

# PAFlink Realtime Programmers Reference Manual

Australia  
Version 8.1a



# PAFlink Realtime Australia Version 8.1b

## **Product Support**

**Phone** – +61 2 9032 3100

**Email** – [support.paflink@acxiom.com](mailto:support.paflink@acxiom.com)

**Fax** – +61 2 9016 4828

## **Product**

PAFlink

## **Document Title**

Programmers Reference Manual

## **Document Version**

Version 8.1b published November 2007.

## **Trademarks**

PAFlink is pending registration as a trademark of Acxiom Australia Pty Ltd (ABN 95 087 293 525).

## **Trademarks**

PAFlink is pending registration as a trademark of Acxiom Australia Pty Ltd (ABN 95 087 293 525).

AMAS® and the AMAS logo are registered trademarks of Australia Post.



The PAFlink Version 8.1 software has been approved by Australia Post in accordance with the Address Matching Approval System (AMAS) for the 2008 AMAS cycle.

## **Copyright Notice**

Copyright© 1999-2007, Acxiom Australia Pty Ltd (ABN 95 087 293 525).

All rights reserved. Not to be reproduced without permission.

# Table of Contents

<b>Introduction .....</b>	<b>4</b>
Introduction to the PAFlink API .....	4
<b>Installation Instructions .....</b>	<b>5</b>
Introduction .....	5
Installing PAFlink on Windows.....	5
Installing PAFlink on Unix.....	6
Installation on Other Systems .....	6
Testing Your Installation.....	7
<b>Templates .....</b>	<b>9</b>
Templates Explained.....	9
Using Case of Template Column Codes .....	9
Specifying Fixed-width Fields.....	10
<b>Programming with the API .....</b>	<b>11</b>
Introduction .....	11
The C interface to the API .....	11
PAFlink Active X/COM Realtime API.....	12
PAFlink .NET Realtime API.....	12
PAFlink Web Service Realtime Interface .....	13
An Overview of the Functions to Call.....	14
API Functions in Detail .....	15
PAFlink API Programming for ICF .....	23
An Overview of the ICF Functions to Call .....	23
ICF API Functions in Detail .....	24
PAFlink API Programming for Data Append .....	25
An Overview of the Data Append Functions to Call .....	25
Data Append API Functions in Detail .....	26
Thread Safe Functionality and Multi-threaded Executable .....	27
Thread-safety for API users.....	27
Multithreaded Executable .....	28
Template Examples.....	29
Search Nearest Strategies .....	31
Files for PAFlink Batch Engine .....	35
<b>TCP/IP Server Engine .....</b>	<b>37</b>
Introduction .....	37
Dpserver .....	38
Dpt - Perl Interface to Dpserver .....	39
Dperem – Windows Interface to Dpserver .....	41
<b>Linking Applications to PAFlink.....</b>	<b>43</b>
Introduction .....	43
<b>Glossary.....</b>	<b>46</b>

# Introduction

---

## Introduction to the PAFlink API

Acxiom has developed several packages for looking processing Australian addresses using the Australia Post PAF. At the heart of these packages is the PAFlink address matching engine.

The PAFlink address matching engine can be handled as a function library, with a standard API. The interface is described in terms of the C language, but other languages have successfully called these functions, including Visual Basic, Cobol, and various database languages.

The PAFlink engine has functions to perform an AMAS-certified address match, or alternatively a partial address search, for use in an interactive “rapid address lookup” application.

The library has a batch 'wrapper', handling a simple file format. This is run as a stand-alone program. This program is called PAFlink Premier Batch and its operation is described in the “**Running PAFlink Batch**” chapter of the **PAFlink User Guide**. For the purposes of this manual, its only function is for testing the integrity of the installation of the software.

This manual is directed mainly towards programmers who wish to incorporate the functions of the PAFlink address matching engine into their own programs. The manual also contains some deeper insights into the ways PAFlink can be used, and real world address problems.

### Sales Contact Details

Acxiom Australia Pty Limited  
Level 9 151 Clarence St  
Sydney NSW 2000

Phone: +61 2 9032 3100  
Fax: +61 2 9016 4828

[support.paflink@acxiom.com](mailto:support.paflink@acxiom.com)  
<http://www.acxiom.com.au>

# Installation Instructions

---

## Introduction

PAFlink is available on various platforms: Unix, Windows and VMS. The installation instructions vary slightly between these systems.

## Installing PAFlink on Windows

PAFlink for Windows is installed using a standard Windows installer.

1. Insert the PAFlink CD into your CD ROM drive.
2. Select Start, Run and then Browse to locate the CD ROM drive in the list of drives.
3. Double click on the Setup icon under directory **\Windows** on the CD.
4. You have an option to select “typical” install (for PAFlink Batch and Rapid), “compact” (PAFlink Batch files), or “custom” if you know which components you require.
5. The installation program will load the required files onto your computer in either the default directory, c:\PAFlink, or the directory specified by you during the installation.
6. To confirm that the installation was successful, double click the PAFlink batch icon found in the PAFlink installation directory. This will open a DOS box, and display some information. You can type in addresses and examine the program's responses. <Ctrl> C will terminate the test.

If you require the reverse lookup functionality you should re-install the software and select the **Reverse Look Up Files** option in the **Custom** install. Reverse lookup functionality is not automatically installed due to the size of the data files.

## Installing PAFlink on Unix

PAFlink for Unix is installed by copying files from the CD in the \Unix directory and extracting the platform-dependant executable files found in the tar file.

1. Insert the PAFlink CD into your CD ROM drive.
2. Create a directory for the PAFlink files, /usr/local/paflink is the default. Give the directory the permissions that you wish. You may wish to give read permission to all, but restrict write permission to root only.
3. Using ftp, or equivalent, copy the contents of the \Unix directory on the CD into the directory you have created. If using FTP remember to transfer files in binary mode, it is not necessary to copy the tar files provided for other platforms.
4. If you require the reverse lookup functionality you will need to copy the contents of the \Reverse directory from the CD into the PAFlink installation directory.

***NOTE:** Remember to copy with long files names. The CD has been created with Windows long filename support, rather than RockRidge. If you can't see long filenames on the CD, you may have to copy, or ftp, the files first from a Windows system.)*

5. In the installation directory unpack the platform dependant tar file for your system. For example for Sun this will be pafsun.tar.Z, for Compaq Alpha Tru64 this will be paftru.tar.Z, for Linux paflinux.tar.gz etc.

The appropriate file for your installation is unpacked using the command:  
`zcat pafXXXX.tar.Z | tar xvf -`

After unpacking, several executables and several shared libraries (.so) files should be present. Ensure that the executables **paflink.sh** and **paflinkexe** have execute permissions.

## Installation on Other Systems

Installation on other systems is covered in separate information sheets.

## Testing Your Installation

### Starting PAFlink

After installation you can confirm that all files have been copied across and unpacked correctly by running PAFlink Batch in interactive mode as follows:

For **Windows**, the procedure is to first open a DOS box. You can do this by running 'Command' or by double clicking the MS-DOS icon. Then run PAFlink by typing:

```
C:\PAFlink>paflink
```

If you installed PAFlink in another directory, say D:\PAFlink, you would run it as follows:

```
D:\PAFlink>paflink -p d:\paflink
```

The case of the **-p** flag is important. It must be lower case. The program should respond with the following output:

```
C:\>
C:\>c:\paflink\paflink
c:\paflink\paflink
Copyright (C) 2005, Acxiom Australia Pty Ltd
Tables take a moment to load...
>
```

The arrow indicates that PAFlink is waiting for input.

On **Unix**, you would be running a terminal (command line) session. Run PAFlink by executing the **paflink.sh** script as follows:

```
usr@server:/usr/local/paflink >
usr@server:/usr/local/paflink >./paflink.sh
./paflinkexe
Copyright (C) 2005, Acxiom Australia Pty Ltd
Tables take a moment to load...
>
```

This Unix executable is called **paflinkexe** and is started by the **paflink.sh** shell script. The reason for this is that PAFlink uses shared libraries, which do not reside in the standard Unix location. Because of this the **paflink.sh** script is used to tell the execution environment where the libraries are located.



If instead of displaying the “>” symbol, the program responds as follows:

```
C:\>
C:\>C:\paflink\paflink
c:\PAflink\paflink
Copyright (C) 2005, Acxiom Australia Pty Ltd
Tables take a moment to load...
F 1

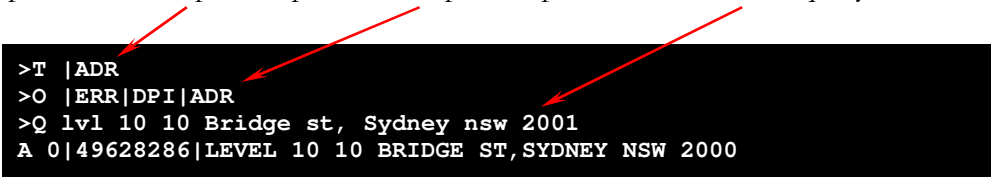
C:\>
```

**F 1** is a fatal error, which indicates that the PAflink data files could not be found. This may be because the **-p** flag and directory were not specified when the application was started, or because the data was copied incorrectly, or corrupted, during the installation.

Try either restarting the application again using the **-p** flag or reinstalling from the CD. If the problem persists, contact PAflink support. More information on Fatal Error Codes can be found in Chapter 5 of the PAflink User Guide.

## Processing Data

Normally input data for PAflink Batch would come from a file, but when using it interactively we can type input directly into the application. In the following example we have provided: an input template, an output template and an address query:



```
>T |ADR
>O |ERR|DPI|ADR
>Q lvl 10 10 Bridge st, Sydney nsw 2001
A 0|49628286|LEVEL 10 10 BRIDGE ST,SYDNEY NSW 2000
```

The input template, the **T** line, tells PAflink what your input address will look like. In this case we are going to input a single addresses line, and no other information.

The output template, the **O** line, tells PAflink the required content and format of the output. There are a great variety of input and output column types we can choose from. See Chapter 5 of the PAflink User Guide, for details on all the available column codes.

The **Q** line is our address query and the last line, prefixed with an **A**, is the answer or response from PAflink. In the example above an error code of 0, a DPID and a standardised PAF address were all returned from PAflink Batch. This indicates that the software is functioning correctly.



# Templates

---

## Templates Explained

The PAFlink engine relies on the user describing the input data so that it can be processed meaningfully. Templates are used to describe how the input will be presented e.g. fixed width or delimited. They are also used to describe the type of input, full address, free format address, individual address elements and non-address elements.

See the API description of the function call **DPE\_SetInTemplate**, and Chapter 5 of the **PAFlink User Guide**, for descriptions of available column codes.

Column codes are all three-character abbreviations for the field name. These three-character codes are separated by the delimiter character which is required for the template layout, even if the data is fixed-width.

An output template is used to describe to PAFlink what information we require to be returned. This is similar to the input template.

See the function call **DPE\_SetOutTemplate**, and Chapter 5 of the **PAFlink User Guide**, for the available column codes. Some codes will only make sense for input, and some for output, while the majority will be available for use in both input and output columns.

## Using Case of Template Column Codes

The text case (UPPER or lower) of the column codes is used to indicate the required case conversion of the output fields.

### Input Template

The standard for input column codes is UPPER CASE. In other situations it may be important for the case to be preserved, such as the customer information content of the barcode.

When the address parsing function is required, however, all address fields will be converted to upper case, even pre-parsed fields such as “STT” state code.

### Output Template

The output column codes can be in upper, lower case or mixed case. UPPER CASE codes force output to be all upper case, lower case, or mixed case produces Proper Case output.

## Specifying Fixed-width Fields

Column codes can be used in conjunction with two numbers to indicate fixed length fields. Fixed length fields are available on input and output records. The first number is the starting column and the second gives the column width. It is easiest to illustrate with an example:

**Input:** Delimiter “,” template “ign, ad1, ad2, ad3, ad4”  
**Output:** Delimiter “|” template “ERR(1,2)|DPI(3,8)|ADR(11,40)”  
**Query Address:** EXAMPLE RECORD 1,L9 151 CLARENCE  
 ST,SYDNEY,NSW,2000  
**Produces:** 45713519LEVEL 9 135-151 CLARENCE ST,SYDNEY NSW  
 2000 TN1 TN2

```
>T ,IGN,AD1,AD2,AD3,AD4
>O |ERR(1,2)|DPI(3,8)|ADR(11,45)|CHG(56,10)
>Q EXAMPLE RECORD 1,L9 151 CLARENCE ST,SYDNEY,NSW,2000
A 0 45713519LEVEL 9 135-151 CLARENCE ST,SYDNEY NSW 2000
TN1 TN2
>
```

# Programming with the API

---

## Introduction

Acxiom has developed several packages for looking up addresses on the PAF. At the heart of all these packages is the PAFlink engine.

The PAFlink engine can be handled as a function library, with a standard API. The interface is described in terms of the C language, but other languages have successfully called these functions, including Basic, Cobol, and various database languages.

This section of the manual is directed mainly towards programmers who wish to incorporate the functions of the PAFlink engine into their own programs. This section also contains some deeper insights into the ways PAFlink can be used.

## The C interface to the API

The API, or Application Programming Interface, is a set of functions that can be called from third party applications. This provides record at a time interaction with the matching engine.

These functions are also defined in the header file **dpe.h**, which is distributed with PAFlink. Also distributed is **paftest.c**, or **ccdtest.c** for PAFlink with CCD Append, a sample program that demonstrates calling these functions.

The file **dpecodes.h** contains some manifest constants, for example the fatal error codes and allocation error codes.

## PAFlink Active X/COM Realtime API

This is a set of functions that provide address validation and DPID allocation in a real-time Windows environment. The real time ActiveX API provides developers with a means to easily integrate PAFlink address matching technology into their Windows and Web based CRM applications. This solution provides developer friendly access to the PAFlink and Address Tune API's. The developer has access to a range of functions including:

- An AMAS certified matching function
- A 'search near' partial lookup for developing a rapid addressing front-end
- Barcode function for producing a barcode from an entered DPID
- Sort plan function for producing the Barcoded Pre-sort code
- Ability to specify output using customer or predefined filters
- Ability to output results as delimited text (for file generation)
- Integrated error handling

## PAFlink .NET Realtime API

This is a set of functions that provide address validation and DPID allocation in a real-time Windows environment via a .NET interface. The real time .NET API provides developers with a means to easily integrate PAFlink address matching technology into their Windows and Web based CRM applications. This solution provides developer friendly access to the PAFlink and Address Tune API's. The developer has access to a range of functions including:

- An AMAS certified matching function
- A 'search near' partial lookup for developing a rapid addressing front-end
- Barcode function for producing a barcode from an entered DPID
- Sort plan function for producing the Barcoded Pre-sort code
- Ability to specify output using customer or predefined filters
- Ability to output results as delimited text (for file generation)
- Integrated error handling

For more information, look at the PAFlink .NET API User Guide.

## PAFlink Web Service Realtime Interface

This is a set of functions that provide address validation and DPID allocation via the SOAP protocol (web service) interface. The real time web service interface provides developers with a means to easily integrate PAFlink address matching technology into their CRM applications. A web service interface allows distributed applications to interact across different machines and with different programming languages.

The PAFlink methods are made available via an Acxiom hosted ADP Web Service.

The PAFlink methods available from the web service are defined by the WSDL (Web Services Description Language) file.

This solution provides developer friendly access to the PAFlink and Address Tune API's.

Table 1 PAFlink Interface Architecture

Client Application		
COM	.NET	Web Service
PAFlink Realtime API (C Interface)		
PAFlink Files		

The PAFlink Architecture allows the following application interactions:

- 1) Any application can interact using the C API Interface
- 2) Windows applications can interact using the COM interface.
- 3) Windows applications can interact using the .NET interface.
- 4) Any application can interface via the Web Service interface.

For more information, look at the PAFlink Web Service Information Guide.

## An Overview of the Functions to Call

The most common functions used to drive the PAFlink engine would be as follows:

Function	Description
<b>DPE_OpenFiles</b>	Opens index and data files, load the parser interpreter and its associated data tables.
<b>DPE_NewQis</b>	Returns a handle that is used in subsequent queries.
<b>DPE_SetInTemplate</b> <b>DPE_SetOutTemplate</b>	These two functions set the input and output templates, in the same manner as the 'T' and 'O' command codes in the batch engine. Note that the separator character is a separate parameter to these functions, not the first character as in the 'T' and 'O' lines.
<b>DPE_SearchAMAS</b>	After the templates are defined, the search function can be called repeatedly. This will return one record for a complete input address. It is the AMAS compatible function.
<b>DPE_SearchNearest</b>	If a search for near addresses is desired, the function can be called repeatedly. This will return multiple records. Normally used instead of DPE_SearchAMAS.
<b>DPE_DisposeQis</b> <b>DPE_CloseFiles</b>	Finally, to clean up the library, use these two functions.

Functions that return an integer value can return one of the serious error conditions, as outlined above.

## API Functions in Detail

```
int
DPE_OpenFiles(char *path);
```

This function can take a path to where the data, index and script files have been installed. Set path to NULL to accept the default location.

The return code is DPERR\_OK or an error code (most likely DPERR\_NOFILE).

```
int DPE_EXPORT
DPE_DataFilesInMemory(int flag);
```

Some systems do not efficiently cache reads of PAFlink data files. To increase processing speed on systems with sufficient real memory, use this function to load some or all of the data files into RAM at the start of execution. This will not have a large effect on Unix, Linux and Windows NT systems.

Flag can take the following values–

- 0 - no data tables loaded (default)
- 1 - small data tables loaded (25mb)
- 2 - all data tables loaded (98mb)

The function should be called after **DPE\_OpenFiles**, and before **DPE\_NewQis**.

```
typedef void * QIS;
```

QIS is a handle. It's a pointer to a private data structure used by the engine.

```
QIS
DPE_NewQis(void);
```

Before calling any functions that require a QIS, this function should be called.

This function can return NULL, if memory cannot be allocated.

```
int
DPE_SetAnswerFormat(QIS qisv, char *pre, char *post);
```

For the **SearchNearest** function, multiple responses can be returned. The default separator between these multiple responses is the C string 'NUL' value. This can be changed by this function. The pre parameter value is prepended to each record and the post parameter value appended to each record.

This also sets these values for the single record for the **DPE\_SearchAMAS** function.



```
int
DPE_SetInTemplate(QIS qis, char sep, char *in_template);
```

This function sets the characteristics of queries. The field delimiter of input queries, and of the input template, is set by the **sep** character.

```
DPE_SetInTemplate(qis, '|', "Adr");
is the same as the input line
T |Adr
```

Note the difference between this and the 'T' line of the batch engine. The 'T' line has the separator as the first character. This is not true for this function.

This function can return a fault code. An example might be DPERR\_INTERFACE, indicating that one of the three letter field names was perhaps invalid.

```
int
DPE_SetOutTemplate(QIS qis, char sep, char *out_template);
```

This function sets the characteristics of responses. The field delimiter of responses, and of the output template, is set by the **sep** character.

```
DPE_SetOutTemplate(qis, ':', "ERR:DPI:ADR");
is the same as the input line
O :ERR:DPI:ADR
```

Note the difference between this and the 'O' line of the batch engine. The 'O' line has the separator as the first character. This is not true for this function.

This function can return a fault code. An example might be DPERR\_INTERFACE, indicating that one of the three letter field names was perhaps invalid.

```
void
DPE_DisposeQis(QIS qis);
```

When the QIS handle is no longer required, this function is called.

```
int
DPE_SearchAMAS(QIS qis,
char *query,
char *buffer, unsigned int *ret_cnt, unsigned max_len);
```

Single AMAS compliant address query for DPID allocation.

The query input is passed in the **query** parameter. The result (for this function at most one result will come back) will be returned in **buffer**. The parameter **ret\_cnt** contains the address of an integer where the number of results (which can only be one or zero) will be placed.

The parameter **max\_len** lets the function know the maximum number of characters that buffer can take (including the NUL).

This function can return a fault code. An example might be DPERR\_TRUNCATE. Truncate is not a serious fault, just that the buffer you gave was not large enough, and obviously some of the returned data will be lost.

After calling this function, inspect the output field 'ERR' to determine why an address may not have been allocated.

An example of calling this function, given the two **DPE\_SetInTemplate** and **DPE\_SetOutTemplate** calls above might be:

```
cnt = 0;

res = DPE_SearchAMAS(qis, "12 Carlotta St, Artarmon NSW
                        2064", buffer, &cnt, sizeof(buffer));

/* buffer should now contain
0|93059437|12 CARLOTTA ST,ARTARMON NSW 2064
*/
```

```
int
DPE_SearchNearest(QIS qis,
char *query,
char *buffer, unsigned int *ret_cnt, unsigned max_len);
```

Multiple result search nearest function.

The query is passed in the **query** parameter. The results will be returned in **buffer**. The parameter **ret\_cnt** contains the address of an integer where the number of results (which can be zero) will be placed.

The parameter **max\_len** lets the function know the maximum number of characters that buffer can take (including the NUL).

This function can return a fault code. An example might be DPERR\_TRUNCATE. Truncate is not a serious fault, just that the buffer you gave was no large enough.

More information on Search Nearest strategies is given in the Search Nearest Strategies section of this chapter.

```
int
DPE_GetLastAddressable(QIS qisv, int *ecode);
```

This function sets the integer variable pointed to by **ecode** to the value 0 if the previously searched for address was not an address that appears to contain the minimum required number of elements for the Australia Post bulk mail programs.

If the previous address does appear to contain the required elements, the variable pointed to by **ecode** is set to non zero.

The return code for the function is DPERR\_OK if OK, otherwise one of the codes given in the file dpecodes.h

```
void
DPE_CloseFiles();
```

Closes PAF files.

```
int
DPE_OpenReverseFiles(char *path);
```

This function opens the extra files needed for a reverse DPID lookup. This requires extra database file to have been installed. These files are groupg.dp\* and pointd.dp\*.

Set path to NULL to accept the default location.

This function should be called before **DPE\_NewQis** if you wish to do reverse queries.

```
int  
DPE_SearchReverse(QIS qis,  
char *query,  
char *buffer, unsigned int *ret_cnt, unsigned max_len);
```

This function goes backwards from a DPID to return an address. The function takes the usual template line, but really only examines the DPI field.

The query is passed in the **query** parameter. The results will be returned in **buffer**. The parameter **ret\_cnt** contains the address of an integer where the number of results (which can be zero or one) will be placed.

The parameter **max\_len** lets the function know the maximum number of characters that buffer can take (including the NUL).

This function can return a fault code.

Localities can have synonym names. Only the main name given by Australia Post (the "Canonical Locality", see column code "CLC") is returned.

```
int  
DPE_Barcode(QIS qis,  
char *query,  
char *buffer, unsigned int *ret_cnt, unsigned max_len);
```

This function doesn't look up addresses. Instead it allocates a barcode using the DPI, CUS, CET and FCC fields. The resulting barcode is returned in the BAR field.

The query is passed in the **query** parameter. The results will be returned in **buffer**. The parameter **ret\_cnt** contains the address of an integer where the number of results (which can be zero or one) will be placed.

The parameter **max\_len** lets the function know the maximum number of characters that buffer can take (including the NUL).

This function can return a fault code.

```
int
DPE_Sortplan(QIS qis,
char *query,
char *buffer, unsigned int *ret_cnt, unsigned max_len);
```

This function doesn't look up addresses. Instead it allocates a sortplan number using the postcode (PCD) field. The resulting sortplan number is returned in the BSP field.

The query is passed in the **query** parameter. The results will be returned in **buffer**. The parameter **ret\_cnt** contains the address of an integer where the number of results (which can be zero or one) will be placed.

The parameter **max\_len** lets the function know the maximum number of characters that buffer can take (including the NUL).

This function can return a fault code.

```
char *
DPE_ErrStr(int err);
```

Return a printable string given a catastrophic error code. Use this function up to and including return from **OpenFiles**. Within a thread, and for a particular QIS handle, use **DPE\_LastErrStr**.

```
DPE_CHAR_STAR
DPE_LastErrStr(QIS qisv, int err)
```

Return a printable string given a fatal error code. This function is useful within an active thread, as it uses the QIS value.

```
void
DPE_SetLastError(QIS qisv, int err)
```

Used to set the fatal error code, if required to signal to another function that such a condition has occurred.

```
int
DPE_GetLastCode(QIS qisv, int *ecode);
```

This function allows you to examine the last allocation error code (the error code that tells you whether a DPID was allocated or not).

```
int
DPE_GetCountValues(QIS qisv, int count_type, unsigned int *retvalue)
```

This function allows you to examine the counts for the various types of records processed. After calling this function, the count is set to zero.

The count types are:

<b>QI_CNT_TOTAL</b>	This is the total number of records processed.
<b>QI_CNT_OK</b>	This is the number of records returned with a DPID, and that are not primary point.
<b>QI_CNT_NODPID</b>	This is the number of records for which no DPID is allocated.
<b>QI_CNT_PP</b>	This is the number of records for which an exact match wasn't found but that a primary point (same street number) was found.

```
int
DPE_SetPersonality(int argc, const char **argv);
```

This function allows you to set the 'meta' file name, used in parsing.

To use this function, you should call it before the **DPE\_OpenFiles** call.

The 'meta' file is used to configure the parser process inside PAFlink. The 'meta' file is different for statically linked versions of PAFlink, as opposed to shared library or dynamically linked versions. It is different again for windows.

Normally these differences are accounted for by renaming the meta file. This may be inconvenient when using PAFlink as a library.

The meta file must reside inside the standard PAFlink directory, where all the other tables go. You cannot pre-pend a path to the name.

The function is called by setting **argc** to the number of arguments, in this case it will always be one.

**Argv** is a pointer to an array of char pointers. The last one is always a NULL pointer.

The first entry should be "-m" with the **meta** file name next. No spaces should be used.

Here is some sample code, setting the meta file name to 'paflinkstatic.meta'.

```
void
set_meta_name_and_open_files(void)
{
char *ptrs[2];
    ptrs[0] = "-mpaflinkstatic.meta";
    ptrs[1] = (char*)NULL;

    DPE_SetPersonality(1, ptrs);
    DPE_OpenFiles((char*)NULL);
}
```

```
unsigned int
DPE_FormatHeader(char *Label, char *retbuffer, unsigned int
max_size);
```

This can be used to get the version, date and time stamps for the library and for data files. Returns the count of number of null terminated strings passed to **retbuffer**.

**Label** can be one of :

"Library"	engine version string
"Locality"	Suburb data
"Localitym"	also suburb data
"Localityi"	also suburb data
"Localityn"	also suburb data
"Group"	Street Names
"Groupm"	also street names (metaphones)
"Groupg"	street names used for reverse DPID lookups
"Point"	addresses within the street
"Pointd"	addresses used for reverse DPID lookups
"lcf"	addresses in Incremental Change File



## PAFlink API Programming for ICF

The great benefit of integrating the ICF functions into an application system is for databases that are subject to continual updating. The updates can be made record by record, without the need to bring the database down.

### An Overview of the ICF Functions to Call

#### **DPE\_OpenIcfFiles**

This function opens the extra files needed for an ICF (Incremental Change) lookup. This requires extra database file to have been installed. These files are **icf.dpi** and **icf.dpd**.

#### **DPE\_SearchIcf**

This function performs a lookup of the ICF to check if the DPID is contained there. If it is, the ERR 13 DPRC\_DPID is returned. Requires previous call to **OpenIcfFiles**.

#### **DPE\_Verify**

Takes a DPID flagged by the function **DPE\_SearchIcf** and the corresponding address. Will provide an updated address and DPID (which may be 0). Requires previous call to **OpenIcfFiles**.

## ICF API Functions in Detail

```
int
DPE_OpenIcfFiles(char *inpath, char *early_date, char *late_date)
```

This function opens the extra files needed for an ICF (Incremental Change) lookup. This requires extra database file to have been installed. These files are icf.dpi and icf.dpd

Set path to NULL to accept the default location.

**Early\_date** and **late\_date** are used to control the processing of the saved DPID against the current ICF. **Early\_date** is the first date DPID have been allocated, or the last ICF update; **late\_date** is the most recent date that DPID have been allocated. These dates can be null, indicating the user is taking responsibility for determining that the saved DPID are from the previous PAF version.

```
int
DPE_SearchIcf(QIS qisv,
char *query,
char *buffer, unsigned int *ret_cnt, unsigned max_len)
```

Takes a DPID and lets you know if the dpid is listed in the ICF change list. This is for a quick scan for users who hold DPID long term in their database to see which addresses may have changed.

ERR is DPRC\_DPID if it has changed, DPRC\_OK if not changed.

```
int
DPE_Verify(QIS qisv,
char *query,
char *buffer, unsigned int *ret_cnt, unsigned max_len)
```

Takes a DPID flagged by the function **DPE\_SearchIcf** and the corresponding address. Requires previous call to **OpenIcfFiles**.

The logical steps performed by this function:

- Run address through ordinary search. If search succeeds, return corrected address, and changed DPID.
- If fails, look up DPID on the ICF list. If not on list, or deleted, then finished query. The returned DPID is zero.
- For changed records, alter user fields as indicated by ICF file.

Run this altered record through search engine again, output resulting match or non-match.

## PAFlink API Programming for Data Append

The great benefit of integrating the Data Append functions into an application system is that the data can be returned in real-time as each record is being validated.

### An Overview of the Data Append Functions to Call

#### **DPE\_OpenDpidDataFiles**

This function opens the extra files needed for a Data Append lookup. This requires extra database file to have been installed. These files are **dpiddata.dpi** and **dpiddata.dpd**.

#### **DPE\_SearchDpidData**

This function performs a lookup of the Data Append data files. If the DPID is contained within the file the associated data will be returned. The Data Append data files contain a record for each DPID which is currently valid even if no additional data has been associated with that DPID. Requires previous call to **OpenDpidDataFiles**.

## Data Append API Functions in Detail

```
int
DPE_OpenDpidDataFiles(char *inpath);
```

This function opens the extra files needed for a data append lookup. The extra database files, **dpiddata.dpi** and **dpiddata.dpd**, must have already been installed. Set inpath to NULL to accept the default location.

This function is not strictly necessary, as the files will be otherwise be automatically opened the first search that requires them. However, this verifies the files are present and that the time taken to load them happens at a predictable processing step. This function is also needed for multi-threaded applications (since the threading is only safe across calls involving the QIS variable, not for the openfiles steps), and also should be done if database files are going to be loaded into memory by a subsequent call to **DPE\_DataFilesInMemory**.

The return code for the function is **DPERR\_OK** if OK, otherwise one of the codes given in the file dpecodes.h

```
int
DPE_SearchDpidData(QIS qisv,
char *query,
char *buffer, unsigned int *ret_cnt, unsigned max_len);
```

This function uses a given DPID to search for additional data such as the 2001 CCD code. The default template, if none is given, is for input: DPI, and for output:

**ERR|DPI|DD0|DD1|DD2|DD3|DD4|DD5|DD6|DD7|DD8|DD9.**

This function will not perform any address lookup functions, and it expects you to provide a DPID.

The function parameters are used in the same way as the function **DPE\_SearchAMAS**, **DPE\_SearchNearest** and identically patterned functions. The return code for the function is **DPERR\_OK** if OK, otherwise one of the codes given in the file dpecodes.h. Generally, if not **DPERR\_OK**, then this is a processing error, not a search failure as the data file contains a record for each current DPID.

## Thread Safe Functionality and Multi-threaded Executable

The PAFlink engine is thread safe subject to the following details. Threading provides two speed advantages:

- I/O can be overlapped with CPU processing (use two worker threads), and
- where there are multiple CPUs (use as many worker threads as is appropriate for the number of CPUs available).

### Thread-safety for API users

To get correct operation from the PAFlink API in a threaded environment the following conditions should be met.

The following functions are not thread safe and should be called once only, at program initialisation and program termination:

```
DPE_OpenFiles  
DPE_OpenReverseFiles  
DPE_OpenIcFiles  
DPE_CloseFiles
```

The following functions are thread safe but are best called serially in a main thread. They obtain and dispose the QIS handle.

```
DPE_NewQis  
DPE_DisposeQis
```

The following functions are thread safe provided they are called with a different QIS handle (i.e. obtain one QIS handle per working thread). All these functions take a QIS handle as a parameter.

```
DPE_SetFlagValue  
DPE_GetFlagValue  
DPE_SetAnswerFormat  
DPE_SetInTemplate  
DPE_SetOutTemplate  
DPE_SearchAMAS  
DPE_SearchReverse  
DPE_SearchNearest  
DPE_Barcode  
DPE_Sortplan  
DPE_GetLastCode  
DPE_GetCountValues
```

**DPE\_SearchIcf**  
**DPE\_Verify**

The following new functions have been provided to handle errors inside the functions that use a QIS handle, and are thread safe alternatives to the global error routines.

**DPE\_LastErrStr**  
**DPE\_SetLastErr**

The following functions are not thread safe, and are provided for backwards compatibility. They are used when a function from the above list indicates an error. For functions that use a QIS handle, there are new error functions (immediately above) that are thread safe.

**DPE\_ErrStr**  
**DPE\_SetErr**

## Multithreaded Executable

In addition to the PAFlnt engine or libraries being thread safe, there is a new executable that takes advantage of multithreading, called **paflntkmt**.

This executable can take one extra parameter on the command line, '-m <number>', where <number> is the number of threads you desire.

By default, **paflntkmt** starts 4 worker threads.

**Paflntkmt** takes care of synchronisation of input and output, template synchronisation etc. Because of this, **paflntkmt** takes all the other parameters, input files, templates etc. of the normal PAFlnt executable.

## Template Examples

An example of an input template might be: **"Adr:Loc:Stt:Pcd"**. This indicates that queries will have the format address lines, followed by a locality, state and postcode.

Another template might simply be: **"Adr"**. This would indicate a completely unparsed address.

Note that the address parsing function is only invoked when the template item ADR (or Adr) is used.

An example of an output template might be: **"ERR:DPI:Adr"**. This asks for an error code (e.g. street not found, postcode incorrect - these are non fatal errors), a DPID and corrected address back.

If the address fully separated and parsed was desired the output template might be:

```
FUT:FUN:BLT:BLN:BG1:BG2:ALN:TN1:TS1:TN2:TS2:Thn:Tht:Tts:Pdt:PDP:PDN:PDS:Loc:
STT:PCD:DPI
```

Here is an example (long lines are wrapped for the document only):

The "T", "O" and "Q" Line Commands from the batch program are shown here for illustration. The appropriate function call would be used to invoke the action.

```
T
|IGN|FUT|IGN|FUN|FUS|BLT|BLN|BG1|BG2|IGN|ALN|IGN|TN1|TS1|IGN|TN2|TS2|THN|THT
|TTS|PDT
|PDP|PDN|PDS|LOC|STT|PCD|CTY|IGN|IGN|IGN|IGN|IGN|IGN|IGN|IGN|UNK|IGN|IGN|IGN
|IGN|IGN|IGN
O
|ERR|DPI|FUT|FUN|FUS|BLT|BLN|BG1|BG2|ALN|TN1|TS1|TN2|TS2|THN|THT|TTS|PDT|PDP
|PDN|PDS |LOC|STT|PCD
Q 1||18|||||||3||5|GEDDES|ST|||||VICTORIA PARK|WA|6100|||||||D
|11164739|19981021|18/3-5 GEDDES ST|VICTORIA PARK WA 6100|
Q 2||30|||||||5||7|PARK|AVE|||||CRAWLEY|WA|6009|||||||D|11164965
|19981021|30/5-7 PARK AVE|CRAWLEY WA 6009|
Q
3|F|9|||||||32||34|HAYLES|AVE|||||ARCADIA|QLD|4819|||||||D|1132101
8 |19981021|F 9 32-34 HAYLES AVE|ARCADIA QLD 4819|
```



If we just wanted to pass the address, but get the corrected parsed result as well, the file would be:

```
T |ADR
O
|ERR|DPI|FUT|FUN|BLT|BLN|BG1|BG2|ALN|TN1|TS1|TN2|TS2|THN|THT|TTS|PDT|PDF|PDN
|PDS|LOC
|STT|PCD
Q 18/3-5 GEDDES ST,VICTORIA PARK WA 6100
Q 30/5-7 PARK AVE,CRAWLEY WA 6009
Q FLAT 9 32-34 HAYLES AVE,ARCADIA QLD 4819
```

In the output results, every input will have an output response, similar to the following. (for search nearest ('N' queries) there may be several 'A' lines).

```
A 0|53789493||18|||||3||5||GEDDES|ST|||||VICTORIA PARK|WA|6100
A 0|59947953||30|||||5||7||PARK|AVE|||||CRAWLEY|WA|6009
A 0|77526059|F|9|||||32||34||HAYLES|AVE|||||ARCADIA|QLD|4819
```

## Search Nearest Strategies

The PAFlink batch 'N' query and the API interface **DPE\_SearchNearest** are used to input a partial address, and return a list of near matches or partial address matches.

Previous versions of the PAFlink Address Matching Engine did not accept ADR as an input column type (or rather accepted but did nothing with it). It was expected that an incomplete address would be given, one that a parser could not cope with.

The PAFlink Address Matching Engine now parses ADR input fields, but a reasonable address should be given. That is, it should have either a postcode and or suburb, a state, a street name and street number. This is because the address will go through the parser, expecting all these items.

In addition, if you use the ADR field, then the address is expected to be correct up to the street number level. That is, the extra searching and correcting performed in the SearchAMAS will not be done in SearchNearest.

**Note !** One thing to watch with the 'N' query is that no response may be returned. This means that no match occurred. This is no problem for the API interface (the `ret_cnt` returned count will indicate zero). (You can easily change this in the `paftest` program, which is distributed on the CD as C source code).

The SearchNearest will only do a search if a street name is given. This is best qualified with at least a postcode or suburb.

The SearchNearest is a relatively slow search. If you have a choice of SearchNearest or SearchAMAS the latter will be much quicker. It is not a sensible idea to build an address parser around the idea of looking each word up to find a street match.

**Note !** When doing a SearchNearest, if you give a street number, a PO box number (called a postal delivery number) or a lot number, it is assumed that the postcode, suburb and street details are already exactly correct. This means that these details will not be fuzzy-matched and corrected. To “automatically” match and correct an address, use SearchAMAS or SearchAdt.

If you give these details, it is assumed that you probably already did a search nearest on street name and suburb, and picked one of the values returned. This is the initial reason that the ADR field was not supported.

If you give a detail like a flat number, second number of a street range or floor number, then the search is unlikely to work without a street number.

A search nearest with a street number returns all the addresses that start with that street number. Thus if you entered '1', then 1, 10 to 19, 100 to 199 etc. would all be candidates. It is well to remember that the SearchNearest function returns only up to the maximum buffer size you have specified. You may get a `DPERR_TRUNCATE` fault if you ask a query that returns too much data.

Some strategies that work well when using SearchNearest are:

#### **Using postcode.**

If a postcode is given, this is used to narrow searches before other fields. Thus if a suburb and postcode is given, the suburb is looked for within the postcode. Then the partial street name is examined.

#### **Using suburb.**

There are two ways of searching for a suburb. You can use the MLC (sounds like) field, or the LOC field. In either case, you can restrict the search further by using the state field. In using the MLC field, you request a “sounds like” suburb, up to the number of characters you have entered.

This means that if you entered 'VICTORIA' you should match suburbs like VICTORIA PARK. A single character can also be used.

The difference between MLC and LOC is that the LOC also does a sounds like search (up to the number or characters entered), but LOC also requires that the exact characters match up to the number entered.

#### **Using street name.**

The street name can be used in conjunction with suburb and or postcode, or on its own.

Once again you can use MTH (sounds like street name) or THN, street name. You can also use the PDT, postal delivery type field in conjunction with THN. For example you may have streets starting with 'P', but you may also want to catch 'PO BOX'.

If you use MTH (without THN), you are asking for “sounds like” street names, without restricting street names to match exactly. All of these searches restrict matches up to the length you have entered. Thus you can ask for all streets starting with 'K'.

Here is an example query, looking for all the streets that start with 'AL' in a suburb that sounds like 'VICTORY' (or starts off sounding like 'VICTORY').

```
T :THN:MLC
O :THN:THT:LOC:PCD
N AL:VICTORY
A ALANDALE:AVE:FIGTREE:2525
A ALBERT:PL:VICTOR HARBOR:5211
A ALMOND:AVE:VICTOR HARBOR:5211
A ALTMAN:AVE:VICTOR HARBOR:5211
A ALBERT:ST:VICTORY HEIGHTS:4570
A ALENOLA:ST:VICTORY HEIGHTS:4570
A ALEXANDER:PL:VICTORY HEIGHTS:4570
A ALFRED:ST:VICTORY HEIGHTS:4570
A ALICE:ST:VICTORY HEIGHTS:4570
A ALLARADO:PL:VICTORY HEIGHTS:4570
A ALLEN:LANE:VICTORY HEIGHTS:4570
A ALMA:ST:VICTORY HEIGHTS:4570
A ALARNA:ST:VICTORIA POINT:4165
A ALBATROSS:ST:VICTORIA POINT:4165
A ALBERT:ST:VICTORIA POINT:4165
A ALEXANDER:AVE:VICTORIA POINT:4165
A ALISON:CT:VICTORIA POINT:4165
A ALLEN:ST:VICTORIA POINT:4165
A ALBANY:HWY:VICTORIA PARK:6100
```

Here is another query, asking for streets that sound like 'VICTORY' in suburbs that sound like 'VICTORY':

```
T :MTH:MLC
O :THN:THT:LOC:PCD
N VICTORY:VICTORY
A FIGTREE:CRES:FIGTREE:2525
A VICTORIA VALLEY:RD:VICTORIA VALLEY:7140
A VICTORIA:ST:VICTOR HARBOR:5211
A VICTORY:ST:VICTORY HEIGHTS:4570
A VICTORY:LANE:VICTORY HEIGHTS:4570
A VICTORIA:RD:VICTORY HEIGHTS:4570
A FIG TREE:RD:VICTORY HEIGHTS:4570
A FIG TREE POCKET:RD:FIG TREE POCKET:4069
```

The PAFlink Rapid search tool uses the following input templates after a part-street name has been entered:

One of **Thn** or **Mth** (depending if preceded by '#'), one of **PCD**, **Loc** or **Mlc** (depending if a postcode was entered, or a suburb name or suburb name starting with '#') and **Pdt**.

The entered part-street name is inserted into the **Thn** (or **Mth**) and also the **Pdt** field, because Rapid cannot be sure, if the user entered “po” for instance, whether it is intended to locate PORT ST or PO BOX.

After an initial part-street number has been entered, PAFlink rapid uses the following template to get street numbers. Many fields are filled in as a result of the previous street level query.

**TN1 : ALN : PDP : PDN : Thn : Tht : Tts : Pdt : Loc : STT : PCD**

The entered part-street number is inserted into the **TN1** field when a street name was selected from the previous step. If the number is prefixed by “lot” the part-street number is inserted into the **ALN** field. If a postal delivery type was selected, the part-street number is inserted into the **PDP/PDN** fields.

## Files for PAFlink Batch Engine

File Name(s)	Description
pafpardll.dll pafpardll.lib	These files provide the batch engine for Windows, including the parser interface. The engine is contained in the dll, the lib file satisfies the 'C' references.
pafdpdll.dll pafdpdll.lib	These files provide the batch engine for Windows, excluding the parser. The engine is contained in the dll, the lib file satisfies the 'C' references. This engine is most suitable for pre-parsed data eg. via a rapid address lookup process.
pafparlib.lib	This is the batch engine for Windows, and requires the parser libraries.
pafdpelib.lib	This is the batch engine for Windows, but doesn't require the parser libraries.
pafparlib.so	This is the engine in a shared library for Unix. You would only have this file if you unpacked one of the system specific tar files for one of the Unix systems. This requires the parser shared libraries to work.
pafdpelib.so	This file provides the batch engine for Unix, excluding the parser. You would only have this file if you unpacked one of the system specific tar files for one of the Unix systems. This engine is most suitable where the data is already parsed eg. via a rapid address lookup process.
libcgap.dll GapEngine.dll GapEnglish.dll AuBase.dll aufilters.dll AusGap.dll	These files are the parser executable dll's for Windows. Several of these files are specified by the paflink.meta file.
libcgap.so libgap.so libaubase.so libaufilters.so libgapenglish.so	These files are the parser executable shared libraries for Unix. Several of these files are specified by the paflink.meta file.
paflink.meta	<p>This file that directs the behaviour of the parser. Unfortunately it differs between Unix and Windows, and for static libraries and dynamic libraries. Be sure to unpack the system specific packed Unix tar file, containing the Unix version of paflink.meta.</p> <p>On platforms where it is possible to run a version of PAFlink with both shared libraries and static libraries two versions of paflink.meta are provided. One, the default for shared libraries is the standard paflink.meta. The second paflink.meta.static is used when the statically linked version of PAFlink is desired. You must rename this paflink.meta.static to paflink.meta in order to use it. Be sure you are using the static version of the executable, by making sure the <code>paflink.sh</code> shell script invoked it correctly.</p>

---

abbreviation.txt	These files are used by the address parser, as specified by the <b>paflink.meta</b> file.
building.txt	
delivery.txt	
company.txt	
floor.txt	
job_title.txt	
locality_qualifier.txt	
lot.txt	
ordinal.txt	
reverse_ordinal.txt	
state.txt	
street_name.txt	
street_type.txt	
street_suffix.txt	
suburb.txt	
suburb_noise.txt	
unit.txt	

---

## Test Files

File Name(s)	Description
dpe.h dpecodes.h paftest.c ccdtest.c	<p>Included with the PAFlink installation are the dpe.h file (header file), dpecodes.h (header file with manifest constants) and a paftest.c file. These and the library files or dlls are all that is required to build a paftest executable. ccdtest.c is included with PAFlink CCD Append installations.</p> <p>See paftest.c, or ccdtest.c, for details of how to try out this program.</p> <p>There is also a small sample test file, testfile.in, with the results in testfile.out. The PAFlink CCD Append versions are ccdtestfile.in and ccdtestfile.out</p>
Makefile.test Makeccd.test	<p>These files help you to build your own application that uses the paflink API. The Makefiles are setup for Linux, for other platforms they are easily modified.</p>
ptest.pl	<p>This sample perl script shows how the PAFlink engine can be controlled as a sub-program. This is a much easier interface than building the paflink engine as a library, but this only works under Unix.</p>

---



# TCP/IP Server Engine

---

## Introduction

The Acxiom PAFlink Server Engine, **dpserver**, is a wrapper around the ordinary PAFlink Engine. This wrapper allows TCP/IP connections to the engine for queries. Throughout this document, reference will be made to functionality of the Engine.

Uses for dpserver might include:

- Remote access to the client/server access to the PAF and PAFlink engine
- Internet based DPID assignment
- Another non-programming method of controlling the DPID allocation mechanism

**dpserver** is available for Windows NT Server, Linux and Unix systems. Two sample clients are provided, one in Perl and one as a Windows DLL.

For a full description of the capabilities of the **dpserver**, and technical advice on its deployment, contact PAFlink Sales or PAFlink Support.

# Dpserver

## Dpserver Command Synopsis

```
Dpserver      [ -l <data-directory>] [ -q <maximum-
returned-buffer-size>]
[ -t <maximum-idle-timeout-(secs)>] [ -p <port>]
[ -d <debug-level>]
```

## Dpserver Description

**dpserver** services DPID allocation requests. Requests are described in the Perl module Dpt.pm (included on the distribution CD). **dpserver** uses the PAFlink batch engine.

The queries that can be done of the engine are the same as those of the batch engine.

Further documentation can be seen in the Perl module Dpt.pm, Dptest.pl and in the PAFlink User Guide. Typical queries involve setting an input template, output template and performing a query.

**dpserver** accepts connections on the nominated port, forks and handles queries. It initialises all its files on start up, and before the first connection is accepted. For this reason there is very little overhead in making new connections. However, for the same reason, it is not a good idea to place an entry for **dpserver** in the inetd.conf file, as the initial start up for PAFlink is quite lengthy (up to 5 seconds).

The **dpserver** does not allow reverse DPID lookups.

## Dpserver Options

Flag	Name	Description
-l <data-directory>	Path	The directory where the data files are located. By default these are in /usr/local/paflink. This is the default directory if this parameter is not given.
-q <maximum-returned-buffer-size>	Max Buffer	This can be changed to limit or expand request sizes. The default size is 256000 bytes.
-t <maximum-idle-time>	Max Idle	This allows dpserver to shut down a connection after no activity has occurred. The default time is 10 minutes. The value that can be specified here is in seconds.
-p <port>	Port	Port number to use. A name can be used if there is an entry in the /etc/services file.
-d <debug-level>	Debug	1 means report errors. Higher numbers give more chatty information.

## Dpt - Perl Interface to Dpserver

### Synopsis

```
require 'Dpt';

my ($dobj, $err) = new Dpt($host_address, $host_port);
($errs) = $dobj->ValidateUser ( $name, $pass);
($errs) = dobj->CheckAlive ();
($results, $errs) = dobj->Query ($query);
dobj->Close();

See Dptest.pl for examples of use.
```

### Description

```
my ($dobj, $err) = new Dpt($host_address, $host_port);
```

If `host_port` is not given, 9890 is used. If `host_address` is not given, `localhost` is used.

Returns the object as the first returned value, and or the errors in `$err`. If `$err` is an empty string, `$dobj` is the correct object.

```
($errs) = $dobj->ValidateUser ( $name, $pass);
```

Defaults are used for `$name` and `$pass`, if not given. This is a placeholder for further functional expansion.

If an error is found, it is returned as a non empty string in `$errs`.

```
($errs) = dobj->CheckAlive ();
```

If the remote server is not alive, `$errs` will return a non zero length string.

The remote server is configured to run with a timeout. If you do not do a `CheckAlive` or `Query` within the timeout period, the server will drop the connection. The default time is 10 minutes.

```
($results, $errs) = dobj->Query ($query);
```

Queries can be one of the types of lines described in the `paflink` programmers guide. For example 'T' for input template, 'O' for output template and 'Q' for search exact.

A valid query might be:

```
`T |ADR\nO |ERR|DPI|ADR\nQ 12 CAROTTA ST,ARTARMON NSW\n'
```

The results are returned in the first return parameter, any errors in the second parameter. Note that truncation can occur with 'N' (nearest) searches, which can return a lot of data.

This is controlled by the server, and can be configured with command line switches. The default value is 256000 bytes.

Note that as long as you keep your connection open, your templates remain in effect (for your connection).

```
dobj->Close();
```

This closes the connection.

## Dperem – Windows Interface to Dpserver

The 32 bit Windows DLL, dperem.dll, and the include file dperem.h constitute a method for querying the dpserver from a Windows machine.

The simplest way of making your query is to use the following function:

```
int DLLEXPORT
DPER_OneShot(
    LPSTR server,
    LPSTR port,
    char near_exact,
    char insep,
    LPSTR intemplate,
    char outsep,
    LPSTR outtemplate,
    LPSTR query,
    LPSTR retbuf,
    unsigned int maxlen,
    unsigned int *ret_cnt);
```

This function opens a connection to the server, performs the query, fills your buffer, and closes the connection.

**server** is the name of your server (or the IP address).

**port** is the port of the server. Use "0" for the default.

**near\_exact** use 'N' for search nearest queries, 'Q' for exact.

**insep** the input separator character - examples might be '|' or ':' or ','.

**intemplate** the input template. See the Programmer's Guide for the types.

**outsep** the output separator character - examples might be '|' or ':' or ','.

**outtemplate** the output template. See the Programmer's Guide for the types.

**query** the query string (without the 'Q' at the front).

**retbuf** the return buffer. Make this big enough to hold responses.

**maxlen** the size of the return buffer.

**\*ret\_cnt** the number of responses returned.

The return code is 0 for OK, or the fault number as specified elsewhere in the Programmers Reference Manual.

# Linking Applications to PAFlink

## Introduction

To use the PAFlink API, you will need to have your application linked to the following sets of libraries (listed by their standard base names only):

	Windows	Unix	VMS
<b>PAFlink</b> <sup>(1)</sup>	pafparlib libcgap ausgap aufilters gapenglish gap aubase	pafparlib <sup>(4)</sup> libcgap libausgap libaufilters libgapenglish libgap libabase	pafall
<b>GCC</b> <sup>(2)</sup>	n/a	libstdc++ libgcc	n/a

Depending on your distribution and platform, the library names may have the following extensions:

	Windows	Unix		VMS
		HP-UX	Others	
<b>Static</b> <sup>(3)</sup>	.lib	.a	.a	.olb
<b>Shared</b> <sup>(3)</sup>	.dll	.sl	.so	n/a

### Things to note:

1. You might notice in your distribution a library, not mentioned above, called “*pafdpelib*”. This library is basically the same as “*pafparlib*”, but does not contain an address parser. Your application should only be linked to either one of these libraries, but not both.

You would typically use “*pafparlib*” to get PAFlink to parse your raw, unparsed input address and return the appropriate DPID. “*pafdpelib*” would be used to get PAFlink to verify an address which had already been parsed. “*pafparlib*” is required if you intend to use one or more of the following input templates: **ADR**, **AD1** through **AD7**.

2. On HP-UX, “*pafparlib*” and “*pafdpelib*” are called “*libpafpar*” and “*libpafdpe*” respectively.
3. The distribution you receive may contain both sets of static and shared libraries. You should not mix linking static and shared libraries (PAFlink only) to your application. Please refer to the example *Makefile* below for more information.
4. For gcc, *libstdc++* is only needed if your application is in C or otherwise linked with a non-C++ application. For a C++ application, this library is already supplied.

Finally, to avoid “unresolved symbols” problems which occur with some linkers, the libraries should be linked in the order specified above.



As an example, a typical ***Makefile*** containing the relevant commands for linking an application to PAFlink on a Unix system is shown below:

```
PAF_PATH = <path.to.your.PAFlink.installtion>
CC       = <your.compiler>
CFLAGS   = -I $(PAF_PATH)
LDFLAGS   = -L $(PAF_PATH)

# A typical way to link to the PAFlink libraries.
# If your PAFlink distribution contains both shared and
# static libraries, then the shared libraries would typically
# be linked to your application, unless you use a linker option
# to specify otherwise, or use an alternative form of PAF_LIBS as
# mentioned below.
PAF_LIBS = $(PAF_PATH)/pafparlib.so -lcgap -lausgap -laufilters -
lgapenglish -lgap -laubase

# Use this form of PAF_LIBS if you intend to make an explicit
# link to the PAFlink libraries. In this example, you are linking
# explicitly to the static libraries.
#PAF_LIBS = \
# $(PAF_PATH)/pafparlib.a\
# $(PAF_PATH)/libcgap.a\
# $(PAF_PATH)/libausgap.a\
# $(PAF_PATH)/libaufilters.a\
# $(PAF_PATH)/libgapenglish.a\
# $(PAF_PATH)/libgap.a\
# $(PAF_PATH)/libaubase.a
# Note: $(GCC_LIBS) is not needed if your compiler is GCC
GCC_LIBS = \
    -lstdc++\
    -lgcc
APP_OBJS = <object.files.for.your.application>

<your.application> : $(APP_OBJS)
    $(CC) $(LDFLAGS) -o $@ $(APP_OBJS) $(PAF_LIBS) $(GCC_LIBS)
...
```

# Glossary

## A

**ActiveX** - A loosely defined set of technologies developed by Microsoft. ActiveX is an outgrowth of two other Microsoft technologies called OLE (Object Linking and Embedding) and COM (Component Object Model). As a moniker, ActiveX can be very confusing because it applies to a whole set of COM-based technologies. Most people, however, think only of ActiveX controls, which represent a specific way of implementing ActiveX technologies.

### Address Format Standards, Australia

**Post** – The last line must contain Locality and Postcode, containing State as well is preferred (mandatory for CleanMail). Second last line must contain either Street Number and Name OR Postal Delivery Type and Number (where applicable). Upper cased and double spacing are preferred.

**AMAS** – Address Matching Approval System. A division of Australia Post.

**API** – Application Programming Interface. This is the specification that defines how the programmer can access the methods and variables of a set of classes.

## B

**Batch** – Processing data from a file rather than line at a time through an interface.

**Barcode** – Australia Post uses a 4-State barcode font to uniquely identify addresses and assist in mail sortation.

**BQP** – Barcode Quality Program. A division of Australia Post.

**BSP** - Barcode sort plan used for mail pre-sorting. Note: the BSP is based on a postcode, an address does not need a DPID to obtain a BSP.

## C

**Canonical** - authorised; recognised by canon law; accepted.

**CCD** – Census Collection District. Numeric code, assigned by the Australian Bureau of Statistics, given to a group of households, approximately 250, in the same geographic region. CCDs are used for assigning other Census data, existing geo-demographic profiles such as MOSAIC or LandScape and profiling data. CCDs are reviewed with each Census and the current codes provided are from 2001.

**CleanMail** – An Australia Post bulk mail initiative. Provides postal discount for lodgements of more than 300 unbarcoded mail pieces (same size) which meet the Address Format Standards.

**Column Codes, PAFlink** – Three character codes used to describe an address element or field input to or output from PAFlink. Also applies to Address Tune and PAFlink with CCD Append.

**COM** - Component Object Model. An open software architecture from DEC and Microsoft, allowing interoperation between ObjectBroker and OLE.

**CORBA** - Common Object Request Broker Architecture. An Object Management Group specification which provides the standard interface definition between OMG-compliant objects.

## D

**Deep Sorting** – Where Australia Post uses machines to sort Barcoded, BSP ordered mail into Postal Routes, Streets, Individual addresses and delivery order within a building.

**Delimiter** – A character that marks the beginning or end of a unit of data.

**Direct Tray, Pre-Sort** – A single tray of barcoded letters going to one barcode presort area. Tray must be sorted by BSP code then postcode.

**DPID** - Delivery Point Identifier. A unique eight digit code which is used by Australia Post to link an address to the delivery process

## G

**GAP** – Generic Address Parser. Acxiom built and maintained address parser used within PAFlink and Address Tune.

**GUI** – Graphical User Interface

## I

**ICF** – Incremental Change File

## L

**Line Commands, PAFlink** – Single character codes used to control PAFlink functions. Also used by Address Tune and PAFlink with CCD Append.

**Locality Synonym** – An alternative name for a locality. Under AMAS rules these are classed as valid or unacceptable.

**Lodgement** – The process of presenting bulk mail to Australia Post for delivery.

## M

**Manifest** - a list of passengers or an invoice of cargo for a vehicle (as a ship or plane). In postal terms this is a document used for lodging bulk mail to Australia Posts PreSort program. The manifest contains a summary of the postal lodgement.

**Minimum Addressing Requirements**  
– *see Address Format Standards*

**MMUA** – Major Mail Users Australia

## N

**NAF** - National Address File. An address reference database compiled and maintained by Australia Post. The PAF is extracted from the NAF.

**NSP** - National sort plan used for mail pre-sorting prior to BSP.

**NPSP** – *see NSP*

## O

**OLE** - Object Linking and Embedding. A distributed object system and protocol from Microsoft, also used on the Acorn Archimedes. OLE allows an editor to "farm out" part of a document to another editor and then re-import it. For example, a desk-top publishing system might send some text to a word processor or a picture to a bitmap editor using OLE.

**Operating System** – The software used to operate a computer. Generic operating system examples are Windows or Unix. Specific operating system examples are Windows NT, HP-UX or VMS.

**Ordinal Number** - a number designating the place (as first, second, or third) occupied by an item in an ordered sequence.

## P

**PAF** - Postal Address File. An address reference database compiled and maintained by Australia Post. While it is the most comprehensive address source, it is not exhaustive.

**PAFlink engine** - Drives interaction between the PAF file and the chosen interface. Pre-built, not platform independent. Written in ANSI C. Also available as a library. Can support multi-processor and multiple concurrent threads.

**Parse** - to resolve into component parts.

**Phantom Primary Point** – This is the name given to a primary point to which no delivery can be made, such as the street address of a commercial building where there is no reception area. To minimise return mail, if an address match is made to a Phantom Primary Point, and no secondary information is available, the DPID may not be returned.

**Postal Delivery Address** – e.g. PO Box, GPO Box or Community Mail Agent (CMA).

**Postal Manifest** – *see Manifest*

**Pre-Sort** – An Australia Post bulk mail initiative. Provides postal discount for lodgements of more than 300 barcoded mail pieces (same size) which meet the Address Format Standards. Must contain at least on direct tray.

**Primary Point** – A block of flats will have one delivery point designated for the street address this is known as the primary point. Each flat or unit within the block will also have a delivery point these are known as secondary points and differ to the primary point DPID.

## R

**Rapid** – Processing data line at a time through an interface rather than from a file. Also known as Rapid Key.

**Realtime** – Accessing PAFlink or Address Tune through the API. Generally the integration of PAFlink or Address Tune into a third party application.

**Residue Tray - Barcoded, PreSort** – A tray of barcoded letters going to more than one barcode pre-sort area. Tray must be sorted by BSP code then postcode.

**Residue Tray - Unbarcoded, PreSort** – A single tray of unbarcoded letters going to one barcode pre-sort area. Tray must be sorted by BSP code then postcode.

## S

**Secondary Point** – See Primary Point

**Soundex Matching** – Matching based on the way a word sounds rather than the way which it is spelt.

**Street Address** – An address which relates to a building or a delivery point within a building e.g. Level 9 151 Clarence St, SYDNEY NSW 2000.

**Synonym** – A word or an expression that serves as a figurative or symbolic substitute for another.

## T

**TCP/IP** – Transmission Control Protocol / Internet Protocol. The suite of communications protocols used to connect hosts on the internet. It is the de facto standard for transmitting data over networks.

**Template, PAFlink** – Method used for describing input and outputs to PAFlink. Made up of a combination of Line Commands and Column Codes. Also used by Address Tune and PAFlink with CCD Append.

## V

**Valid Synonym** – A valid synonym is one which may be returned in place of the official name under the AMAS rules. Currently only used for localities, rules of use vary between street addresses and postal delivery addresses.

**Vanity Suburb** – A suburb name which differs from the official suburb name. Generally found with addresses near the border of suburbs.