPV                3    ( UNKNOWN ID FOR THE OPERATION)
#define NULL_VALUE_DPV                4    ( ENCOUNTER A NULL VALUE)
#define ACCESS_DENIED_DPV             5    ( ACCESS ATTEMPT ON DEVICE DENIED)
#define OPEN_FAILED_DPV               6    ( OPEN ATTEMPT ON DEVICE FAILED)
#define READ_FAILED_DPV               7    ( READ ATTEMPT ON DEVICE FAILED)
#define WRITE_FAILED_DPV8             ( WRITE ATTEMPT TO DEVICE FAILED)
#define SEEK_FAILED_DPV               9    ( SEEK ATTEMPT ON DEVICE FAILED)
#define UNAVAILABLE_DATA_DPV         10    ( MISSING DATA NEEDED FOR OPERATION)
#define LOW_MEMORY_DPV                11    ( CAN NOT ALLOCATE ENOUGH MEMORY)
#define EXPIRATION_DPV               12    ( DATA AND LIBRARY ARE NOT COMPATIBLE)
#define SYNCHRONIZE_DPV               13    ( DPV DATABASE TABLES NOT SYNCHRONIZED)
#define LIST_DPV                      14    ( DETECTED ADDRESS LIST CREATION)
#define ENV_DPV                       100   ( FAILED TO ALLOCATE INITIAL ENVIRONMENT)
#define INSYNCH_DPV                  101   ( AMS AND DPV TABLES NOT SYNCHRONIZED)
#define LIB_COMPAT_DPV                102   ( FOUND AN INCOMPATIBLE DPV LIBRARY)
#define LIB_LOAD_DPV                  103   ( FAILED TO LOAD DPV LIBRARY)
#define SECURITY_KEY_DPV104           ( VIOLATION - SECURITY KEY GENERATED)
#define CONFIGURED_DPV                105   ( MISSING DATA TABLE(S))
```

Database Tables

Below is a description of the tables used by DPV®:

dphe.hsa	- contains all delivery points (required).
dphe.hsb	- contains business delivery points (DSF2).
dphe.hsc	- contains CMRA delivery points.
dphe.hsd	- contains drop site delivery points (DSF2).
dphe.hsf	- corrects errors in the dphe.hsa table (required).
dphe.hst	- contains throwback delivery points (DSF2).
dphe.hss	- contains seasonal delivery points (DSF2).
dphe.hsv	- contains vacant delivery points.
dphe.hsl	- contains lacs delivery points (DSF2).
dphe.hsk	- contains drop counts for HSC and HSD tables (DSF2).
dphe.hs1	- contains curb delivery type delivery points (DSF2).
dphe.hs2	- contains NDCBU delivery type delivery points (DSF2).
dphe.hs3	- contains centralized delivery type delivery points (DSF2).
dphe.hs4	- contains other/doorslot delivery type delivery points (DSF2).
dphe.hsx	- contains nostat delivery points.
lcd	- resolves ZIP-codes to a common base ZIP-code (required).
lcd.ndx	- used to maximize the performance of the lcd table (required).

Constants used to identify the above tables:

```
#define HSA_DPV          0      (dphe.hsa)
#define HSB_DPV          1      (dphe.hsb)
#define HSC_DPV          2      (dphe.hsc)
#define HSD_DPV          3      (dphe.hsd)
#define HSF_DPV          4      (dphe.hsf)
#define HST_DPV          5      (dphe.hst)
#define HSS_DPV          6      (dphe.hss)
#define HSV_DPV          7      (dphe.hsv)
#define HSL_DPV          8      (dphe.hsl)
#define HSK_DPV          9      (dphe.hsk)
#define HS1_DPV         10      (dphe.hs1)
#define HS2_DPV         11      (dphe.hs2)
#define HS3_DPV         12      (dphe.hs3)
#define HS4_DPV         13      (dphe.hs4)
#define HSX_DPV         14      (dphe.hsx)
#define LCDNDX_DPV16    (lcd.ndx)
#define LCDFILE_DPV     17      (lcd)
```

Database Table Options

Constants used to identify table options for `z4DpvIsOptions()` and `z4DpvSetOptions()`:

```
#define RAMLOAD_DPV      8      (Load table into RAM)
```

Data Types

```
typedef struct tagDeliveryPointValidationParameter
{
    char szAddress    [51 + 1]; /* DELIVERY ADDRESS LINE (OPTIONAL) */
    char szPrimary    [10 + 1]; /* PRIMARY NUMBER */
    char szUnit       [ 4 + 1]; /* UNIT DESIGNATOR */
    char szSecondary  [ 8 + 1]; /* SECONARY NUMBER */
    char szZip5       [ 5 + 1]; /* 5 DIGIT ZIPCODE */
    char szZip4       [ 4 + 1]; /* 4 DIGIT ADDON */
    char szPMB        [ 8 + 1]; /* PRIVATE MAIL BOX NUMBER */
    char szRecType    [ 1 + 1]; /* RECORD TYPE (F,H,P,R,S,G) */
    char szMilitary   [ 1 + 1]; /* Y = Military */
    char szUnique     [ 1 + 1]; /* Y = Unique */
    char szRetCode    [ 1 + 1]; /* AMS RETURN CODE (BINARY FIELD)
                               /* 32  DEFAULT RESPONSE
                               /* 31  EXACT MATCH
                               /* 22  MULTIPLE RESPONSE
                               /* 21  ADDRESS NOT FOUND
                               /* 13  INVALID CITY
                               /* 12  INVALID STATE
                               /* 11  INVALID ZIP CODE
                               /* 10  INVALID ADDRESS
} TDPVParm, *TPDPVParm;
```

Interface Overview

- Interface: `int Z4FUNC z4Dpv(TDPVParam* pDPV)`
 Description: Provides a means of performing a DPV lookup without performing an address lookup.
 Input: TDPVParam structure that identifies the address to perform the DPV query on
 Output: integer with one of the following character values
 'Y' - Confirmed entire address
 'N' - Could not confirm address
 'S' - Confirmed address by dropping secondary information
 'D' - Confirmed a hirise or box type address w/o secondary information
 '' - Address was not submitted for confirmation
 Note: One of these values is always returned.
 The same DPV calls you would normally make after `z4adrinq()` can be made after `z4Dpv()`.
- Interface: `int z4DpvGetCode(int iTableID)`
 Description: Provides the status of a lookup.
 Input: integer identifying database table
 Output: integer with one of the following character values
 'Y' - Confirmed entire address
 'N' - Could not confirm address
 'S' - Confirmed address by dropping secondary information
 'D' - Confirmed a hirise or box type address w/o secondary information
 '' - Address was not submitted for confirmation
 Note: One of these values is always returned.
- Interface: `int z4DpvGetDlvryType(void)`
 Description: Provides the delivery type of a lookup.
 Input: none
 Output: integer with one of the following character values
 '1' - Curb delivery type
 '2' - NDCBU delivery type
 '3' - Centralized delivery type
 '4' - Other/doorslot delivery type
 '' - Address was not submitted for a delivery type lookup
 Note: One of the above values is always returned. This is a DSF2 interface call and is associated with the HS1_DPV, HS2_DPV, HS3_DPV, and HS4_DPV tables.
- Interface: `const char* z4DpvGetDate(void)`
 Description: Identifies the database used by DPV.
 Input: none
 Output: null terminated string formatted as MMDDYYYY
 Note: The date string is null terminated and always returned unless the zip4/dpv library is not loaded correctly or the database is not found. The return value is zero/null when the date string is not returned.
- Interface: `int z4DpvGetDropCnt(void)`
 Description: Identifies the drop count associated with a lookup.
 Input: none
 Output: integer identifying drop count
 Note: The drop count returned is only valid when a lookup is a CMRA or a drop site for any other situation a zero is returned. This is a DSF2 interface call and is associated with the HSC_DPV, HSD_DPV, and HSK_DPV table.

Interface:	const char* z4DpvGetFootnotes(void)
Description:	Provides the state of a lookup.
Input:	none
Output:	null terminated string with a combination of the following character pairs "AA" - zip4 matched "A1" - zip4 did not match "BB" - HSA_DPV confirmed entire address "CC" - HSA_DPV confirmed address by dropping secondary information "F1" - Military match "G1" - General deliver match "N1" - HSA_DPV confirmed a hi-rise address w/o secondary information "M1" - Primary number missing "M3" - Primary number invalid "P1" - Box number missing "P3" - Box number invalid "RR" - HSC_DPV confirmed address with PMB information "R1" - HSC_DPV confirmed address without PMB information "U1" - Unique ZIP code match
Note:	The footnote string is null terminated and always returned unless the zip4/dpv library is not loaded correctly. The return value is zero/null when the footnote string is not returned.
Interface:	const char* z4DpvGetKey (void)
Description:	Returns a security key as a result of a stop processing event.
Input:	none
Output:	null terminated string containing a security key
Note:	The security key is a null terminated alphanumeric string. This key is only returned when stop processing occurs. This security key is used to obtain a second security key that becomes the input of z4DpvSetKey() which verifies the second security key for validity and enables DPV if valid.
Interface:	long z4GetLastErrorCode (void)
Description:	Provides the error codes of the last error to occur in DPV.
Input:	none
Output:	long integer value identifying the last error
Note:	This interface returns errors occurring within DPV. If an error occurs within AMS even as a result of DPV® the error will not show up here. The purpose of this interface is more towards debugging/logging.
Interface:	const char* z4GetLastErrorMsg(void)
Description:	Text description of the error code return from z4GetLastErrorCode();
Input:	none
Output:	null terminated string containing a text description of the last error
Note:	This is a null terminated ASCII string and it may not be formatted enough for user feedback.
Interface:	unsigned long z4DpvGetOptions(int IID)
Description:	Provides a way to determine the option being used by DPV.
Input:	integer identifying database table
Output:	unsigned long integer identifying the currently set options for the specified table
Note:	Returns the options currently being used for a DPV table.
Interface:	const char* z4DpvGetPathname(int IID)
Description:	Provides the path and filename of a database table
Input:	integer identifying database table
Output:	null terminated string indicating path and filename of database table

Interface: const char* z4DpvGetVersion(void)
Description: Identifies DPV® library version.
Input: none
Output: null terminated string formatted as Major.Minor.Micro.CassCycle
Note: The alphanumeric version string (e.g. "3.01.01.G") is null terminated and always returned unless the zip4/dpv library can not be. The return value is zero/null when the version string is not returned.
Major - Major changes made to interface (not backwards compatible)
Minor - Additions made to interface (backwards compatible)
Micro - Internal changes made to library (backwards compatible)
Cass Cycle - Only used for identification purposes

Interface: int z4DpvIsConfirmed(int iTableID)
Description: Identifies the status of a lookup
Input: integer identifying the database table
Output: integer identifying status (TRUE/FALSE)
TRUE - Confirmed address
FALSE - Could not confirmed address
Note: FALSE equals zero and TRUE is not equal to zero

Interface: int z4DpvIsDataValid(int iTableID)
Description: Performs a checksum integrity check on database tables.
Input: integer identifying the database table
Output: integer identifying status (TRUE/FALSE)
TRUE - Good table
FALSE - Bad table
Note: FALSE equals zero and TRUE is not equal to zero

Interface: int z4DpvIsDisabled(void)
Description: Identifies the status of the DPV library
Input: none
Output: integer identifying status (TRUE/FALSE)
TRUE - Disable
FALSE - Enable
Note: FALSE equals zero and TRUE is not equal to zero

Interface: int z4DpvIsDisabledEx(int bDPV)
Description: Identifies the status of the DPV/DSF2 library
Input: integer identifying library type (TRUE/FALSE)
TRUE - DPV®
FALSE - DSF2®
Output: integer identifying status (TRUE/FALSE)
TRUE - Disable
FALSE - Enable
Note: FALSE equals zero and TRUE is not equal to zero

Interface: int z4DpvIsOptions(int iID, unsigned long iOption)
Description: Identifies option(s) currently inuse by DPV
Input: integer identifying database table
integer identifying option(s) to check for
Output: integer identifying state (TRUE/FALSE)
TRUE - Option(s) used
FALSE - Option(s) not used
Note: FALSE equals zero and TRUE is not equal to zero. This call is only effective when used after an AMS api open call.

- Interface: int z4DpvResolveMultiResp(ZIP4_PARM* ext)
 Description: Attempts to resolve an address with a multiple response into a single response
 Input: pointer to a ZIP4_PARM structure with a multiple response result
 Output: integer identifying status (TRUE/FALSE)
 TRUE - Resolved
 FALSE - Unresolved
 Note: When an address lookup results in a multiple response call this interface to break the tie. This call is only effective after a z4adring call.
- Interface: int z4DpvSetKey(const char* szKey)
 Description: Accepts the security key used to enable DPV® after stop processing occurs
 Input: null terminate string containing a security key
 Output: integer identifying the DPV® library status (TRUE/FALSE)
 TRUE - Enabled
 FALSE - Disabled
 Note: The input security key is a key obtain as a result of providing the security key from z4DpvSetKey() to some customer care/tech support rep. Any formatting characters should be removed from the security key before calling this interface. DPV® will be enabled if the security key is valid. This call is only effective when used after an AMS api open call.
- Interface: void z4DpvSetOptions(int iID, unsigned long iOption)
 Description: Provides control over the functionality of database tables
 Input: integer identifying database table
 integer identifying option(s)
 Output: none
 Note: Options are only a suggestion. For example, if the option to load a table into RAM is set the table is only loaded if there is enough memory. This call is only effective when used before an AMS api open call.
- Interface: int z4DpvSetPath(int iTableID, char* pszPathname)
 Description: Updates the path and filename of a database table
 Input: integer identifying database table
 null terminated string indicating path and filename of database table
 Output: integer value indicating the error status
 0 - Success (found and opened table)
 -1 - AMS is already open.
 -2 - Failed creating DPV® environment
 -3 - Internally failed to load DPV® interface
 -4 - User specified an invalid TableID
 -5 - Failed to load DPV® library
 -6 - Failed to open table
 Note: This call must occur before any AMS api open call.

Notes

- The DPV® interface is provided via the AMS API library. When building a DPV® application link only to the AMS API library and not to the DPV® library. An example of a C/C++ compile for the linux platform is listed below:
`cc -o sample.exe sample.o zip4_lnx.dll`
- You can not check individual delivery type tables. The delivery type tables function as a single table. When an ID for a delivery type table is used for a DPV® lookup each delivery type table is checked until one confirms.
- In order to reset errors `z4close()` must be called or if the error is corrected when a DPV® lookup is made the error will be reset.
- Whenever an attempt is made to open the AMS API library, a call to `z4close()` must be made in order to release allocated resources. If the AMS API library fails to open you must still call `z4close()`.
- When using DPV® it is possible for `z4openCfg()` to return values 4 or 5.
 4 = AMS and DPV data tables are not within the same month. (Out of Sync error)
 5 = A security violation has been detected. This occurs when AMS/DPV needs a security key or a DPV® library can not be found. If an error code 5 is returned and `z4DpvGetKey()` does not return a security code then a DPV® library is missing.

DPV® Sample

```

/*****\
*
*                               INCLUDES
*
\*****/
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <string.h>
#include "zip4.h"
#include "z4dpv.h"

/*****\
*
*                               MACROS
*
\*****/
#if !defined(SAFE_STR)
#define SAFE_STR(_String_) ((_String_) ? (_String_) : "")
#endif /* #if !defined(SAFE_STR) */

/*****\
*
*                               DATA TYPES
*
\*****/
typedef struct tagTableInformation
{
    char*    szName;
    char*    szDescr;
    long     iID;
    int      bUsed;
    int      bQuery;
} TTableInfo, *TPTableInfo;

/*****\
*
*                               GLOBALS
*
\*****/
static TTableInfo g_pTableInfo[] =
{
    {"HSA", "Delivery point",          HSA_DPV, 0, 1},
    {"HSB", "Business delivery point", HSB_DPV, 0, 1},
    {"HSC", "CMRA delivery point",     HSC_DPV, 0, 1},
    {"HSD", "Drop site delivery point", HSD_DPV, 0, 1},
    {"HSF", "False positive table",    HSF_DPV, 0, 1},
    {"HST", "Throwback delivery point", HST_DPV, 0, 1},
    {"HSS", "Seasonal delivery point",  HSS_DPV, 0, 1},
    {"HSV", "Vacant delivery point",    HSV_DPV, 0, 1},
    {"HSL", "LACS delivery point",     HSL_DPV, 0, 1},
    {"HSK", "Drop count",              HSK_DPV, 0, 1},
    {"HS1", "Delivery type delivery point", HS1_DPV, 0, 1},
    {"HS2", "Delivery type delivery point", HS2_DPV, 0, 0},
    {"HS3", "Delivery type delivery point", HS3_DPV, 0, 0},
    {"HS4", "Delivery type delivery point", HS4_DPV, 0, 0},
    {"HSX", "Nostat delivery point",   HSX_DPV, 0, 1},
    {"LCD", "LCD table",                LCDFILE_DPV, 0, 0},
    {"LCD NDX", "LCD index",           LCDNDX_DPV, 0, 0},
}

```

```

};

static int g_iTableCount = (sizeof(g_pTableInfo) /
sizeof(g_pTableInfo[0]));

/*****\
*
*          PRIVATE INTERFACE
*
\*****/
static char* GetInput(const char* szMessage, char* szResponse, int iSize);
static void DisplayResults(ZIP4_PARM* pZip4Parm);
static void ReportStatus(int iRetCode);
static void Shutdown(int iRetCode);

/*****\
* Description:   Program entry point
* Input:         none
* Globals:       TTableInfo* - g_pTableInfo (table info array)
*               int         - g_iTableCount (table info array count)
* Return:        int         - error status
\*****/
int main(void)
{
    Z4OPEN_PARM Z4OpenParm      = {{0}, 0};
    ZIP4_PARM   Zip4Parm        = {{0}, {0}};
    char        szPathAMS[1024] = {0};
    char        szPathDPV[1024] = {0};
    char        szPathSecure[1024] = {0};
    char        szResp[100]      = {0};
    int         iRetCode         = 0;
    int         iIndex          = 0;
    /* DISPLAY BANNER */
    printf("\n-----");
    printf("\n|                               USPS AMS/DPV API                               |");
    printf("\n|                               Sample Application                               |");
    printf("\n|-----");
    printf("\n");

    /* SETUP DPV BEFORE Z4OPEN CALL */
    GetInput("\n\nLocation of DPV data files (Include ending slash): ",
            szPathDPV, sizeof(szPathDPV) - 1);

    Z4OpenParm.dpvflag      = 'Y';
    Z4OpenParm.config.dpvpath = szPathDPV;

    /* SETUP AMS BEFORE Z4OPEN CALL */
    GetInput("\n\nLocation of AMS data files (Include ending slash): ",
            szPathAMS, sizeof(szPathAMS) - 1);

    Z4OpenParm.config.address1 = szPathAMS;
    Z4OpenParm.config.addrindex = szPathAMS;
    Z4OpenParm.config.citystate = szPathAMS;
    Z4OpenParm.config.crossref = szPathAMS;
    Z4OpenParm.config.ewspath = szPathAMS;

    GetInput("\n\nLocation of security file (Include ending slash): ",
            szPathSecure, sizeof(szPathSecure) - 1);

    Z4OpenParm.config.system = szPathSecure;

    /* OPTIONAL: CHECK FOR ELOT USERS */
    GetInput("\n\nWould you like to use ELOT (Y for yes, default no)? ",

```

```

        szResp, sizeof(szResp)-1);

if(toupper(szResp[0]) == 'Y')
{
    Z4OpenParm.elotflag          = 'Y';
    Z4OpenParm.config.elot      = szPathAMS;
    Z4OpenParm.config.elotindex = szPathAMS;
}

/* OPEN AMS/DPV API */
iRetCode = z4opencfg(&Z4OpenParm);
if( iRetCode != 0 )
{
    Shutdown(iRetCode);
    return 0;
}

/* SEARCH FOR USED TABLES */
for(iIndex = 0; iIndex < g_iTableCount; iIndex++)
{
    const char* szPathname = z4DpvGetPathname(g_pTableInfo[iIndex].IID);

    /* THE PATHNAME IS SET FOR USED TABLES */
    g_pTableInfo[iIndex].bUsed = (szPathname && szPathname[0]);
}

/* OPTIONAL: CHECK DATABASE INTEGRITY */
GetInput("\nWould you like to check database integrity (Y for yes,
default no)? ", szResp, sizeof(szResp)-1);

if(toupper(szResp[0]) == 'Y')
{
    for(iIndex = 0; iIndex < g_iTableCount; iIndex++)
    {
        /* SKIP UNUSED TABLES */
        if( !g_pTableInfo[iIndex].bUsed )
            continue;

        /* DISPLAY STATUS */
        printf("\n%s table: %s", g_pTableInfo[iIndex].szName,
            z4DpvIsDataValid(g_pTableInfo[iIndex].IID) ? "Y" : "N");
    }
}

/* OPTIONAL: CHECK TABLES BEING LOADED INTO RAM */
if( z4DpvIsOptions(HSC_DPV, RAMLOAD_DPV) )
    printf("\n\ncMRA table loaded into RAM.");
if( z4DpvIsOptions(HSF_DPV, RAMLOAD_DPV) )
    printf("\nFALSE table loaded into RAM.");

/* OPTIONAL: CHECK DATABASE DATE */
printf("\nDPV date (YYYYMMDD): %s", z4DpvGetDate());

/* OPTIONAL: CHECK VERSION */
printf("\nDPV version: %s", z4DpvGetVersion());

/* OPTIONAL: CHECK FOR DPV BEING ENABLED */
if( z4DpvIsDisabled() )
    printf("\nDPV is disabled.");
else
    printf("\nDPV is enabled.");

/* PROGRAM LOOP */
do
{

```

```

printf("\n\n-----");
printf("\n                Address Lookup                ");
printf("\n-----");

/* CLEAR LOOKUP INFO */
memset(&Zip4Parm, 0x00, sizeof(Zip4Parm));

/* GET USER'S ADDRESS INFO */
GetInput("\nFirm Name                : ", Zip4Parm.iadl2,
sizeof(Zip4Parm.iadl2));
GetInput("\nDelivery Address Xtra: ", Zip4Parm.iadl3,
sizeof(Zip4Parm.iadl3));
GetInput("\nDelivery Address        : ", Zip4Parm.iadl1,
sizeof(Zip4Parm.iadl1));
GetInput("\nLast Line                : ", Zip4Parm.ictyi,
sizeof(Zip4Parm.ictyi));
GetInput("\nUrbanization            : ", Zip4Parm.iprurb,
sizeof(Zip4Parm.iprurb));
/* PERFORM LOOKUP */
z4adrinq(&Zip4Parm);

/* CHECK FOR STOP PROCESSING */
if( z4DpvGetLastErrorCode( ) == LIST_DPV )
{
const char* szSecurityCode = z4DpvGetKey();
char        szResp[1024]   = {0};

printf( z4DpvGetLastErrorMsg() );

/* CHECK FOR STOP PROCESSING */
if( szSecurityCode && *szSecurityCode )
{
printf("\nDPV has been disabled.");
printf("\n\nSecurity code: %s", szSecurityCode);
printf("\nTo enable DPV contact customer support with the
security");
printf("\ncode above to recieve the security key you need to
enable DPV");

GetInput("\nEnter security key w/o formatting
characters: ",
szResp, sizeof(szResp)-1);

/* INFORM USER IF SUCCESSFUL */
if( z4DpvSetKey(szResp) )
{
printf("\nDPV has been enabled");
}
}
}

/* CHECK FOR RESOLVING MULTIPLE RESPONSES */
if(Zip4Parm.retcc == Z4_MULTIPLE)
z4DpvResolveMultiResp(&Zip4Parm);

/* DISPLAY RESULTS */
DisplayResults(&Zip4Parm);

/* CHECK FOR ENDING PROGRAM */
GetInput("\nTo lookup another address press Y: ", szResp,
sizeof(szResp)-1);
}while( toupper(szResp[0]) == 'Y' );

```

```
/* CLEANUP (CLOSE AMS API) */  
Shutdown(iRetCode);  
  
return 0;  
}
```

```

/*****\
* Description: Show user the results of a lookup
* Input      : ZIP4_PARM* - pointer to ZIP4_PARM record
* Globals:   TTableInfo* - g_pTableInfo (table info array)
*           int          - g_iTableCount (table info array count)
* Return    : None
\*****/
static void DisplayResults(ZIP4_PARM* pZip4Parm)
{
    int iIndex = 0;
    int iTables = 0;

    /* ERROR CHECK */
    if( !pZip4Parm )
        return;

    /* DISPLAY AMS RESULTS */
    printf("\nReturn Code [%i]      Stack Records [%i]", pZip4Parm->retcc,
pZip4Parm->respn);
    printf("\nRecord Type           : %c", pZip4Parm->stack->rec_type);
    printf("\nFinance                 : %s", pZip4Parm->stack->finance);
    printf("\nFirm Name                   : %s", pZip4Parm->dadl2);
    printf("\nDelivery Address Xtra: %s", pZip4Parm->dadl3);
    printf("\nDelivery Address             : %s", pZip4Parm->dadl1);
    printf("\nUrbanization                 : %s", pZip4Parm->dprurb );
    printf("\nLast Line                    : %s %s %s-%s[%s]",
        pZip4Parm->dctya, pZip4Parm->dstaa, pZip4Parm->zipc,
        pZip4Parm->addon, pZip4Parm->dpbc);
    printf("\nCarrier route                : %s", pZip4Parm->cris);
    printf("\nLOT Number                   : %s%c\n", pZip4Parm->elot_num,
pZip4Parm->elot_code);

    /* DISPLAY DPV RESULTS */
    for(iIndex = 0; iIndex < g_iTableCount; iIndex++)
    {
        int bNewLine = ((iTables % 4) == 0);

        /* SKIP NONQUERY TYPE TABLES */
        if( !(g_pTableInfo[iIndex].bUsed && g_pTableInfo[iIndex].bQuery) )
            continue;

        /* DISPLAY 4 TABLE STATS PER LINE */
        printf(bNewLine ? "\n" : "");

        /* DISPLAY TABLE STATUS */
        if(bNewLine)
        {
            printf("%s=%c",
                g_pTableInfo[iIndex].szName,
                z4DpvGetCode(g_pTableInfo[iIndex].iID));
        }
        else
        {
            printf(" %s=%c",
                g_pTableInfo[iIndex].szName,
                z4DpvGetCode(g_pTableInfo[iIndex].iID));
        }

        /* TRACK USED TABLES */
        iTables++;
    }

    printf("\nDelivery type: %c", z4DpvGetDlvryType());
    printf("\nFootnotes: %s", z4DpvGetFootnotes());
}

```

```

        printf("\nDrop count:      %li\n", z4DpvGetDropCnt(HSK_DPV));
    }

/*****\
* Description: Prompts and retrieves a response from the command line
* Input:      const char* - Message
*             char*      - User's response buffer
*             int        - Size of response buffer
* Return:     char*      - User's response buffer
\*****/
static char* GetInput(const char* szMessage, char* szResponse, int iSize)
{
    int iLen = 0;

    /* ERROR CHECK INPUT */
    if( !szResponse )
        return szResponse;

    /* PROMPT USER */
    if( szMessage )
        printf(szMessage);

    /* GET RESPONSE */
    szResponse[0] = 0;
    fgets(szResponse, iSize, stdin );
    fflush(stdin);

    /* REMOVE THE ENTER KEY CHARACTER */
    iLen = strlen(szResponse);
    if(iLen && ((szResponse[iLen - 1] == '\n') || (szResponse[iLen - 1] ==
'\r')) )
        szResponse[iLen - 1] = 0;

    /* RETURN THE RESPONSE */
    return szResponse;
}

/*****\
* Description: Display diagnostic info about the API
* Input:      int          - z4open()/z4opencfg() return code
* Globals:    TTableInfo* - g_pTableInfo (table info array)
*             int          - g_iTableCount (table info array count)
* Return:     None
\*****/
static void ReportStatus(int iRetCode)
{
    int      iIndex      = 0;
    char     szVersion[20] = {0};
    char     szDate[20]   = {0};
    Z4_ERROR ErrorParm    = {0};
    Z4_ENV   EnvParm       = {{0},{0}};

    /* DISPLAY REPORT BANNER */
    printf("\n\n-----");
    printf("\n                        STATUS REPORT                        ");
    printf("\n-----\n");

    /* GET AMS STATUS INFO */
    z4ver(szVersion);
    z4date(szDate);
    z4getenv(&EnvParm);
    z4geterror(&ErrorParm);
}

```

```

/* DISPLAY AMS STATUS INFO */
printf("\nAMS version:          %s", szVersion);
printf("\nAMS date (YYYYMMDD):   %s", szDate);
printf("\nz4open() return code: %d", iRetCode);

printf("\nError Message:          %s", ErrorParm.strErrorMessage);
printf("\nFile Name:              %s", ErrorParm.strFileName);
printf("\nDiagnostics:              %s", ErrorParm.strDiagnostics);

printf("\nConfiguration File:       %s", EnvParm.strConfigFile);
printf("\nAddress1:                  %s", EnvParm.address1);
printf("\nAddrIndex:                 %s", EnvParm.addrindex);
printf("\nCityState:                 %s", EnvParm.citystate);
printf("\nCrossRef:                  %s", EnvParm.crossref);
printf("\nSystem:                    %s", EnvParm.system);
printf("\neLOT:                      %s", EnvParm.elot);
printf("\neLOTIndex:                 %s", EnvParm.elotindex);
printf("\nEWS Path:                  %s", EnvParm.ewspath);
printf("\neLOT Flag:                 %c\n", EnvParm.elotflag);

/* DISPLAY DPV STATUS INFO */
printf("\nDPV:                        %s", z4DpvIsDisabled() ? "disabled" :
"enabled");
printf("\nDPV version:              %s", SAFE_STR(z4DpvGetVersion()));
printf("\nDPV date (YYYYMMDD):      %s", SAFE_STR(z4DpvGetDate()));
printf("\nError Code:                %li", z4DpvGetLastErrorCode());
printf("\nError Message:            %s", SAFE_STR(z4DpvGetLastErrorMsg()));

/* DISPLAY DPV TABLE INFO */
for(iIndex = 0; iIndex < g_iTableCount; iIndex++)
{
    /* SKIP UNUSED TABLES */
    if( !g_pTableInfo[iIndex].bUsed )
        continue;

    printf("\n%s: %s", g_pTableInfo[iIndex].szName,
z4DpvGetPathname(g_pTableInfo[iIndex].iID));
}

/*****\
* Description: Properly shuts down the AMS/DPV engine
* Input:      int - z4open()/z4opencfg() return code
* Return:     None
*****/
static void Shutdown(int iRetCode)
{
    const char* szSecurityCode = z4DpvGetKey();
    char        szResp[1024]   = {0};
    int         iSize          = sizeof(szResp) - 1;

    /* CHECK FOR STOP PROCESSING */
    if( szSecurityCode && *szSecurityCode )
    {
        printf("\nDPV has been disabled.");

        printf("\n\nSecurity code: %s", szSecurityCode);
        printf("\nTo enable DPV contact customer support with the
security");
        printf("\ncode above to receive the security key you need to
enable DPV");

        GetInput("\nEnter security key w/o formatting characters: ",
szResp, iSize);

```

```
        /* INFORM USER IF SUCCESSFUL */
        if( z4DpvSetKey(szResp) )
            printf("\nDPV has been enabled");
    }

    /* OPTIONAL: DISPLAY STATUS REPORT */
    GetInput("\nWould you like a status report (Y for yes, default no)? ",
        szResp, iSize);

    if(toupper(szResp[0]) == 'Y')
        ReportStatus(iRetCode);

    /* CLOSE AMS/DPV API */
    if(z4close())
        printf("\n\nError closing USPS AMS API\n");
    else
        printf("\n\nUSPS AMS API is closed\n");
}
```