pitney bowes

Location Intelligence

Demographics Library

Version 39.04

Reference Manual for Windows, UNIX

April 2019

Table of Contents

1 – About Demographics Library

Corporations now use demographic information to target their marketing campaigns, forecast sales, determine market penetration, assess market potential, and more. Demographic coding of existing databases is an important part of this process. Centrus® Demographics Library is an easy-to-embed multi-platform Application Programming Interface (API) that allows you to append valuable demographics to your own databases.

In this chapter

About this Manual

This manual contains reference information for all API elements of Centrus Demographics Library. The API is available as a DLL for 32-bit or 64-bit Windows developers and as a C Language library for UNIX developers. The functions within this API allow developers to retrieve demographic information from Pitney Bowes demographic data files. A separately-licensed loader program is also available for converting other data files into the format used by the Demographics Library.

This document contains information on the general use of the Demographics Library API, as well as a complete function reference. A `README.TXT` file may be on the distribution disk and would contain any modifications or corrections to this document. Always assume that the information in `README.TXT` supersedes the printed documentation.

Within this manual, all language references that appear embedded in descriptive text, including function, macro and variable references, will be shown in a monospaced Courier font:

```
x = 1; // Just a sample.
```

Function names embedded in text are shown in **`bold`**.

Both decimal and hexadecimal numbers may be used in this document. Hexadecimal numbers are always preceded by `0x`:

```
0x1F// hex 1F
0x04// hex 04
12// decimal 12
```

While the API is accessible to a variety of programming languages, code examples, function declarations, and comments are given in C. Within Demographics Library, you will find that all of the public functions begin with "`Dl`", such as **`DlFind`**. Where possible, you should avoid using variable names, constant names or macro names beginning with `Dl`. Check the include file in your development environment's directory for a complete list of visible symbols.

If You Need More Help

If you are unable to resolve a problem, a Pitney Bowes Technical Support Representative can help guide you to a solution. When you call Pitney Bowes Technical Support, please have the following information ready:

- A description of the task you were performing
- The resulting reports (specifically, the Execution Log and Parameter Record Listing).

Reporting complete details to Technical Support will help you and the technical support representative quickly resolve the problem.

Pitney Bowes Technical Support representatives work closely with you so that:

- Your questions on using Demographics Library are answered quickly
- Any problems you may encounter while using Demographics Library are resolved.

To reach Technical Support, refer to the contact information on our website: https://www.pitneybowes.com/us/contact-dcs.html.

The Website

You can also find out about Pitney Bowes software products and services on our website: https://www.pitneybowes.com/us/support.html.

2 – Installation

This chapter describes system requirements and how to install Demographics Library for Windows and UNIX operating systems.

In this chapter

Installation Requirements

Demographics Library is supported on the Windows operating system and is distributed as Windows DLLs.

Demographics Library is also supported on multiple UNIX platforms. It is distributed as both shared and static libraries.

Refer to the *Centrus Products Supported Platforms* document located at Pitney Bowes Support and for latest support information.

Windows Installation

This section describes how to install the Windows version of Demographics Library.

Install the Product

1. Download the Demographics Library installation .zip for your platform and extract the files.
2. In the root folder where you extracted the files, locate **setup.exe.** Right-click on **setup.exe** and select "**Run as administrator**".
3. The Demographics Library install wizard launches and will guide you through the installation process.

Install the Data Files

Note: If you plan to use GSD Split, you do not need to complete the steps provided here. The *Centrus® Utilities Manual* provides all the steps necessary to split and install the data files.

For faster processing, copy *all* the data files you will be using to a local directory. These files include Census2k.dld and Census10.dld. Copying the data files to your hard drive allows for quicker processing, but requires considerable disk space.

For descriptions of the data files you may need to complete the desired processing, view the *Installation Notes for the Centrus Data Products Suite* and the *Centrus Data Products Release Announcement* included in the installation package.

Note: To assign Claritas demographics you must have a license. Contact your sales person for details.

Once you have copied all of the data files to your local hard drive, add the data files path to Demographics Library's Search Path.

UNIX Installation

This section describes how to install UNIX versions of Demographics Library.

Install the Product

Download and extract the contents of the tar file for your particular UNIX operating system.

The tar files for each supported UNIX operating system are located in the */demolib/unix* directory.

Install the License File(s)

You only need to install the license file if you are a new customer or if you are renewing your contract with Pitney Bowes. Regular updates do not require new license files.

To install the license file copy or FTP the license file to your installation directory.

Install the Data Files

For faster processing, copy *all* the data files you will be using to a local directory. These files include Census2k.dld and Census10.dld. Copying the data files to your hard drive allows for quicker processing, but requires considerable disk space.

For descriptions of the data files you may need to complete the desired processing, refer to the “content” and “datasets” notes available through viewrel.sh as follows:

```
csh viewre1.sh <cdrom> <unix file name>
```

Where `<cdrom>` indicates where the *Software Installation* CD is mounted and `<unix file name>` indicates which note to display (`content` or `datasets`).

Once you have copied all of the data files to your local hard drive, add the data files path to Demographics Library’s Search Path.

3 – General Use of Demographics Library

The Demographics Library allows users to query a library file for demographic information. The functions in the Demographics Library API work on files called Demographics Library Data files, which are encoded in a proprietary format. These files are referred to as “data files” or .DLD files throughout this manual. The file names have the extension “.DLD”.

In this chapter

Demographics Library files

The Demographics Library ships with one or more .DLD files; you can access multiple .DLD files in a single Demographics Library session. Additional data files can be obtained from Pitney Bowes. If you have existing demographic data files, a builder is available to convert them into the format used by the Demographics Library. These product enhancements are subject to additional licensing fees.

Demographics Library Functions

The Demographics Library functions all use a common internal data structure that is initialized with a call to **`DlInit`**. This function returns a handle, *`DlId`*, which is then passed to other Demographics Library functions. A handle is a token that lets a program access some resource, usually a file or a location within a file. Other Demographics Library functions that return handles are **`DlFileOpen`**, **`DlFieldGet`**, and **`DlFieldGetN`**. Upon completion of the Demographics Library session, a call to **`DlTerm`** frees the memory allocated by **`DlInit`** and closes all files used by Demographics Library. After a call to **`DlTerm`**, any handles that were returned by the Demographics Library are now invalid.

Demographics Library Data Files

Demographics Library Data files contain four kinds of information: *file attributes, field attributes, field values, and field value attributes*.

File attributes are information about the Demographics Library Data file. This includes an overview of the data available, the number of fields contained in the .DLD file, and information about the key field. This information can be accessed with the functions **`DlFileGetAttribute`** and **`DlFileNumFields`**.

Field attributes include information about each field within the .DLD file, such as field name and description, field size, and data types. This information can be accessed with the function **`DlFieldGetAttribute`**.

Field values are the actual data contained in a field, such as the congressional district for a specific ZIP Code. This information can be accessed with the functions **`DlFieldGetNum`** and **`DlFieldGetString`**.

Field value attributes are information about specific data values within a field, such as a description of a piece of data. This information can be accessed with the functions **`DlFieldGetValueAttribute`** and **`DlFieldGetString`**.

Information Available via Demographics Library

Two different Demographics Library Data sets are available for license from Pitney Bowes:

- Standard 2000 and 2010 Census Demographics

A complete listing of data available in these DLD files can be found in Appendix B: Data Files.

Developing a Demographics Library Application

Once Demographics Library has been initialized, there are many options available through the Demographics Library functions. One approach is outlined below:

1. Initialize Demographics Library using **`DlSetLicense`** and **`DlInit`**.
2. Open one or more data files using **`DLFileOpen`**.
3. Get data file attributes using **`DlFileGetAttribute`**.
4. Search data file for key information using **`DlFileSearch`**.
5. Specify a field using **`DlFieldGet`** or **`DlFieldGetN`**.
6. Return data for the field located in step 5 using **`DlFieldGetString`**, **`DlFieldGetNum`**, **`DlFieldGetAttribute`**, **`DlFieldGetValueAttribute`**
7. Repeat steps 4, 5 and 6 as necessary.
8. Close open data file(s) using **`DlFileClose`**.
9. Close Demographics Library using **`DlTerm`**.

Appendix A: Sample Program contains the code listing for a sample program in C illustrating the use of Demographics Library as outlined above. A complete version of this program (DEMOTEST.C) can also be found on your distribution disk in the Demographics Library directory.

DL.H

The C header file DL.H (the file may have different extensions and function definitions for other development systems) contains all symbolic constants used or returned by the Demographics Library functions. This file is extensively commented and it is recommended that you review this file before using any Demographics Library functions.

Using Demographics Library in Windows

To successfully use the Demographics Library in Windows, be sure one of the following symbols is defined:

```
__NT__
_WIN32
```

Please note that while you will not be warned with compile or link errors if these symbols are undefined, your application will not function properly. The most common error is to run out of stack space. Though Microsoft compilers typically define these symbols for you, we strongly recommend that you define them in your project.

Using Demographics Library in UNIX

When using Demographics Library in UNIX, be sure the following symbol is defined:

```
___UNIX
```

On 64-bit machines, the following symbol must also be defined:

```
___64BIT
```

The static library file name is `libdemolib.a` on all UNIX platforms. The shared library file name is `libdemolib.so` for AIX and Solaris, and `libdemolib.sl` for HP-UX. The symbolic link on all UNIX platforms is `libdemolib.so/sl.38.04`.

4 – Function Reference

This chapter describes in detail each of the public functions available through Demographics Library API.

In this chapter

Function syntax

The format for the syntax of each function in this chapter is:

```
ReturnType FunctionName (ArgumentType identifier, ArgumentType
   identifier)
```

where:

- **`ReturnType`** is the type of return value associated with the function.
- **`FunctionName`** is the name of the function.
- **`ArgumentType`** is the type of the identifier that follows.
- *`identifier`* is the descriptive name of the argument.

Data Types Used

ints	short integer
intsu	unsigned short integer
intl	long integer
intlu	unsigned long integer
char *	pointer to type char
intlu *	pointer to unsigned long integer
intl *	pointer to long integer
ints *	pointer to short integer
pstr	pointer to type char
pcstr	constant pointer to type char
pvoid	pointer to type void
DlID	Demographics library handle
DlFileID	Demographics file handle
DlFieldID	Demographics field handle

Note: The asterisk symbol after the type indicates a pointer to a variable. For example, `intl *p` indicates that `p` is a pointer to a signed long integer.

Quick Reference

Initialization, Termination, and Error Functions

`DlInit`

Initializes Demographics Library.

`DlTerm`

Terminates Demographics Library.

`DlErrorGet`

Returns current error information.

`DlSetLicense`

Specifies Pitney Bowes license file.

File Functions

DlFileOpen

Opens .DLD file.

DlFileClose

Closes .DLD file.

DlFileSearch

Searches .DLD file for key.

DlFileNumFields

Returns number of fields in .DLD file.

DlFileGetAttribute

Returns .DLD file information.

Field Functions

DlFieldGet

Returns handle for specified field.

DlFieldGetAttribute

Returns information about specified field.

DlFieldGetN

Returns handle for numerically-specified field.

DlFieldGetNum

Returns numeric field data.

DlFieldGetString

Returns string data for field attribute or field value.

DlFieldGetValueAttribute

Returns information about specified field values.

Full function descriptions

The function descriptions are listed below in alphabetical order.

DlErrorGet

Returns current error information.

Syntax

```
intl DlErrorGet (DlId hLib, char *msg, char *detail)
```

Arguments

hLib
Handle for current instance of Demographics Library, as returned by **DlInit**. *Input*

**msg*
The basic explanation for error. Can be up to 256 characters long. *Output*

**detail*
The particulars of an error, such as file name. Can be up to 256 characters long. *Output*

Note: Both *msg* and *detail* can return up to 256 bytes of data. Buffers for these items should always be 256 bytes or larger.

Return Value

TRUE if an error exists, else FALSE.

Notes

DlErrorGet returns the error message and detail information for the most recent Demographics Library error.

Example

- // This example retrieves the last error information
-
- char ErrorMsg[256], ErrorDetail[256];
-
- DlErrorGet(hLib, ErrorMsg, ErrorDetail);
- printf("An error has occurred: %s, %s", ErrorMsg, ErrorDetail);

DlFieldGet

Returns handle for specified field.

Syntax

```
DlFieldId DlFieldGet (DlFileId hfile, const char *fname)
```

Arguments

hfile
Handle to file object, as returned by **DlFileOpen**. *Input*

**fname*
Field name. *Input*

Return Value

Returns handle to specified field if successful, else NULL.

Notes

DlFieldGet takes a field name and returns the corresponding handle.

See Also

DlFieldGetN

Example

- // This code fragment gets a handle for the field
- // called “DMA” and then gets short and long field
- // descriptions.
-
- DlFieldId field = **DlFieldGet**(hFile,”DMA”);
- char temp[256];
- DlFieldGetString(field, “SDESC”, (char*)temp, sizeof(temp));
- DlFieldGetString(field, “LDESC”, (char*)temp, sizeof(temp));

DlFieldGetAttribute

Returns attribute information about specified field.

Syntax

```
intl DlFieldGetAttribute (DlFieldId hfield, DlAttribute
   which, char *buffer, intl bufSize)
```

Arguments

hfield
Handle to field object, as returned by **DlFieldGet**. Identifies the field for which attribute information is to be returned. *Input*

which
The specific attribute to return. *Input*.

Field Attribute Enum	Description
DlName	Field name
DlFieldname	Field name as it appears in .DLD file

Field Attribute Enum	Description
DlWidth	Field width
DlDecimals	Number of decimal places, if numeric
DlPrice	Not implemented
DlLicenseType	Not implemented
DlLicenseStatus	Not implemented
DlDescription	Short (32 character) description of field
DlHelp	Long (256 character) description of field
DlNumvalues	Number of unique values possible for field
DlType	Field type; “N” (numeric) or “C” (character)

`*buffer`

Location to store the returned data. *Output*

`bufSize`

Maximum size of data to return in *buffer. Input*

Note: If the data returned by *`buffer`* is larger than the space allotted by *`bufSize`*, it will be truncated.

Return Value

Returns attribute information for specified field.

Notes

`DlFieldGetAttribute` is used to retrieve attribute information about a specified field within a .DLD file. This may include such information as field name and description, field size, and data types.

See Also

`DlFieldGet`, **`DlFieldGet`**, **`DlFieldGetValueAttribute`**

Example

See “DlFieldGetN” on page 22 for an additional example.

```
•
•  // This code fragment “walks” through the data file,
•  // fetches each field’s name, and prints it to both
•  // screen and buffer
•
•  intl NumFields = DlFileNumFields(hFile);
•  if (NumFields<=0)
•  return;
```

```
•
•   fields = new DlFieldId[NumFields];
•
•   for (intl counter=0; counter < NumFields; counter++)
•   {
•   fields[counter] = DlFieldGetN(hFile, counter);
•   if (fields[counter] == 0)
•   {
•   delete[ ]fields;
•   return;
•   }
•   if(!DlFieldGetAttribute(fields [counter], DlName,
•   string, 256) && string != "")
•   {
•   /* insert error-handling routines here */
•   }
•   else
•   {
•   sprintf(output, “Name: %s\n”, string);
•   printf(output);
•   }
•
•   }
•
```

DlFieldGetN

Returns handle for numerically-specified field.

Syntax

`DlFieldId DlFieldGetN` (DlFileId *`hfile`*, intl *`fnum`*)

Arguments

`hfile`
Handle to file object, as returned by **`DlFileOpen`**. *Input*

`fnum`
Zero-based field number. *Input*

Return Value

Returns handle to specified field if successful, else NULL.

Notes

Use **DlFileNumFields** to retrieve the number of fields in a data file. With this information, you can specify a loop using **DlFieldGetN** to return the handles for each field in the data file. Alternatively, you can retrieve the handle for a specific field, if you know the field number. This is useful if you don't know the field names in the .DLD file.

See Also

DlFileNumFields, **DlFieldGet**, **DlFieldGetAttribute**

Example

```
// This function determines how many fields there are,
// then loops through all of them, outputting their names.

void ListFields(DlId dl, DlFileId idFile)
{
// Number of fields in the file, and a counter for the loop
intl lNumFields, lCount;

// ID that will be used to retrieve each field name.
DlFieldId idField;
char buffer[256];

lNumFields = DlFileNumFields(idFile);

// Here we get the number of fields.
if (lNumFields < 1)
   {
printf("\nDlFileNumFields returned less than 1.\n");

   // Error if the return was invalid.
ShowError(dl);
}
else
{
printf("\n");
for (lCount = 0; lCount < lNumFields; lCount++)
```

```
// Now loop through all the fields (zero based).
{
// Set idField to the current field number
idField = DlFieldGetN(idFile, lCount);

// Make sure it was set, print an error if not.
if (!idField)
{
printf("DlFieldGetN failed to retrieve field
number %d.\n", lCount);
ShowError(dl);
break;
}

// Get the field name and print it.
DlFieldGetAttribute(idField, DlName, buffer,
sizeof(buffer));
printf("%d: %s\n", lCount, buffer);

// If LINES_PER_PAGE lines have been printed then pause.
// until return is pressed so the user can read what was
// output.

if (!((lCount + 1) % LINES_PER_PAGE))
{
printf("<more>");
gets(buffer);
}
}
}
printf("\n");
}
```

DlFieldGetNum

Returns numeric data from specified field.

Syntax

intl DlFieldGetNum (DlFieldId *hField*)

Arguments

hField

Handle to field, as returned by **DlFieldGet** or **DlFieldGetN**. *Input*

Return Value

Returns the specified field's numeric value, or -1 if the field is not numeric.

Notes

DlFieldGetNum returns the numeric data contained within a specified field in the data file. The specified field is the record located with **DlFileSearch**.

See Also

DlFileSearch, **DlFieldGetAttribute**, **DlFieldGetString**

DlFieldGetString

Returns string data from specified field.

Syntax

intl DlFieldGetString (DlFieldId *hField*, const char **form*, char **buffer*, intl *bufSize*)

Arguments

hField

Handle to field, as returned by **DlFieldGet**. *Input*

**form*

The form in which the data is to be returned. *Input*

form	Return Format
SDESC	Short (32 character) description of the data.
LDESC	Long (256 character) description of the data.
printf string	Returns formatted value data. Format specifications are identical to that used in the "C" function **printf**. Example: "%d total population"

**buffer*

Location to store the returned data. *Output*

bufSize

Maximum size of data to return in *buffer*. *Input*

Note: If the data returned by *buffer* is larger than the space allotted by *bufSize*, it will be truncated.

Return Value

Returns TRUE if successful, else FALSE.

Notes

DlFieldGetString returns the string data contained within a specified field in the data file. You can specify the form in which the data will be returned with the argument *form*. Valid forms are a printf string, "SDESC" for a short (32 character) description explaining the data, or "LDESC" for a long (256 character) description of the data. These descriptions are the same descriptions that you can get using the **DlFieldGetValueAttribute** function. However, while **DlFieldGetValueAttribute** searches the data file for a given field, **DlFieldGetString** will return a description for the specific data value that the record pointer is currently on, as set with **DlFileSearch**.

See Also

DlFileSearch, **DlFieldGetAttribute**, **DlFieldGetNum**

DlFieldGetValueAttribute

Retrieves attribute information for field values.

Syntax

```
intl DlFieldGetValueAttribute ( DlFieldId hField, intl valnum, ValAttribute which, char *buffer, intl bufSize )
```

Arguments

hField
Handle to field object, as returned by **DlFieldGetN**. *Input*

valnum
Zero-based index for values. Use **DlFieldGetAttribute** with the *DlNumValues* enum to return the number of possible values. *Input*

which
Which attribute to return. *Input*

ValAttribute Enums	Description
DlKey	Key value
DlValDescription	Short (32 character) description of the data.
DlValHelp	Long (256 character) description of the data.

**buffer*
Location to store the returned data. *Output*

bufSize
Maximum size of data to return in *buffer*. *Input*

Note: If the data returned by *buffer* is larger than the space allotted by *bufSize*, it will be truncated.

Return Value

Returns TRUE if attribute is a string and the value was copied to buffer, FALSE if unsuccessful, and -1 for an error condition.

Notes

This function is used to retrieve information about specific data values within a field, such as a description of a piece of data.

See Also

DlFieldGetAttribute

Example

```
// This example fetches all field value attributes for a given
// field.

void ShowFieldValueAttributes(DlFileId idField)
{
intl lNumValues, lCount;
char buffer[256];

  if (idField)
  {
    lNumValues = DlFieldGetAttribute(idField, DlNumvalues,
buffer, sizeof(buffer));
    printf("\n");

    if(!lNumValues)
    {
printf("No value descriptions for this field.\n");
    }

for (lCount = 0; lCount < lNumValues; lCount++)
    {
      if (DlFieldGetValueAttribute(idField, lCount,
DlKey, buffer, sizeof(buffer)))
printf("DlKey: %s\n", buffer);
      else
printf("DlKey: <valnum %d unsuccessful>\n",
lCount);
```

```
•
•           if (DlFieldGetValueAttribute(idField, lCount,
•   DlValDescription, buffer, sizeof(buffer)))
•   printf("DlValDescription: %s\n", buffer);
•           else
•   printf("DlValDescription: <unsuccessful>\n");
•
•           if (DlFieldGetValueAttribute(idField, lCount,
•   DlValHelp, buffer, sizeof(buffer)))
•   printf("DlValHelp: %s\n", buffer);
•           else
•   printf("DlValHelp: <unsuccessful>\n");
•
•           if (!(((lCount + 1) * 4) % PAGE_SIZE))
•           {
•   printf("<more>");
•   gets(buffer);
•           }
•         }
•       }
•   else
•       {
•   printf("\nNo field specified.\n");
•       }
•   printf("\n");
•   }
•
•
```

DlFileClose

Closes data file.

Syntax

```
intl DlFileClose (DlFileId hFile)
```

Arguments

hFile
Handle to the data file object, as returned by **DlFileOpen**. *Input*

Return Value

TRUE if successful, else FALSE if failure.

Notes

`DlFileClose` closes the open data file and deallocates its associated memory, invalidating the file handle.

See Also

`DlFileOpen`

DlFileGetAttribute

returns attribute information for .DLD file.

Syntax

```
intl DlFileGetAttribute (DlFileId hFile, DlFileAttribute
   which, char *buffer, intl bufSize)
```

Arguments

`hFile`

Handle to data file object, as returned by **`DlFileOpen`**. *Input*

`which`

The specific attribute to return. *Input*

DlFileAttribute Enums	Description
DlKeyType	Numeric type of key (see below)
DlNumFields	Number of fields in file
DlNumValues	Maximum number of values per field
DlFileDescription	Long (256 character) description of file
DlKeyFieldName	Name of key field
DlKeyLimit	If 1, limits keys to alphanumeric characters

`DlKeyType` is returned as one of the following values:

DlKeyType	Description
DL_TYPE_ZIP	ZIP Code
DL_TYPE_ZIP4	ZIP+4 Code
DL_TYPE_CTY	County
DL_TYPE_CT	Census tract

DlKeyType	Description
DL_TYPE_BG	Block group
DL_TYPE_BLK	Block

**buffer*
Location to store the returned data. *Output*

bufSize
Maximum size of data to return in *buffer*. *Input*

Note: If the data returned by *buffer* is larger than the space allotted by *bufSize*, it will be truncated.

Return Value

Returns TRUE if attribute is a string and the value was copied to buffer, or a number if attribute is numeric.

Notes

DlFileGetAttribute is used to retrieve attribute information about a specified .DLD file. This may include such information as field name and description, field size, and data types.

DlFileNumFields

Returns number of fields in data file.

Syntax

```
intl DlFileNumFields (DlFileId hFile)
```

Arguments

hFile
Handle to file object, as returned by **DlFileOpen**. *Input*

Return Value

Returns number of fields in data file if successful, else -1.

Notes

DlFileNumFields counts the number of fields in an open data file. This information can then be used with **DlGetFieldN** to return handles for any or all fields in the data file.

See Also

DlFieldGetN, **DlFileGetAttribute**

DlFileOpen

Opens data file.

Syntax

DlFileId DlFileOpen (DlId *hLib*, const char **DLD*)

Arguments

hLib

Handle for current instance of Demographics Library, as returned by **DlInit**. *Input*

**DLD*

String containing the full name and path of the file to open. This string must be 256 characters or less. Strings longer than 256 characters will be truncated. *Input*

Return Value

Returns handle to the data file if successful, NULL if unable to open file.

Notes

DlFileOpen opens a .DLD file for use and initializes memory for reading the data. A single instance of the Demographics Library can open multiple data files by repeating the **DlFileOpen** function.

See Also

DlFileClose

DlFileSearch

Searches data file for specified key.

Syntax

intl DlFileSearch (DlFileId *hFile*, const char **key*)

Arguments

hFile

Handle to data file object, as returned by **DlFileOpen**. *Input*

**key*

Pointer to the string containing the specified key. *Input*

Return Value

TRUE if key is found, else FALSE.

Notes

DlFileSearch searches the data file for a specified key.

See Also

DlFileOpen

DlInit

Initializes Demographics Library.

Syntax

`DlId DlInit` `(const char *paths)`

Arguments

`*paths`
A list of paths to search for the files CTYST.DIR and MSAC.DIR. *Input*

Return Value

A valid `DlId` is returned if the system initializes correctly. Zero is returned if **`DlInit`** failed.

Notes

`DlInit` initializes the API and allocates required memory. **`DlInit`** should be called once at the start of a session. When the session is done, **`DlTerm`** should be called to perform cleanup.

Files CTYST.DIR and MSAC.DIR are required unless the installation is licensed for "all states."

See Also

`DlTerm, DlSetLicense`

Example

```
// Initializes then terminates Demographics Library

DlId Init;

Init = DlInit(path);
if (Init==0)
{
printf("Unable to initialize demographics library.\n");
exit(1);
}

// If program reached this point, initialization was
// successful.
//
// Insert program code...
//
```

- // ...then terminate Demographics Library.
-
- DlTerm(Init);

DlSetLicense

Identifies license file and password.

Syntax

intl DlSetLicense(DlId *hLib*, const char **filename*, intl *password*)

Arguments

hLib
Handle for current instance of Demographics Library, as returned by **DlInit**. *Input*

**filename*
The fully qualified file name, including drive and path, of the license file containing the licensed information. *Input*

password
The long integer used as a unique key. *Input*

Return Value

TRUE if license file was successfully opened, else FALSE.

Notes

This identifies the password and license file name used when invoking Demographics Library. The license file may have any name.

Files CTYST.DIR and MSAC.DIR are required unless the installation is licensed for “all states.”

See Also

DlInit

DlTerm

Terminates Demographics Library.

Syntax

intl DlTerm (DlId *hLib*)

Arguments

hLib
Handle for current instance of Demographics Library, as returned by **DlInit**. *Input*

Return Value

`DL_SUCCESS`

Notes

Frees memory allocated by **`DlInit`** and closes all files used by the Demographics Library. **`DlTerm`** is called at the conclusion of a session to close files and release other resources. Once this is done, all handles returned by Demographics Library are invalid. You must then reinitialize the Demographics library with **`DlInit`** before you can call Demographics Library functions.

See Also

`DlInit`, **`DlFileClose`**

A - Sample Program

This Appendix contains the code listing for a sample program illustrating the use of the main Demographics Library functions. A complete version of this program (DEMOTEST.C) can be found in your Demographics Library installation directory.

In this appendix

Example

```
//////////////////////////////////////////////////////////////
// Module:DemoTest.c
//
// Purpose:Simple test program to illustrate use of the
// demographic library interface.
//
// These coded instructions, statements, and computer programs
// contain unpublished trade secrets and proprietary
// information of Pitney Bowes and are protected by
// Federal Copyright law and by Trade Secret law.  They may not
// be disclosed to third parties or used or copied or
// duplicated in any form, in whole or in part, without the
// prior written consent of Pitney Bowes.
//////////////////////////////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include “dl.h”

static long password;
static char initPaths[256], licenseFile[256];

#define FILE_HANDLES 3 // How many demographics files can
// be open at once
#define LINES_PER_PAGE 24// How many lines to scroll at a
// time

/*
** Function: ShowError
**
** Purpose:  Prints the message and detail of the last error
** that occured, or “No Error.” if none have occured
*/
void ShowError(DlId dl)
```

```
{
char msg[256], detail[256];
if ( DlErrorGet(dl, msg, detail) )// DlErrorGet returns
// zero if there is NOT
// an error
{

printf("\nDlErrorGet:\n");
printf("Msg: %s\n", msg);
printf("Detail: %s\n\n", detail);
  }
  else
  {
printf("\nNo Error.\n\n");
  }
}

/*
** Function: PickHandle
**
** Purpose:  DemoTest has several file handles, so that more
** than one .DLD file can be open at once. The exact number is
** specified by FILE_HANDLES. This function lets the user
** choose which of these to use.
*/

void PickHandle(int *pCurrentFile)
{
char buffer[10];
int  iTemp;

printf("\nChoose a file number between 1 and %d.\n",
FILE_HANDLES);

// Handles are zero based, so add one
  printf("Current file handle is %d.\n", *pCurrentFile + 1);
```

```
•
•       gets(buffer);
•       if ( *buffer )// If user simply hits return, return to
•    // menu
•       {
•    iTemp = atoi(buffer) - 1;// Subtract 1 because
•    // handles are zero-based
•
•    // Make sure that the number is a valid choice
•          if (iTemp >= 0 && iTemp < FILE_HANDLES)
•          {
•    *pCurrentFile = iTemp;
•          }
•          else
•    {
•    printf("Handle must be between 1 and %d.\n",
•    FILE_HANDLES);
•    }
•       }
•       printf("\n");
•    }
•
•    /*
•    ** Function: Close
•    **
•    ** Purpose:  Close an open .DLD file.
•    */
•
•    void Close(DlId dl, DlFileId *pId)
•    {
•    if (!(*pId))// File Id's are kept equal to zero
•    // if not open in this program
•
•    {
•    printf("\nNo file open.\n");
•       }
```

```
    else
    {
if (!DlFileClose(*pId))// DlFileClose returns zero if
// it failed
        {
printf("\nError closing file.\n");
ShowError(dl);
        }
        else
        {
*pId = 0;// If a file id is zero we know it
// is NOT open
printf("\nClosed.\n");
}
}
printf("\n");
}

/*
** Function: Open
**
** Purpose:  Open a .DLD file. If a file is currently open, it
** is closed.
*/

void Open(DlId dl, DlFileId *pId)
{
char buffer[256];

if (*pId)// First make sure no file is open
    {
if (!DlFileClose(*pId))// If it is, then close it
{
printf("\nError closing file.\n");
ShowError(dl);
return;
```

```
    }
    }
    printf("\nEnter the path and filename of the demographics
    file.\n");
      gets(buffer);
      if (*buffer)// If the user simply hits return, return to
    // menu
      {
    *pId = DlFileOpen(dl, buffer);
    if (!(*pId))// DlFileOpen returns zero if it failed
    {
    printf("Unable to open specified file.\n");
    ShowError(dl);
    }
    }
    printf("\n");
    }


    /*
    ** Function: ShowFileAttributes
    **
    ** Purpose:  Shows attributes about the currently open file.
    ** These are returned from DlFileGetAttribute, and are such
    ** things as the number of fields, the key type, the key field
    ** name, etc.
    */
    void ShowFileAttributes(DlFileId id)
    {
    char buffer[256];
    intl lRetVal, lNumFields;

    if (!id)// First make sure a .DLD file is
    // open
    {
    printf("\nNo file open.\n");
```

```
    }
    else
    {
  printf("\nFILE ATTRIBUTES:\n");

  // Retrieve the key field name - the key field is the one you
  // search by
  DlFileGetAttribute(id, DlKeyFieldName, buffer,
  sizeof(buffer));
  printf("Name of key field:        %s\n", buffer);

  // Retrieve the type of value the key field is
  lRetVal = DlFileGetAttribute(id, DlKeyType, buffer,
  sizeof(buffer));
      printf("Numeric type of key:      ");
  // To check if a value (bit) is set in lRetVal, the binary &
  // operator is used
  if (lRetVal & DL_TYPE_ZIP)
  printf("ZIP Code\n");
  else if (lRetVal & DL_TYPE_ZIP4)
  printf("ZIP+4 Code\n");
  else if (lRetVal & DL_TYPE_CTY)
  printf("County\n");
  else if (lRetVal & DL_TYPE_CT)
  printf("Census tract\n");
  else if (lRetVal & DL_TYPE_BG)
  printf("Block group\n");
  else if (lRetVal & DL_TYPE_BLK)
  printf("Block\n");

  // Retrieve the number of fields in the file.
  lRetVal = DlFileGetAttribute(id, DlNumFields, buffer,
  sizeof(buffer));
  lNumFields = DlFileNumFields(id);
  if (lRetVal == lNumFields)// Quick test to verify both
  // functions return the same
```

```
// value
printf("Number of fields:          %d\n", lRetVal);
else
{
printf("\nERROR\n");
printf("DlFileGetAttribute returned %d fields in
file.\n", lRetVal);
        printf("DlFileNumFields returned %d fields in
file.\n\n", lNumFields);
}

// Retrieve the maximum number of values per field
lRetVal = DlFileGetAttribute(id, DlNumValues, buffer,
sizeof(buffer));
printf("Max Values per field:      %d\n", lRetVal);

// Retrieve the key limit - if it is 1, then the key is limited
// to alphanumeric characters
DlFileGetAttribute(id, DlKeyLimit, buffer,
sizeof(buffer));
printf("Limited to alphanumeric?:  %s\n", ( (lRetVal ==
1)? "TRUE": "FALSE") );

// Retrieve a description of the file.
DlFileGetAttribute(id, DlFileDescription, buffer,
sizeof(buffer));
printf("Description of file:       %s\n", buffer);
}
printf("\n");
}

/*
** Function: Search
**
** Purpose:  To search the current .DLD file for a key,
** specified by the user. Remember that the key field type can
```

```
** be displayed from Attributes in the file menu
** (ShowFileAttributes).
*/
void Search(DlFileId id)
{
char buffer[256];

printf("\nEnter the key you wish to search for.\n");
gets(buffer);

if ( DlFileSearch(id, buffer) )// DlFileSearch returns a
// non- zero value for
// success
{
printf("Found\n");
  }
else
{
printf("Not Found\n");
}
printf("\n");
}

/*
** Function: ListFields
**
** Purpose:  This function determines how many fields there
** are, and then loops through all of them, outputting their
** names.
*/
void ListFields(DlId dl, DlFileId idFile)
{
intl lNumFields, lCount;// Number of fields in the
// file, and a counter for the
// loop ID that will be used to
// retrieve
```

```
DlFieldId idField;// each field name
char buffer[256];
//Here we get the number of fields
lNumFields = DlFileNumFields(idFile);
if (lNumFields < 1)
{
//Error if the return was invalid
printf("\nDlFileNumFields returned less than 1.\n");

ShowError(dl);
}
else
{
printf("\n");

// Now loop through all the fields (zero based)
for (lCount = 0; lCount < lNumFields; lCount++)
{
// Set idField to the current field number
idField = DlFieldGetN(idFile, lCount);

// Make sure it was set, print an error if not
if (!idField)
{
printf("DlFieldGetN failed to retrieve field
number %d.\n", lCount);
ShowError(dl);
break;
}
// Get the field name and print it
DlFieldGetAttribute(idField, DlName, buffer,
sizeof(buffer));
printf("%d: %s\n", lCount, buffer);

// If LINES_PER_PAGE lines have been printed
if (!((lCount + 1) % LINES_PER_PAGE))
```

```
{// Pause until return is
// pressed so the
printf("<more>");// user can read what was
// output.
gets(buffer);
}
}
}
printf("\n");
}

/*
** Function: ChooseField
**
** Purpose:  To retrieve data, a successful file search on a
** key value
** must have been done, and a field must be specified.
** This function specifies the field by name or number.
*/
void ChooseField(DlFileId idFile, DlFieldId *idField)
{
intl lFieldNum;// Used when a user chooses by number
intl lNumFields;// Used to show a user the valid range
// when choosing by number
char choice[10];// Choice - to choose by name or number
char buffer[256];// Used when a user chooses by name

printf("\nChoose by number? (Y or N for name): ");
gets(choice);

if (*choice)// If the user simply hit return,
// return to menu
{
if (*choice == 'Y' || *choice== 'y')// Check for upper &
// lower case...
{
```

```
    lNumFields = DlFileNumFields(idFile);

    printf("Enter the field number (between 0 and %d): ",
    lNumFields - 1);
    gets(buffer);
    lFieldNum = atol(buffer);//Convert to a number

    // DlFieldGetN retrieves a field by number
    *idField = DlFieldGetN(idFile, lFieldNum);
    }
    else if (*choice == 'N' || *choice== 'n')// Check for
    // upper & lower case.
        {
    printf("Enter the field name: ");
    gets(buffer);
    *idField = DlFieldGet(idFile, buffer);// DlFieldGet
    // retrieves a field name
    }
    else
    printf("Invalid entry.\n");

    if (!*idField)// Check to see if the field was
    // found
    {
    printf("Field not found.\n");
    }
    }
    printf("\n");
    }

    /*
    ** Function: ShowFieldAttributes
    **
    ** Purpose:  Shows attribute information on the currently
    ** selected field.
    ** Some of these include field name, width, type
```

```
** (character or numeric),etc.
*/
void ShowFieldAttributes(DlFieldId idField)
{
char buffer[256];
intl lRetVal;

// Make sure there is a field selected
if (!idField)
{
printf("\nNo field specified.\n");
}
else
{
printf("\nFIELD ATTRIBUTES:\n");
// Retrieve the name
DlFieldGetAttribute(idField, DlName, buffer,
sizeof(buffer));
printf("Field Name:              %s\n", buffer);

// Retrieve the name as seen in the .DLD file
DlFieldGetAttribute(idField, DlFieldname, buffer,
sizeof(buffer));
printf("Name in .DLD file:       %s\n", buffer);

// Retrieve the field width. Notice that numeric values are
// retrieved by return value, rather than through buffer
lRetVal = DlFieldGetAttribute(idField, DlWidth, buffer,
sizeof(buffer));
printf("Field Width:             %d\n", lRetVal);

// Retrieve the field type.
lRetVal = DlFieldGetAttribute(idField, DlType, buffer,
sizeof(buffer));
if (lRetVal == 'C')
{
```

```
printf("Field Type:          Character\n");
}
else if (lRetVal == 'N')
{
printf("Field Type:          Numeric\n");

// If the field is numeric, then retrieve the number of decimal
// places
// All numbers come back without decimals, and this value can
// used to display them correctly
lRetVal = DlFieldGetAttribute(idField, DlDecimals,
buffer, sizeof(buffer));
printf("Decimal Places:      %d\n", lRetVal);
}

// Retrieve the number of possible unique values for this field
// This value is used with DlFieldGetValueAttribute
lRetVal = DlFieldGetAttribute(idField, DlNumvalues,
buffer, sizeof(buffer));
printf("Number of possible values: %d\n", lRetVal);

// Retrieve a short description of the field
DlFieldGetAttribute(idField, DlDescription, buffer,
sizeof(buffer));
printf("Short Description of file: %s\n", buffer);

// Retrieve a long description of the field
DlFieldGetAttribute(idField, DlHelp, buffer,
sizeof(buffer));
printf("Long Description of file:  %s\n", buffer);
}
printf("\n");
}

/*
** Function: ShowFieldValueAttributes
```

```
**
** Purpose:  Show attributes for unique values in current
** field. The number of unique values is retrieved by
** DlFieldGetAttribue, and the attributes for these values are
** retrieved by DlFieldGetValueAttribute.
*/
void ShowFieldValueAttributes(DlFieldId idField)
{
intl lNumValues, lCount;// Number of unique values in
// field, and a counter for the
// loop
char buffer[256];

if (!idField)// Make sure a field is selected
{
printf("\nNo field specified.\n");
}
  else
{
printf("\n");

// First get the number of unique values
lNumValues = DlFieldGetAttribute(idField, DlNumvalues,
buffer, sizeof(buffer));

// If it returns zero, there are no unique values, and thus no
// descriptions
if(!lNumValues)
{
printf("No value descriptions for this field.\n");
}

// DlFieldGetValueAttribute accepts a number which serves as an
// index to retrieve a description. Remember that this index is
// zero based.
for (lCount = 0; lCount < lNumValues; lCount++)
```

```
{

// Retrieve the unique value, notify user if unsuccessful
if ( DlFieldGetValueAttribute(idField, lCount, DlKey,
buffer, sizeof(buffer)) )
printf("DlKey: %s\n", buffer);
else
printf("DlKey: <valnum %d unsuccessful>\n",
lCount);

// Retrieve the short description for this unique value
if ( DlFieldGetValueAttribute(idField, lCount,
DlValDescription, buffer, sizeof(buffer)) )
printf("  DlValDescription: %s\n", buffer);
else
printf("  DlValDescription: <unsuccessful>\n");

// Retrieve the long description for this unique value
if ( DlFieldGetValueAttribute(idField, lCount,
DlValHelp, buffer, sizeof(buffer)) )
printf("  DlValHelp: %s\n", buffer);
else
printf("  DlValHelp: <unsuccessful>\n");

// We'll assume that each value is 4 lines, just to make
// sure nothing scrolls off the top of the screen before the
// user can read it.
   // Waits for return to be pressed to continue.
if (!( ((lCount + 1) * 4 ) % LINES_PER_PAGE))
{
printf("<more>");
gets(buffer);
}
}
}
printf("\n");
```

```
}

/*
** Function: PrintData
**
** Purpose:  To output the data specified after a successful
** search has been completed, and a file has been successfully
** specified. This checks if a field is specified, but whether
** a search has been completed is tracked internally by
** DemoLib.
*/

void PrintData(DlId dl, DlFieldId idField)
{
char buffer[256];
intl lRetVal;
// Make sure a field is selected
if (!idField)
{
printf("\nNo field specified.\n");
}
else
{
printf("\n");

// First determine what type of value will be retrieved, by
// using DlFieldGetAttribute
lRetVal = DlFieldGetAttribute(idField, DlType, buffer,
sizeof(buffer));

// If it's a character, then the value will be retrieved from
// DlFieldGetString
if (lRetVal == 'C')
{
printf("Value is character; results will be shown
from DlFieldGetString.\n");
```

```
•
•   // DlFieldGetString returns zero if it failed.
•   if (!(DlFieldGetString(idField, “DlFieldGetString
•   printf string: %s\n”, buffer, sizeof(buffer))) )
•   {
•   printf(“No data returned from
•   DlFieldGetString.\n”);
•   ShowError(dl);
•   }
•   printf(buffer);
•   }
•
•   // If it's numeric, the value will be retrieved from
•   // DlFieldGetString and DlFieldGetNum.
•   else if (lRetVal == 'N')
•   {
•   printf(“Value is numeric; results will be shown from
•   DlFieldGetString and DlFieldGetNum.\n”);
•   printf(“Decimal places are not shown.\n”);
•
•   lRetVal = DlFieldGetNum(idField);
•   printf(“DlFieldGetNum value: %d\n”, lRetVal);
•
•   // DlFieldGetString returns zero if it failed
•   if (!(DlFieldGetString(idField, “DlFieldGetString
•   printf string: %d\n”, buffer, sizeof(buffer))) )
•   {
•   printf(“No data returned from
•   DlFieldGetString.\n”);
•   ShowError(dl);
•   }
•   printf(buffer);
•   }
•
•   // The short description (SDESC) and long description (LDESC)
•   // are always retrieved
```

```
if (!(DlFieldGetString(idField, “SDESC”, buffer,
sizeof(buffer))) )
{
printf(“No short description returned from
DlFieldGetString.\n”);
}
else
{
printf(“DlFieldGetString SDESC: %s\n”, buffer);
}

if (!(DlFieldGetString(idField, “LDESC”, buffer,
sizeof(buffer))) )
{
printf(“No long description returned from
DlFieldGetString.\n”);
}
else
{
printf(“DlFieldGetString LDESC: %s\n”, buffer);
}
}
printf(“\n”);
}

/*
** Function: ReadIniFile
**
** Purpose:  Reads in a file name “demotest.ini” in the working
** directory,which contains the license file name, password,
** and the path for DlInit, so the user doesn't have to enter
** it every time they run the program.
*/
void ReadIniFile()
{
char pass[10];
```

```
char * p;
FILE * setup = fopen( “demotest.ini”, “rt” );
if ( setup )
{
fgets( licenseFile, sizeof(licenseFile), setup );
p = strchr( licenseFile, '\n' );
if ( p ) *p = 0;

fgets( pass, sizeof(pass), setup );
p = strchr( pass, '\n' );
if ( p ) *p = 0;
password = atol(pass);

// If this entry is not in the file, initPaths will simply
// be empty (““) which is what we want
fgets( initPaths, sizeof(initPaths), setup );
p = strchr( initPaths, '\n' );
if ( p ) *p = 0;

fclose( setup );
}
else
{
*licenseFile = 0;// If there is no ini file, and no
// command line parameters,just
password = 0;// make everything empty
*initPaths = 0;// strings or zero.
}
}

/*
** Function: FieldMenu
**
** Purpose:  This is the second menu that is presented after a
** file has
** successfully been opened.
```

```
*/
void FieldMenu(DlId dl, DlFileId idFile)
{
int iDone;
DlFieldId idField;

if (idFile)
{
idField = 0;
for (iDone = 0 ; !iDone ; )
{
char choice[10];
printf( “Search File, List Fields, Choose Field,
Attributes, Value Attributes,\n”
“Print Data, Error, File Menu (S/L/C/A/V/P/E/F):
“ );
gets( choice );
switch (*choice)
{
case 'S':
case 's':
Search(idFile);
break;
case 'L':
case 'l':
ListFields(dl, idFile);
break;
case 'C':
case 'c':
// This accepts a pointer to a DlFieldId, so it can change
// its value
ChooseField(idFile, &idField);
break;
case 'A':
case 'a':
ShowFieldAttributes(idField);
```

- break;
- case 'V':
- case 'v':
- ShowFieldValueAttributes(idField);
- break;
- case 'P':
- case 'p':
- PrintData(dl, idField);
- break;
- case 'E':
- case 'e':
- ShowError(dl);
- break;
- case 'F':
- case 'f':
- iDone = 1;
- break;
- default:
- // If the user enters an invalid choice, request a valid one
-
- printf(“Please enter S, L, C, A, V, P, E, or
- F.\n”);
- }
- }
- }
- else
- {
- printf(“\nNo file open.\n\n”);
- }
- }
-
- /*
- ** Function: main
- **
- ** Purpose: This gets all initilization data, either from the
- ** user or from the ini file, and initialized the demographics

```
** library.
*/
void main( int argc, char ** argv )
{
DlId dl;// Handle to demographics library
DlFileId dlFiles[FILE_HANDLES];// Array of demographics
// file handles
int iCurrentFile;// Current index into handle array
int iCount, iDone;// Counter for loop, and value to check
//  if loop is done
char pass[16]; // Password from user

printf( “Starting...\n” );

// If user didn't enter at least a license file and password,
// read the ini file
if ( argc < 3 )
{
ReadIniFile();
if ( password == 0 )// There was no ini file, so
// get info from user
{
printf(“Enter license file path and name: “);
gets( licenseFile );
if ( *licenseFile )
{
printf(“Enter the 8 digit password: “);
password = atol(pass);
printf(“Enter the search path for DlInit: “);
gets(initPaths);
}
}
}

// Otherwise the user did enter license file and password
else
```

```
{
strcpy( licenseFile, argv[1] );
password= atol(argv[2]);

// If there are more arguments, then take it as the search
// path for DlInit
if (argc > 3)
strcpy( initPaths, argv[3]);
else
*initPaths = 0;
}

// Initialize demographics library
dl = DlInit(initPaths);

// And check to make sure it was successful
if ( !dl )
{
printf( “Unable to initialize demographics library.\n” );
}
else
{
#if defined( _WINDOWS )
_wsetexit( _WINEXITNOPERSIST );
#endif

// Set the license file and password. This is not needed to
// use Census90.dld
if ( !(DlSetLicense(dl, licenseFile, password)) )
printf(“License file could not be opened.\nOnly\
CENSUS90.DLD will be accessible.\n”);

// Set all file handles to zero to indicate no file is open
for (iCount = 0; iCount < FILE_HANDLES; iCount++)
{
dlFiles[iCount] = 0;
```

```
}
// The current file handle (in the array) is the first one
// (zero)
iCurrentFile = 0;

// Loop until user chooses to quit, at which point iDone
// will be set to zero
for (iDone = 0 ; !iDone ; )
{
char choice[10];
printf( “Pick Handle, Open, Close, Attributes, Field
Menu, Error, Quit(P/O/C/A/F/E/Q): “ );
gets( choice );
switch (*choice)
{
case 'P':
case 'p':

// Accepts a pointer so it can alter the value
PickHandle(&iCurrentFile);
break;
case 'O':
case 'o':

// Sending a pointer to the file handle specified
// by iCurrentFile
Open(dl, &(dlFiles[iCurrentFile]));
break;
case 'C':
case 'c':

// Sending a pointer to the file handle specified by
// iCurrentFile
Close(dl, &(dlFiles[iCurrentFile]));
break;
case 'A':
```

```
case 'a':
ShowFileAttributes(dlFiles[iCurrentFile]);
break;
case 'F':
case 'f':
FieldMenu(dl, dlFiles[iCurrentFile]);
break;

case 'E':
case 'e':
ShowError(dl);
break;
case 'Q':
case 'q':
iDone = 1;
break;
default:

// If the user enters an invalid choice, request
// a valid one
printf( “Please enter P, O, C, A, F, E, or
Q.\n” );
}
}
}

// Close all file handles
printf(“\nClosing files...\n”);
for (iCount = 0; iCount < FILE_HANDLES; iCount++)
{
printf(“Handle %d:”, iCount + 1);
Close(dl, &(dlFiles[iCount]));
}

// Free all memory and resources allocated by DlInit
DlTerm( dl );
```

- }

B – Data Files

Two Demographics Library Datasets are available from Pitney Bowes:

- Standard 2000 (Census2k.dld)
- Standard 2010 (Census10.dld)

In this appendix

The variables available in each file are listed below. All variables are numeric in type; variable name, length, and the number of decimal places are listed along with descriptions. If you have any questions regarding licensing of this data, contact a Pitney Bowes sales representative at 800 782-7988.

Standard 2000 Census Demographics

Census2k.dld

Output String Field Name	Data Type N—numeric C—char string	Width—Number of characters including null terminator	Number of decimals if numeric	Description
AGHHI00	N	16	0	2000 Aggregate Household ($000's)
AVGHHSZ00	N	11	2	2000 Average Household Size.
FAM00	N	10	0	2000 Families
FEMPOP00	N	10	0	2000 Total Female Population
GQPOP00	N	10	0	2000 Population in Group Quarters
HHCHLD1800	N	4	0	2000 Total Households with Children under 18
HHOVER6000	N	4	0	2000 Total Households with Adults over 60
HH15TO2400	N	10	0	2000 Householder Age 15-24
HH25TO3400	N	10	0	2000 Householder Age 25-34
HH35TO4400	N	10	0	2000 Householder Age 35-44
HH45TO5400	N	10	0	2000 Householder Age 45-54
HH55TO6400	N	10	0	2000 Householder Age 55-64
HH65TO7400	N	10	0	2000 Householder Age 65-74
HH75TO8400	N	10	0	2000 Householder Age 75-84
HH85OVR00	N	10	0	2000 Householder Age 85+
HH00	N	10	0	2000 Households
HU00	N	10	0	2000 Total Housing Units
OWNOCCHU00	N	4	0	2000 Owner Occupied Housing Units
MEDAGE00	N	11	1	2000 Median Age.
MEDAGEF00	N	11	1	2000 Median Age Female.
MEDAGEM00	N	11	1	2000 Median Age Male
MEDFAMI00	N	10	0	2000 Median Family Income
MEDHHI00	N	10	0	2000 Median Household Income
MEDHOMEV00	N	10	0	2000 Median Housing Value
POP00	N	10	0	2000 Total Population

Output String Field Name	Data Type N—numeric C—char string	Width—Number of characters including null terminator	Number of decimals if numeric	Description
XASNPOP00	N	11	1	2000 % Asian/Pacific Islander Population
XBLKPOP00	N	11	1	2000 % Black Population
XINDPOP00	N	11	1	2000 % American Indian/Eskimo Population
XWHTPOP00	N	11	1	2000 % White Population
BG90	N	12	0	Census 1990 Block Group
CSUBFIPS00	N	5	0	Census County Subdivision FIPS 55 Code
PLACEFPS00	N	5	0	Place FIPS 55 Code

Standard 2010 Census Demographics

Census10.dld

Output String Field Name	Data Type N—numeric C—char string	Width—Number of characters including null terminator	Number of decimals if numeric	Description
AGGHHI10	N	16	0	2010 Aggregate Household ($000's)
AVGHHSZ10	N	11	2	2010 Average Household Size
FAM10	N	10	0	2010 Families
FEMPOP10	N	10	0	2010 Total Female Population
GQPOP10	N	10	0	2010 Population in Group Quarters
HHCHLD1810	N	10	0	2010 Total Households with Children under 18
HHOVER6010	N	10	0	2010 Total Households with Adults over 60
HH15TO2410	N	10	0	2010 Householder Age 15-24
HH25TO3410	N	10	0	2010 Householder Age 25-34
HH35TO4410	N	10	0	2010 Householder Age 35-44
HH45TO5410	N	10	0	2010 Householder Age 45-54
HH55TO6410	N	10	0	2010 Householder Age 55-64

Output String Field Name	Data Type N—numeric C—char string	Width—Number of characters including null terminator	Number of decimals if numeric	Description
HH65TO7410	N	10	0	2010 Householder Age 65-74
HH75TO8410	N	10	0	2010 Householder Age 75-84
HH85OVR10	N	10	0	2010 Householder Age 85+
HH10	N	10	0	2010 Households
HU10	N	10	0	2010 Total Housing Units
OWNOCCHU10	N	10	0	2010 Owner Occupied Housing Units
MEDAGE10	N	11	1	2010 Median Age.
MEDAGEF10	N	11	1	2010 Median Age Female
MEDAGEM10	N	11	1	2010 Median Age Male
MEDFAMI10	N	10	0	2010 Median Family Income
MEDHHI10	N	10	0	2010 Median Household Income
MEDHOMEV10	N	10	0	2010 Median Housing Value
POP10	N	10	0	2010 Total Population
XASNPOP10	N	11	2	2010 % Asian/Pacific Islander Population
XBLKPOP10	N	11	2	2010 % Black Population.
XINDPOP10	N	11	2	2010 % American Indian/Eskimo Population.
XWHTPOP10	N	11	2	2010 % White Population.
BG	N	12	0	Census 2010 Block Group index
BG10	N	12	0	Census 2010 Block Group output field
CSUBFIPS10	C	70	0	comma-delimited set of Census County Subdivision FIPS 55 Code values
PLACEFPS10	C	50	0	comma-delimited set of Place FIPS 55 Code values
NECTA	C	11	0	comma-delimited set of 2010 New England City and Town Area code values
NECTA_NAME	C	120	0	comma-delimited set of 2010 New England City and Town Area names
DIVISION	C	11	0	comma-delimited set of 2010 Division values
DIVISION_N	C	100	0	comma-delimited set of 2010 Division names

Index

A

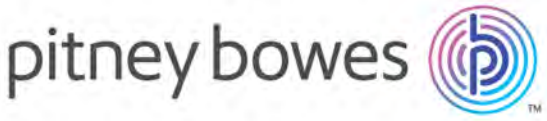
C

D

E

F

W



350 Jordan Road
Troy, NY 12180
USA

www.pitneybowes.com/us

Support: +1 (800) 367-6950
Main: +1 (518) 285-6000
Fax: +1 (518) 285-7060