A F D S O F T W A R E

AFD Common API

Unified Access to AFD Address Management Solutions

Desktop Integration Guide

February 2016

Table of Contents

1. Introduction

The AFD Common API provides full access to the AFD API for all our products. The API is easy to use and quick to implement, while balancing that with providing full flexibility. This enables you to rapidly develop using the API with practically any development environment to provide the data that you require. All AFD products provide rapid lookup and search functionality allowing you to implement address management solutions and provide bank data, account and card validation.

Our address management products are fully interchangeable with the Common API, meaning that you can include the name and all address fields in your integration even if you are only using our lowest level Postcode product. Your integration will then function fully with our Postcode Plus or Names & Numbers product should you, or your customer, wish to upgrade in the future. Similarly if you only develop for one product now, it's easy to add fields and features from another later without having to learn a whole new API.

To make life as easy as possible, the AFD Common API comes with a Wizard which will generate sample projects and code for the major development environments. The AFD Common API is also backed up by our free customer support services. You can visit our website at www.afd.co.uk/support for full developer support with using our API.

2. Getting Started

We recommend that for the most rapid development and to help you know where to start that you use our API Wizard to generate a sample project for your development environment. If your environment is not one that is listed, then select one that is closest to your own and use that as a basis for your coding. By looking at a sample project you can get a look and feel for how the API works and what you can do with it and you can easily copy and paste the code from that into your own and adapt it to meet your needs. Our sample projects work in the way that your own application is most likely to work, but also keep code to an absolute minimum, whilst being well commented, so that you can transfer the code with ease. The API Wizard also provides the code to go into a module or class in your application with all our API declarations and constants included which you can copy and paste into your own application. The code for lookup and search functionality is also provided and can be similarly copied and pasted.

3. Standard API

3.1. General Declarations

3.1.1. Type or Structure

All code calling the Common API will need to include a module, class, or header and cpp file (depending on the environment) which includes the declarations required to use the AFD Common API. This file can be included in any project and contains all you need to easily use the full functionality of the API in accordance with your needs.

This code will start with a type or structure declaration which contains all the fields for the product type that you are integrating. This will take account of options you may have selected, for example the address format. By providing all available fields you can easily see the data which may be available and take advantage of it. You should always note that not all fields may return data for all underlying products and not all fields are searchable. For a list of the fields available in each product and to find out which ones are searchable please refer to the appropriate appendix:

Appendix A – Address Management Fields
Appendix B – BankFinder Fields
Appendix C – Nearest Fields

While we would recommend that you keep all fields present, should you wish to thin this out, you can remove any unwanted fields, as long as you also remove them from the field specification string described below.

VB Type returning only basic Address fields and fields necessary for lookup and result retrieval:

```
Type afdAddressData
  Lookup As String * 255
  Name As String * 120
  Organisation As String * 120
  Property As String * 120
  Street As String * 120
  Locality As String * 70
  Town As String * 30
  Postcode As String * 10
  PostcodeFrom As String * 8
  Key As String * 255
  List As String * 512
End Type
```

C++ Structure returning only basic Address fields and fields necessary for lookup and result retrieval:

```
struct afdAddressData {
  char Lookup[256];
  char Name[121];
  char Organisation[121];
  char Property[121];
  char Street[121];
  char Locality[71];
  char Town[31];
```

```
  char Postcode[11];
  char PostcodeFrom[9];
  char Key[256];
  char List[513];
  afdAddressData(){     // constructor - zero the contents
    clear();
  }
  void clear(){
    memset(this,'\0',sizeof(*this));
  }
};
```

Note that the C++ declaration has fields one character larger than the VB one as we are allowing for the addition of a null terminator. The C++ structure also has code to clear the structure negating the need for an additional method to do this.

3.1.2. Function Declaration

Next comes the function declaration which is used to perform all operations with the Common API. This is the AFDData function, found in the afddata.dll (or afddata64.dll for 64-bit systems). It has the following parameters:

DataName (String)

Operation (Long)

tData (Any)
fields to use to lookup and return results.

The function returns a long which is the result code. This will be >= 0 if the function is successful, or < 0 in the case of an error (constants for this are described below).

Example VB Declaration for AFDData:

```
Public Declare Function AFDData Lib "afddata.dll" (ByVal dataName As String, ByVal
operation As Long, tData As Any) As Long
```

Example C++ Declaration for AFDData:

```
long __stdcall AFDData(char* dataName, long operation, char* tData);
typedef long(__stdcall *AFDDATA)(char* dataName, long operation, char* tData);
```

3.1.3. Field Specification String

A field specification string is described next, this will vary between the different product types (Address Management, BankFinder and Nearest). Its purpose is to tell the Common API the product type in use and the fields required as well as any additional options. It is a string in the following format:

{Data Name}@{Options}**<**{Refiner Options}**>{**{International}**}**@{Field List}

{Data Name} will be one of the following:

Address	Address Management Products
BankFinder	AFD BankFinder
Nearest*	Nearest Integration
List	Functions to list the alias localities for any address

	As well as retrieving lists of possible field values (Names & Numbers and TraceMaster only).
Grids	Grid Reference related utility functions
Email	Email utility function
String - Deprecated	String utility functions

* With Nearest this will be followed by your database details in the following format:

Nearest:{DBType}:{DBName}:{UID}:{PWD}:{SQL}:{Primary}

Where:

DBType:	The type of database, O=ODBC, A=Access, P=Paradox, X=Xbase
DBName:	The DSN or database file name (should contain > in place of :)
UID:	Any user name needed to connect to the database (ODBC Only)
PWD:	Any password needed to connect to the database (ODBC Only)
SQL:	The SQL string to use to query the data (N/A for FoxPro/Xbase)
Primary:	The Primary Key field

{Options} - One or more of the following options can be used as required:

U – Specifies that the structure passed is in Unicode (Wide Bytes)

L – Specifies that list items should not contain a Tab. Tabs are useful as they help align results correctly with each other, however some environments have list boxes which do not support these and so this option allows them to be omitted.

X – Specifies that Null Terminators should be used rather than space padded values (particularly useful in languages such as C/C++)

F – Specifies that Individual Fields will be retrieved from the Common API rather than a structure – see Appendix I for more information.

G - Specifies that approximate grid references for the locality or town of the address will be supplied for any location which does not contain a grid reference for that postcode (for example some non-geographical addresses, Isle of Man and Channel Islands etc.).

R - Specifies that Royal Mail Postzon grid references are used in-preference to GeoRef grid references.

{Field List} – A list of fields and there lengths to retrieve. (See Appendix A, B or C as appropriate for a list of the possible fields). These are each specified in the following format:

{Field Name 1}:{Field Length 1}@...{Field Name n}:{Field Length n}

Where
{Field Name x} – Specifies the name of the field
{Field Length x} – Specifies the length of the field

Example VB Declaration for the Field Specification string matching the VB type previously given:

```
Public Const afdFieldSpec =
"Address@@Lookup:255@Name:120@Organisation:120@Property:120@Street:120@Locality:70@Tow
n:30@Postcode:10@PostcodeFrom:8@Key:255@List:512"

Example C++ Declaration for the Field Specification string matching the C++ structure
previously given:

static char afdFieldSpec[2048] =
"Address@LX@Lookup:256@Name:121@Organisation:121@Property:121@Street:121@Locality:71@T
own:31@Postcode:11@PostcodeFrom:9@Key:256@List:513";
```

Note that when using Nearest the GBGridE, GBGridN, and List fields also specify the name of the field in your database table to use for that field in pointed brackets, e.g.

GBGridE<GridE>:10@GBGridN<GridN>:10@List<Miles, Title>:10

{Refiner Options}
Refiner API users can also add a set of advanced cleaning options, if they are required, to the end of the options portion of the field specification string, enclosing them in pointed brackets, e.g. <0AS>.

The options supported are as follows: (Please see the main Refiner manual for more detail regarding each of these options)

0 - Specifies the default cleaning mode where the address is fully cleaned

1 - Specifies that the postcode should be verified only

2 – Specifies that only full matches should be returned

3 – Uses Attach Mode only (fields are returned based on the postcode)

N – Use non-separated fields (Useful for databases where fields are not seperated, e.g. the street and town are entered on the same line with no comma etc. between them)

A – No Ambiguous Matches (do not return list of addresses to choose from if the address cannot be uniquely matched)

S – No Suggested Matches (do not return a suggested match along with the original address if the address cannot be matched but there is a possible unique match)

U – Assume the Postcode is correct (this option allows less reliable matching on the assumption that the postcode is correct if the address cannot otherwise be verified. In Attach mode this allows a property and postcode to be matched)

T – Give Ambiguous Matches in Preference to Street Level (if an address cannot be uniquely matched to an individual property the original property information is normally retained, this option gives the ambiguous addresses to choose from instead).

P – Match PO Box Last (Some PO Box addresses contain some Street address information too even though the address is meant for a PO Box. If you wish Refiner to try and match it to a street address first then select this option).

L – Retain Alias Localities (If the address is matched using an alias locality this will be retained in the address – Alias Localities are not normally retained as they are not required for the address to be deliverable).

O – Do not move data to Organisation (Normally Refiner will put additional address data for street level only matches in the property field unless they look like an Organisation or there is already a property. Specifying this option ensures Refiner never returns such data in the Organisation field - useful if you are not going to use the Organisation field returned).

W – Do not use the Default DPS (if an address is not matched to a full Delivery Point Record, a default of 9Z is assigned which can still be used for printing bar codes etc., if you do not wish this to be used then use this option)

F – Do not use Field Placement (By default if an address cannot be matched Refiner attempts to format the address correctly on return, if you would rather it was left “as-is” then use this option.

3.1.4. Function Type Constants

Next a set of constants are defined which specify the lookup and search operations available. The DLL supports the following operations:

Constant	Value	Description
AFD_POSTCODE_LOOKUP	0	Carries out a standard postcode (or zipcode) lookup from the data specified in the Lookup field. (Not BankFinder)
AFD_POSTCODE_PROPERTY_LOOKUP	1	Carries out a lookup based on a postcode or combination of property name/number and a postcode. (Address Management Only)
AFD_MULTIPLE_FASTFIND_LOOKUP	1	Like a FastFind lookup for Nearest but where a specified locality or town matches multiple locations the user is presented with a list to choose from. (Nearest Only)
AFD_FASTFIND_LOOKUP	2	Full fast-find functionality, allowing either a postcode or an address portion to be entered to find the address.
AFD_SEARCH	3	Reverse search, set fields to specify reverse search criteria. (See Appendix A for details of which fields are searchable in which products). Fields not

		searchable will be ignored if specified.
AFD_RETRIEVE_RECORD	4	Retrieves a previous record from a lookup/search. Useful when you add items to a list box, using the List field and then wish to retrieve the item the user clicks on. Set the Key field to use this operation with the value of the Key field that was returned from the original lookup/search for the record you want.
AFD_ACCOUNT_VALIDATE	5	Used to validate a supplied sortcode and account number (BankFinder only)
AFD_CARD_VALIDATE	6	Used to validate a supplied card number and optional expiry date (BankFinder only)
AFD_CLEAN	7	Used to clean an address (requires a Refiner API license)
AFD_GET_NEXT	32	Should be specified with any of the lookup or search operations for subsequent calls to obtain the next matching result (END_OF_SEARCH,-6, will be returned if there are no further results to return).
AFD_LIST_BOX	64	Specify with any of the lookup/search operations if you wish the DLL to display a listbox for you rather than having to use your own in the case of multiple results. Calls to AFD_GET_NEXT are not needed in this case as the API will only return the result the user selects.
AFD_SHOW_ERROR	128	Set this option if you require the DLL to display any error message (e.g. if no results are found) to the user itself.

Example VB Constant Declarations:

```
Public Const AFD_POSTCODE_LOOKUP = 0
Public Const AFD_POSTCODE_PROPERTY_LOOKUP = 1
Public Const AFD_MULTIPLE_FASTFIND_LOOKUP = 1
Public Const AFD_FASTFIND_LOOKUP = 2
Public Const AFD_SEARCH = 3
Public Const AFD_RETRIEVE_RECORD = 4
Public Const AFD_ACCOUNT_VALIDATE = 5
Public Const AFD_CARD_VALIDATE = 6
Public Const AFD_CLEAN = 7
Public Const AFD_GET_NEXT = 32
Public Const AFD_LIST_BOX = 64
Public Const AFD_SHOW_ERROR = 128
```

Example C++ Constant Declarations:

```
// Function Type Constants
```

```
#define AFD_POSTCODE_LOOKUP 0
#define AFD_POSTCODE_PROPERTY_LOOKUP 1
#define AFD_MULTIPLE_FASTFIND_LOOKUP 1
#define AFD_FASTFIND_LOOKUP 2
#define AFD_SEARCH 3
#define AFD_RETRIEVE_RECORD 4
#define AFD_ACCOUNT_VALIDATE 5
#define AFD_CARD_VALIDATE 6
#define AFD_CLEAN 7
#define AFD_GET_NEXT 32
#define AFD_LIST_BOX 64
#define AFD_SHOW_ERROR 128
```

3.1.5. Skip Constants – UK Address Management Only

For address management products skip constants are provided next which can be added to the operation parameter for calls to the AFDData function to skip records, for example to return the first record on a postcode only.

The available options are as follows:

Constant	Value	Description
AFD_NO_SKIP	0	Default – all matching records are returned
AFD_ADDRESS_SKIP	512	Only the first record per address (e.g. first listed resident) is returned. Only has any effect in Names & Numbers.
AFD_POSTCODE_SKIP	1024	Only the first record per postcode is returned.
AFD_SECTOR_SKIP	1536	Only the first record in each postcode sector is returned. (A postcode sector is the portion of the postcode before the space plus the first digit after it, e.g. B11 1 is a sector).
AFD_OUTCODE_SKIP	2048	Only the first record per Outcode is returned. The Outcode is the portion of the postcode before the space, e.g. B11.
AFD_POST_TOWN_SKIP	2560	Only the first record per Post Town, e.g. Birmingham is returned.
AFD_POSTCODE_AREA_SKIP	3072	Only the first record per Postcode Area is returned. A Postcode Area is the letters at the start of the postcode, e.g. B11 1AA is in Postcode Area B, IM8 is in Postcode Area IM.

Example VB Constant Declarations:

```
Public Const AFD_NO_SKIP = 0
Public Const AFD_ADDRESS_SKIP = 512
Public Const AFD_POSTCODE_SKIP = 1024
Public Const AFD_SECTOR_SKIP = 1536
Public Const AFD_OUTCODE_SKIP = 2048
Public Const AFD_POST_TOWN_SKIP = 2560
Public Const AFD_POSTCODE_AREA_SKIP = 3072
```

Example C++ Constant Declarations:

```
// Function Type Constants
#define AFD_NO_SKIP 0
#define AFD_ADDRESS_SKIP 512
#define AFD_POSTCODE_SKIP 1024
#define AFD_SECTOR_SKIP 1536
#define AFD_OUTCODE_SKIP 2048
#define AFD_POST_TOWN_SKIP 2560
#define AFD_POSTCODE_AREA_SKIP 3072
```

3.1.6. Clearing System Constants – BankFinder Only

The clearing system constants allows you to restrict the results that come back to those which are solely on the UK (BACS) Clearing System or the Irish (IPSO Clearing System), or both systems. Obviously if you are only able to clear through the UK clearing system you should specify this to return results for the UK system only. This constant should be added to the operation parameter for calls to the AFDData function.

The available options are as follows:

Constant	Value	Description
AFD_BOTH_CLEARINGS	0	Default – all matching records are returned
AFD_UK_CLEARING	512	Only records on the UK (BACS) Clearing System are returned
AFD_IRISH_CLEARING	1024	Only records on the Irish (IPSO) Clearing System are returned

Example VB Constant Declarations:

```
Public Const AFD_BOTH_CLEARINGS = 0
Public Const AFD_UK_CLEARING = 512
Public Const AFD_IRISH_CLEARING = 1024
```

Example C++ Constant Declarations:

```
// Function Type Constants
#define AFD_BOTH_CLEARINGS 0
#define AFD_UK_CLEARING 512
#define AFD_IRISH_CLEARING 1024
```

3.1.7. Success Code Constants

These specify the possible success codes returned from any API function:

Constant	Value	Description
AFD_RECORD_BREAK	0	The search/lookup has not completed but may take some time and so is returning to give the user the option to cancel a long search.
AFD_SUCCESS	1	The function was successful and a matching record has been returned.
AFD_SUCCESS_ NO_VALIDATION	2	This applies only to Bankfinder account number validation and indicates that the function was successful and the account number should be taken as valid. However, as account numbers on this sortcode cannot be validated you may wish to double check it is correct.

3.1.8. Error Code Constants

These specify the possible errors returned from any API function:

Constant	Value	Description
AFD_ ERROR_INVALID_FIELDSPEC	-1	The field specification string specified is invalid. This shouldn't be returned under normal circumstances.
AFD_ ERROR_NO_RESULTS_FOUND	-2	No records matching your lookup or

		search criteria were found.
AFD_ERROR_INVALID_ ... _RECORD_NUMBER	-3	The record number provided (e.g. when re-retrieving an item from a list box) is invalid.
AFD_ERROR_OPENING_FILES	-4	An error occurred attempting to open the AFD data files. Check they are correctly installed.
AFD_ERROR_FILE_READ	-5	An error occurred reading the data. Likely to be due to corrupt data so software may need to be re-installed.
AFD_ERROR_END_OF_SEARCH	-6	End of Search (when the last result has already been called off – indicates there are no more results to return).
AFD_DATA_LICENSE_ERROR	-7	Indicates there is an error with the product registration. Normally due to it having expired. Run the Welcome program to re-register the software.
AFD_ERROR_CONFLICTING_ .. _SEARCH_PARAMETERS	-8	Occurs if you attempt to search for a Name and Organisation at the same time. Also occurs with Postcode Plus if the UDPRN field is searched for at the same time as any other field.
AFD_USER_CANCELLED	-99	Indicates that the user clicked the cancel button if the DLL internal list box was used.
The following fields apply to BankFinder validation operations only		
AFD_ERROR_SORTCODE_NOT_FOUND	-12	The sort code specified for an account number validation does not exist.
AFD_ERROR_INVALID_SORTCODE	-13	The sortcode specified for an account number validation is invalid.
AFD_ERROR_INVALID_ACCOUNT_NUMBER	-14	The account number specified for an account number validation is invalid.
AFD_ERROR_INVALID_ROLL_NUMBER	-21	The sort code and account number given are for a building society account which also requires a roll number for account credits. No roll number has been supplied or is incorrect for this building society.
AFD_ERROR_INVALID_IBAN	-22	The International Bank Account Number provided is in an invalid format
AFD_ERROR_UNRECOGNISED_COUNTRY	-23	The IBAN provided contains a country that is not recognised as valid
AFD_ERROR_IBAN_MISMATCH	-24	Both an IBAN and Account Number was provided and these details do not match.
AFD_ERROR_INVALID_EXPIRY	-15	The expiry date specified for a card validation is invalid.
AFD_ERROR_CARD_EXPIRED	-16	The card has expired
AFD_ERROR_INVALID_CARD_NUMBER	-18	The card number specified for a card validation is invalid.
AFD_ERROR_VISA_ATM_ONLY	-19	The card number specified is a Visa card which can be used in an ATM only.
AFD_ERROR_UNRECOGNISED_ .. _CARD_TYPE	-20	While the card number appears to be a valid one, the card is not of any of

		the known types and is therefore unlikely to be acceptable for payment.

Example VB Constant Declarations:

```
Public Const AFD_ERROR_INVALID_FIELDSPEC = -1
Public Const AFD_ERROR_NO_RESULTS_FOUND = -2
Public Const AFD_ERROR_INVALID_RECORD_NUMBER = -3
Public Const AFD_ERROR_OPENING_FILES = -4
Public Const AFD_ERROR_FILE_READ = -5
Public Const AFD_ERROR_END_OF_SEARCH = -6
Public Const AFD_ERROR_DATA_LICENSE_ERROR = -7
Public Const AFD_ERROR_CONFLICTING_SEARCH_PARAMETERS = -8
Public Const AFD_USER_CANCELLED = -99
```

Example C++ Constant Declarations:

```
#define AFD_ERROR_INVALID_FIELDSPEC -1
#define AFD_ERROR_NO_RESULTS_FOUND -2
#define AFD_ERROR_INVALID_RECORD_NUMBER -3
#define AFD_ERROR_OPENING_FILES -4
#define AFD_ERROR_FILE_READ -5
#define AFD_ERROR_END_OF_SEARCH -6
#define AFD_ERROR_DATA_LICENSE_ERROR -7
#define AFD_ERROR_CONFLICTING_SEARCH_PARAMETERS -8
#define AFD_USER_CANCELLED -99
```

3.1.9. Refiner Status Code Constants

Refiner clean operations return a cleaning constant >= 100 or <= -100 which indicates the status of the cleaning operation. These constants are as follows:

Constant	Value	Description
AFD_ REFINER_PAF_MATCH	100	Address verified from Postcode and matches a record in PAF identically.
AFD_REFINER_POSTCODE_MATCH	200	Address verified from the Postcode and matches a record in PAF with some correction.
AFD_REFINER_CHANGED_POSTCODE	201	Address verified from a postcode which was substituted due to a Royal Mail recoding and now matches a record in PAF.
AFD_REFINER_ASSUME_POSTCODE_ CORRECT	202	Match was made with the Assume Postcode Correct option enabled only and the address could only be verified on the assumption that the postcode was correct.
AFD_REFINER_ASSUME_CHANGED_ POSTCODE_CORRECT	203	Match was made with the Assume Postcode Correct option enabled and the address could only be verified on the assumption that the postcode was correct after a Royal Mail recoding change.
AFD_REFINER_ASSUME_POSTCODE_ ADDED_PROPERTY	204	Match was made with the Assume Postcode Correct option enabled and the address could only be verified on the assumption that the postcode was correct and the property was added in.
AFD_REFINER_ASSUME_CHANGED_ POSTCODE_ADDED_PROPERTY	205	Match was made with the Assume Postcode Correct option enabled and

		the address could only be verified on the assumption that the postcode was correct after a Royal Mail recoding change and the property was added in.
AFD_REFINER_FULL_DPS_MATCH	300	Address verified to PAF with some correction, looking wider than just the specified Postcode.
AFD_REFINER_FULL_DPS_MATCH_NO_ ORG	301	Address verified to PAF with ambiguous organisation which was not in the original address so has been omitted.
AFD_REFINER_FULL_DPS_MATCH_ LIMITED	302	Match was made with the Assume Postcode Correct option enabled and the Address was verified to PAF to a more limited degree.
AFD_REFINER_STREET_MATCH	400	Address verified to Street Level, i.e. the property was not on PAF, but a unique match to the street was identified on a single postcode.
AFD_REFINER_NO_MATCH_FOUND	-101	No Match Found - Refiner has been unable to match this record.
AFD_REFINER_AMBIGUOUS_POSTCODE	-102	Ambiguous Postcode Match - Refiner has matched this record to Street Level but cannot determine which is the correct Postcode and so has presented each of the possibilities.
AFD_REFINER_SUGGEST_RECORD	-103	Suggested Match. Refiner has given a possibility that this address could match to as it is unique but there was not enough to be certain of a correct match.
AFD_REFINER_AMBIGUOUS_MATCH	-104	Ambiguous Match. Refiner has given several possibilities that this address could match to.
AFD_REFINER_INTERNATIONAL_ ADDRESS	-105	This address was detected as being an International Address and therefore cannot be cleaned as data is only present for cleaning UK, Channel Isles and Isle of Man addresses.
AFD_REFINER_NO_RECORD_DATA	-106	No record data was supplied. Refiner cannot clean this address as no address data was given.

Example VB Constant Declarations:

```
Public Const AFD_REFINER_PAF_MATCH = 100
Public Const AFD_REFINER_POSTCODE_MATCH = 200
Public Const AFD_REFINER_CHANGED_POSTCODE = 201
Public Const AFD_REFINER_ASSUME_POSTCODE_CORRECT = 202
Public Const AFD_REFINER_ASSUME_CHANGED_POSTCODE_CORRECT = 203
Public Const AFD_REFINER_ASSUME_POSTCODE_ADDED_PROPERTY = 204
Public Const AFD_REFINER_ASSUME_CHANGED_POSTCODE_ADDED_PROPERTY = 205
Public Const AFD_REFINER_FULL_DPS_MATCH = 300
Public Const AFD_REFINER_FULL_DPS_MATCH_NO_ORG = 301
Public Const AFD_REFINER_FULL_DPS_MATCH_LIMITED = 302
Public Const AFD_REFINER_STREET_MATCH = 400
Public Const AFD_REFINER_NO_MATCH_FOUND = -101
Public Const AFD_REFINER_AMBIGUOUS_POSTCODE = -102
Public Const AFD_REFINER_SUGGEST_RECORD = -103
Public Const AFD_REFINER_AMBIGUOUS_MATCH = -104
```

```
Public Const AFD_REFINER_INTERNATIONAL_ADDRESS = -105
Public Const AFD_REFINER_NO_RECORD_DATA = -106

Example C++ Constant Declarations:

#define AFD_REFINER_PAF_MATCH 100
#define AFD_REFINER_POSTCODE_MATCH 200
#define AFD_REFINER_CHANGED_POSTCODE 201
#define AFD_REFINER_ASSUME_POSTCODE_CORRECT 202
#define AFD_REFINER_ASSUME_CHANGED_POSTCODE_CORRECT 203
#define AFD_REFINER_ASSUME_POSTCODE_ADDED_PROPERTY 204
#define AFD_REFINER_ASSUME_CHANGED_POSTCODE_ADDED_PROPERTY 205
#define AFD_REFINER_FULL_DPS_MATCH 300
#define AFD_REFINER_FULL_DPS_MATCH_NO_ORG 301
#define AFD_REFINER_FULL_DPS_MATCH_LIMITED 302
#define AFD_REFINER_STREET_MATCH 400
#define AFD_REFINER_NO_MATCH_FOUND -101
#define AFD_REFINER_AMBIGUOUS_POSTCODE -102
#define AFD_REFINER_SUGGEST_RECORD -103
#define AFD_REFINER_AMBIGUOUS_MATCH -104
#define AFD_REFINER_INTERNATIONAL_ADDRESS -105
#define AFD_REFINER_NO_RECORD_DATA -106
```

3.1.10. AFDErrorText Function

This is a helper function that the Wizard will generate, which will convert an error code (return value less than zero) to a message which explains the error. This makes it easy to simply use this function to obtain text to display in the case of an error. Text is included for each of the error codes listed in the Error Code Constants section above.

3.1.11. AFD RefinerCleaningText Function

This is a helper function that the Wizard will generate, which will convert a return code from the Common API when using the AFD_CLEAN option to a message which explains the error. This makes it easy to simply use this function to obtain text to display in the case of an error. Text is included for each of the error codes listed in the Refiner Status Code Constants section above. Please note that this function is only useful if you are using Refiner API functionality with the appropriate license.

3.1.12. Clear Function

The wizard also generates a helper function to clear the AFD Type or Structure, which you should call prior to carrying out an operation using the API. This is either called ClearAFDAddressData or ClearAFDBankData for Address Management products and BankFinder respectively. The differing names allow these to co-exist in the same module if desired when using both products.

Note: This does not apply to C++ code as they include a clear function in the structure declaration itself.

3.1.13. afdInitDLL

Where necessary, e.g. in C++ a function is also included which will load the DLL and locate the AFDData function:

3.1.14. List Functions – Address Management Only

With Postcode Plus, Names & Numbers and TraceMaster products you can obtain the alias localities for any address or postcode if required. These are non-postally required localities held by Royal Mail which can or may be included on an address if desired. An example of this would be including Wimbledon for an address in London. You should note that these are stored at postal sector level (e.g. SW19 1) and there are often multiple entries for an address so a locality being returned does not mean it is necessarily the best one for the particular address you are viewing.

For Names & Numbers and TraceMaster products only it is also possible to obtain a list of possible values for most fields, e.g. all the Mailsort codes present, business descriptions, etc. You can also specify the start value of the field, e.g. return all surnames present starting with “Smith”.

When using International data you can also use the List functions to obtain a list of all available countries (names or ISO codes).

To use these functions an AFDListData structure should be declared containing the following fields:

Field Name	Length	Description
Lookup	255	In the case of retrieving an alias locality this should be the postcode or key of the address to obtain the alias localities for. In the case of Names & Numbers or TraceMaster lists this should either be blank to retrieve the full list, or contain the value you wish entries to start with.
List	255	Each matching locality name or list entry is returned, in turn, into this field.
Product	40	Optional: Would indicate the product used if desired.

A afdListFieldSpec string should also be declared and works as described in section 4.1.3.

The constants you can use with this function to specify the list operation you wish to perform are as follows:

Constant	Value	Description
AFD_LIST_ALIAS_LOCALITY	0	Returns all alias localities for the sector that the specified postcode or key resides in.
The following are applicable when using International data only:		
AFD_LIST_COUNTRY_ISO	3	Will return the ISO codes of all available countries.

AFD_LIST_COUNTRY	4	Will return the names of all available countries.
The following are applicable to Names & Numbers and TraceMaster Products Only: These all return a list of all entries of the data item specified in the data Setting the lookup parameter will restrict matches to only those items starting with the specified string.		
AFD_LIST_FORENAME	10	Returns Forenames (first names).
AFD_LIST_SURNAME	11	Returns Surnames
AFD_LIST_ORGANISATION	12	Returns Organisations
AFD_LIST_PROPERTY	13	Returns Properties
AFD_LIST_STREET	14	Returns Streets
AFD_LIST_LOCALITY	15	Returns Localities
AFD_LIST_TOWN	16	Returns Postal Towns
AFD_LIST_COUNTY	17	Returns Counties (This includes Postal, Traditional and Administrative County names)
AFD_LIST_MAILSORT_CODE	18	Returns Mailsort codes
AFD_LIST_URBAN_RURAL_CODE	19	Returns Urban Rural Codes
AFD_LIST_URBAN_RURAL_NAME	20	Returns Urban Rural Names
AFD_LIST_WARD_CODE	21	Returns Ward Codes
AFD_LIST_WARD_NAME	22	Returns Ward Names
AFD_LIST_CONSTITUENCY	23	Returns Constituencies
AFD_LIST_EER_CODE	24	Returns EER Codes (European Electoral Region Codes)
AFD_LIST_EER_NAME	25	Returns EER Names
AFD_LIST_AUTHORITY_CODE	26	Returns Local / Unitary Authority Codes
AFD_LIST_AUTHORITY	27	Returns Authority Names
AFD_LIST_LEA_CODE	28	Returns LEA Codes (Local Education Authority)
AFD_LIST_LEA_NAME	29	Returns LEA Names
AFD_LIST_TV_REGION	30	Returns TV Regions
AFD_LIST_NHS_CODE	31	Returns NHS Codes
AFD_LIST_NHS_NAME	512	Returns NHS Names
AFD_LIST_NHS_REGION_CODE	513	Returns NHS Region Codes
AFD_LIST_NHS_REGION_NAME	514	Returns NHS Region Names
AFD_LIST_PCT_CODE	515	Return CCG Codes
AFD_LIST_PCT_NAME	516	Return CCG Names
AFD_LIST_CENSATION_CODE	517	Returns Censation Codes
AFD_LIST_AFFLUENCE	518	Returns Censation Affluence Codes with descriptions
AFD_LIST_LIFESTAGE	519	Returns Censation Lifestage Codes with descriptions
AFD_LIST_ADDITIONAL_CENSUS_INFO	520	Returns Censation Additional Information with descriptions.
AFD_LIST_HOUSEHOLD_COMPOSITION	521	Returns Household composition codes with descriptions.
AFD_LIST_BUSINESS	522	Returns Business descriptions
AFD_LIST_SIZE	523	Returns Company Size catagories
AFD_LIST_SIC_CODE	524	Returns SIC Codes
AFD_LIST_COUNCIL_TAX_BAND	525	Returns Council Tax Bands
AFD_LIST_CONSTITUENCY_CODE	528	Returns Constituency Codes
AFD_LIST_SUB_COUNTRY_NAME	529	Returns Sub Country Names
AFD_LIST_DEVOLVED_CONSTITUENCY_CODE	531	Returns Devolved Constituency Codes
AFD_LIST_DEVOLVED_CONSTITUENCY_NAME	532	Returns Devolved Constituency Names

Example VB Constant Declarations for List Functions:

```
Public Const AFD_LIST_ALIAS_LOCALITY = 0
Public Const AFD_LIST_COUNTRY_ISO = 3
Public Const AFD_LIST_COUNTRY = 4
Public Const AFD_LIST_FORENAME = 10
Public Const AFD_LIST_SURNAME = 11
Public Const AFD_LIST_ORGANISATION = 12
Public Const AFD_LIST_PROPERTY = 13
Public Const AFD_LIST_STREET = 14
Public Const AFD_LIST_LOCALITY = 15
Public Const AFD_LIST_TOWN = 16
Public Const AFD_LIST_COUNTY = 17
Public Const AFD_LIST_MAILSORT_CODE = 18
Public Const AFD_LIST_URBAN_RURAL_CODE = 19
Public Const AFD_LIST_URBAN_RURAL_NAME = 20
Public Const AFD_LIST_WARD_CODE = 21
Public Const AFD_LIST_WARD_NAME = 22
Public Const AFD_LIST_CONSTITUENCY = 23
Public Const AFD_LIST_EER_CODE = 24
Public Const AFD_LIST_EER_NAME = 25
Public Const AFD_LIST_AUTHORITY_CODE = 26
Public Const AFD_LIST_AUTHORITY = 27
Public Const AFD_LIST_LEA_CODE = 28
Public Const AFD_LIST_LEA_NAME = 29
Public Const AFD_LIST_TV_REGION = 30
Public Const AFD_LIST_NHS_CODE = 31
Public Const AFD_LIST_NHS_NAME = 512
Public Const AFD_LIST_NHS_REGION_CODE = 513
Public Const AFD_LIST_NHS_REGION_NAME = 514
Public Const AFD_LIST_PCT_CODE = 515
Public Const AFD_LIST_PCT_NAME = 516
Public Const AFD_LIST_CENSATION_CODE = 517
Public Const AFD_LIST_AFFLUENCE = 518
Public Const AFD_LIST_LIFESTAGE = 519
Public Const AFD_LIST_ADDITIONAL_CENSUS_INFO = 520
Public Const AFD_LIST_HOUSEHOLD_COMPOSITION = 521
Public Const AFD_LIST_BUSINESS = 522
Public Const AFD_LIST_SIZE = 523
Public Const AFD_LIST_SIC_CODE = 524
Public Const AFD_LIST_COUNCIL_TAX_BAND = 525
```

Example C++ Constant Declarations for List Functions:

```
// Function Type Constants
#define AFD_LIST_ALIAS_LOCALITY 0
#define AFD_LIST_COUNTRY_ISO 3
#define AFD_LIST_COUNTRY 4
#define AFD_LIST_FORENAME 10
#define AFD_LIST_SURNAME 11
#define AFD_LIST_ORGANISATION 12
#define AFD_LIST_PROPERTY 13
#define AFD_LIST_STREET 14
#define AFD_LIST_LOCALITY 15
#define AFD_LIST_TOWN 16
#define AFD_LIST_COUNTY 17
#define AFD_LIST_MAILSORT_CODE 18
#define AFD_LIST_URBAN_RURAL_CODE 19
#define AFD_LIST_URBAN_RURAL_NAME 20
#define AFD_LIST_WARD_CODE 21
#define AFD_LIST_WARD_NAME 22
#define AFD_LIST_CONSTITUENCY 23
#define AFD_LIST_EER_CODE 24
#define AFD_LIST_EER_NAME 25
#define AFD_LIST_AUTHORITY_CODE 26
#define AFD_LIST_AUTHORITY 27
#define AFD_LIST_LEA_CODE 28
#define AFD_LIST_LEA_NAME 29
#define AFD_LIST_TV_REGION 30
#define AFD_LIST_NHS_CODE 31
#define AFD_LIST_NHS_NAME 512
#define AFD_LIST_NHS_REGION_CODE 513
#define AFD_LIST_NHS_REGION_NAME 514
#define AFD_LIST_PCT_CODE 515
#define AFD_LIST_PCT_NAME 516
```

```
#define AFD_LIST_CENSATION_CODE 517
#define AFD_LIST_AFFLUENCE 518
#define AFD_LIST_LIFESTAGE 519
#define AFD_LIST_ADDITIONAL_CENSUS_INFO 520
#define AFD_LIST_HOUSEHOLD_COMPOSITION 521
#define AFD_LIST_BUSINESS 522
#define AFD_LIST_SIZE 523
#define AFD_LIST_SIC_CODE 524
#define AFD_LIST_COUNCIL_TAX_BAND 525
```

3.1.15. Utility Declarations – Address Management Only

These utility functions are not necessary for core address or bank validation functionality, but provide additional functionality that may be useful in your application. For full details of what these functions can do please refer to section 4.7 of this document.

3.1.16. String Utility Declarations – Depreciated and Unsupported

These are provided for compatibility with existing applications which may depend on them but for new developments we would recommend you use in-built functions which are included with most modern development environments. For the String Utility functions an AFDStringData structure is declared, containing the fields specified in Appendix E of this manual for String functions. An afdStringFieldSpec is also declared and works in the same way as the general field specification string documented earlier in this section. The following operation constants are also defined which are used to specify the string operation you wish to perform:

Constant	Value	Description
AFD_ STRING_SEARCH_REPLACE	0	All occurrences in the string specified in the Lookup field of the string specified in the Search field are replaced with the string in the Replace field.
AFD_STRING_SEARCH_REPLACE_CASE	1	This is the same as AFD_STRING_SEARCH_REPLACE but is case sensitive.
AFD_STRING_CAPITALISE	2	This corrects the capitalisation of the string specified in the Lookup field. For example 'commercial STREET' would become 'Commercial Street'.
AFD_STRING_CLEAN_LINE	3	This cleans the string specified in the Lookup field by removing spurious characters that should not be in an address line, e.g. a trailing comma.
AFD_STRING_CHECK_POSTCODE	4	This checks if the string specified in the Lookup field looks like a postcode.
AFD_STRING_CLEAN_POSTCODE	5	This cleans the postcode specified in the Lookup field to tidy up the postcode specified.
AFD_STRING_ABBREVIATE_COUNTY	6	This provides the Royal Mail Approved county abbreviation for the county specified in the Lookup field if one exists.

VB Declarations for String Utility Functions:

```
Public Type AFDStringData
  Lookup As String * 255
  Outcode As String * 4
  Incode As String * 3
  Search As String * 255
  Replace As String * 255
End Type
Public Const afdStringFieldSpec =
"String@@Lookup:255@Outcode:4@Incode:3@Search:255@Replace:255"
Public Const AFD_STRING_SEARCH_REPLACE = 0
Public Const AFD_STRING_SEARCH_REPLACE_CASE = 1
Public Const AFD_STRING_CAPITALISE = 2
Public Const AFD_STRING_CLEAN_LINE = 3
Public Const AFD_STRING_CHECK_POSTCODE = 4
Public Const AFD_STRING_CLEAN_POSTCODE = 5
Public Const AFD_STRING_ABBREVIATE_COUNTY = 6
```

C++ Declarations for String Utility Declarations:

```
struct afdStringData {
  char Lookup[256];
  char Outcode[5];
  char Incode[4];
  char Search[256];
  char Replace[256];
  afdStringData(){     // constructor - zero the contents
    clear();
  }
  void clear(){
    memset(this,'\0',sizeof(*this));
  }
};
static char afdStringFieldSpec[2048] =
"String@LX@Lookup:256@Outcode:5@Incode:4@Search:256@Replace:256";
#define AFD_STRING_SEARCH_REPLACE 0
#define AFD_STRING_SEARCH_REPLACE_CASE 1
#define AFD_STRING_CAPITALISE 2
#define AFD_STRING_CLEAN_LINE 3
#define AFD_STRING_CHECK_POSTCODE 4
#define AFD_STRING_CLEAN_POSTCODE 5
#define AFD_STRING_ABBREVIATE_COUNTY 6
```

3.1.17. Grid Utility Declarations (UK Address Management Only)

For the Grid Utility functions an AFDGridData structure is declared, containing the fields specified in Appendix E of this manual for Grid functions. An afdGridFieldSpec is also declared and works in the same way as the general field specification string documented earlier in this section. The following operation constants are also defined which are used to specify the grid operation you wish to perform:

Constant	Value	Description
AFD_GRID_CONVERT	512	Converts a GB or NI based grid reference, or latitude and longitude value to all other grid reference types and latitude and longitude values. (This uses a 1m resolution (6 digit). Using a constant of 0 rather than 512 uses 5 digit grids).
AFD_GRID_LOOKUP_LOCATION	513	Looks up a town, locality, or partial postcode specified in the Lookup field and provides an approximate grid reference for the location if a match is found (returns multiple results if there are multiple matches for this location). (This uses a 1m resolution (6 digit). Using a constant of 1 rather than 513 uses 5 digit grids).
AFD_GRID_DISTANCE	514	Calculates the distance between a pair of grid

		references or latitude and longitude values specified. (This uses a 1m resolution (6 digit). Using a constant of 2 rather than 514 uses 5 digit grids).

VB Declarations for Grid Utility Functions:

```
Public Type AFDGridData
  Lookup As String * 255
  GBGridE As String * 10
  GBGridN As String * 10
  NIGridE As String * 10
  NIGridN As String * 10
  Latitude As String * 10
  Longitude As String * 10
  TextualLatitude As String * 15
  TextualLongitude As String * 15
  Km As String * 6
  Miles As String * 6
  GBGridEFrom As String * 10
  GBGridNFrom As String * 10
  NIGridEFrom As String * 10
  NIGridNFrom As String * 10
  LatitudeFrom As String * 10
  LongitudeFrom As String * 10
  TextualLatitudeFrom As String * 15
  TextualLongitudeFrom As String * 15
End Type
Public Const afdGridFieldSpec =
"Grid@@Lookup:255@GBGridE:10@GBGridN:10@NIGridE:10@NIGridN:10@Latitude:10@Longitude:10
@TextualLatitude:15@TextualLongitude:15@Km:6@Miles:6@GBGridEFrom:10@GBGridNFrom:10@NIG
ridEFrom:10@NIGridNFrom:10@LatitudeFrom:10@LongitudeFrom:10@TextualLatitudeFrom:15@Tex
tualLongitudeFrom:15"
Public Const AFD_GRID_CONVERT = 512
Public Const AFD_GRID_LOOKUP_LOCATION = 513
Public Const AFD_GRID_DISTANCE = 514
```

C++ Declarations for Grid Utility Declarations:

```
struct afdGridData {
  char Lookup[256];
  char GBGridE[11];
  char GBGridN[11];
  char NIGridE[11];
  char NIGridN[11];
  char Latitude[11];
  char Longitude[11];
  char TextualLatitude[16];
  char TextualLongitude[16];
  char Km[7];
  char Miles[7];
  char GBGridEFrom[11];
  char GBGridNFrom[11];
  char NIGridEFrom[11];
  char NIGridNFrom[11];
  char LatitudeFrom[11];
  char LongitudeFrom[11];
  char TextualLatitudeFrom[16];
  char TextualLongitudeFrom[16];
  afdGridData(){     // constructor - zero the contents
    clear();
  }
  void clear(){
    memset(this,'\0',sizeof(*this));
  }
};
static char afdGridFieldSpec[2048] =
"Grid@@Lookup:256@GBGridE:11@GBGridN:11@NIGridE:11@NIGridN:11@Latitude:11@Longitude:11
@TextualLatitude:16@TextualLongitude:16@Km:7@Miles:7@GBGridEFrom:11@GBGridNFrom:11@NIG
ridEFrom:11@NIGridNFrom:11@LatitudeFrom:11@LongitudeFrom:11@TextualLatitudeFrom:16@Tex
tualLongitudeFrom:16";
#define AFD_GRID_CONVERT 512
#define AFD_GRID_LOOKUP_LOCATION 513
#define AFD_GRID_DISTANCE 514
```

3.1.18. Email Utility Declarations

For the Email Utility function an AFDEmailData structure is declared, containing the fields specified in Appendix E of this manual for Email functions. An afdEmailFieldSpec is also declared and works in the same way as the general field specification string documented earlier in this section. The following operation constants are also defined which are used to specify the level of email validation that you wish to perform:

Constant	Value	Description
AFD_EMAIL_FULL	0	Full email validation including live domain lookup
AFD_EMAIL_FORMAT	2	Validate email addres format is correct only
AFD_EMAIL_TLD	3	Validate email format is correct and the top level domain exists
AFD_EMAIL_LOCAL	4	Validate email format, top level domain and for well known domains carry out additional checks of the local portion of the address

VB Declarations for Email Utility Functions:

```
Public Type AFDEmailData
  Email As String * 255
End Type
Public Const afdEmailFieldSpec = "Email@@Email:255"
Public Const AFD_EMAIL_FULL = 0
Public Const AFD_EMAIL_FORMAT = 2
Public Const AFD_EMAIL_TLD = 3
Public Const AFD_EMAIL_LOCAL = 4
```

C++ Declarations for Email Utility Declarations:

```
struct afdEmailData {
  char Email[256];
  afdEmailData(){     // constructor - zero the contents
    clear();
  }
  void clear(){
    memset(this,'\0',sizeof(*this));
  }
};
static char afdEmailFieldSpec[2048] = "Email@@Email:256";
#define AFD_EMAIL_FULL = 0
#define AFD_EMAIL_FORMAT = 2
#define AFD_EMAIL_TLD = 3
#define AFD_EMAIL_LOCAL = 4
```

3.2. Lookup Function

The most commonly used function across our product range is the Lookup function. By entering a single string the user can find the results matching there lookup criteria.

With our address management products three lookup types are provided which you specify as the operation parameter in a call to AFDData:

Operation Constant	Functionality
AFD_FASTFIND_LOOKUP	This method is the most flexible, enabling the user to lookup an address simply by entering the postcode, or by using search criteria such as “Commercial Street, Birmingham” to quickly

	find matching records.
AFD_POSTCODE_PROPERTY_LOOKUP	This method allows the user to type in any postcode (or zipcode) and, optionally, include optional property information to find a match. For example “304, B11 1AA”. When full fastfind functionality is not required using this operation can prevent erroneous input causing long searches.
AFD_POSTCODE_LOOKUP	The user can type in any postcode (or zipcode), e.g. “B11 1AA” and obtain the results for that postcode. Only full correct postcodes are accepted. This is useful when you only want a postcode lookup, for example if you are looking up a list of postcodes to obtain grid references.

Similarily with Nearest, three lookup types are also provided (although they differ slightly due to the nature of the product):

Operation Constant	**Functionality**
AFD_FASTFIND_LOOKUP	This method is the most flexible, enabling the user to find the nearest simply by entering the postcode, or by entering a locality or town name, or a partial postcode.
AFD_MULTIPLE_FASTFIND_LOOKUP	This is similar to AFD_FASTFIND_LOOKUP, except that where a locality or town is given which has multiple matches the user will be presented with a list of locations to choose from to then lookup to find the Nearest.
AFD_POSTCODE_LOOKUP	The user can type in any postcode, e.g. “B11 1AA” and obtain the Nearest records to that postcode. Only full correct postcodes are accepted.

With BankFinder the only option available is AFD_FASTFIND_LOOKUP which allows you to find a bank using a sort code, postcode or other criteria quickly.

To carry out a lookup you will first need to declare an instance of the AFD structure you have declared in your general declarations module or class (see Section 4.1).

You will then need to set the Lookup parameter to the postcode or fast find string that you wish to look up.

If you are using International data you should also set the CountryISO or Country field to specify the country to carry out the lookup for.

If you are using Nearest you should also set the MaxRecords parameter to indicate the maximum number of records to return and the Miles or Km parameter to specify the maximum distance to return. Using low values for these options speeds up the lookup.

You then call the AFDData function with the following three parameters:

1. The Field Specification String (as detailed in Section 4.1)
2. The operation constant you require (one of the 3 above)
3. The instance of the structure or type that you declared.

If you would prefer not to use your own list box in your application, you may wish to add to the operation constant the AFD_LIST_BOX option. This causes the DLL to display a list box for you returning the record that the user selects, rather than returning all matching records to your application. This is only suitable for desktop applications as it displays the list box on-screen. Similarly adding AFD_SHOW_ERROR causes the DLL to display any error message to the user itself.

Should you wish to use one of the skip options in Address Management, for example returning the first record per sector only you can also add any of the Skip constants listed in the declarations (see Section 4.1).

When using BankFinder you may wish to add the clearing system you wish to restrict records to as well. Using AFD_UK_CLEARING restricts records to those on the UK (BACS) clearing system only. Using AFD_IRISH_CLEARING restricts records to those on the Irish (IPSO) clearing system only. If you can only clear through the UK system it is important to use the AFD_UK_CLEARING constant.

The AFDData function will return a negative value (less than zero) in the case of an error. Unless you have used the AFD_SHOW_ERROR option to ask the DLL to present any error to the user, you should display an error for the user before aborting the lookup. The AFDErrorText function will help you obtain a string which can be useful for displaying to the user to describe the error.

In the case of Address Management products the PostcodeFrom field of the structure or type will be set if a postcode was looked up which has changed following a Royal Mail recoding. The lookup will complete using the new postcode (found in the Postcode field), however you may wish to display a message notifying the user of this.

If the return value from the AFDData function is AFD_SUCCESS then a matching result has been returned and you can access the fields in the structure or type instance supplied to obtain full details for it. Included in this is a List property that can be used to provide a formatted item for adding to a list box to allow the user to select the desired option if desired. The Key property should also be stored as this allows quick retrieval of the record should it be selected using the ListFetch method described in Section 4.4.

If you have specified the AFD_LIST_BOX option then the user will have selected the required item and you can access the fields in the supplied structure or type instance and the lookup is complete.

Otherwise, you will have retrieved the first record which you can add to a list box if desired. If the return value was AFD_RECORD_BREAK then no result

has yet been returned but the lookup is taking some time (would not occur with a postcode or property, postcode lookup) and so the user is being given the chance to cancel.

To retrieve the rest of the records you should call the AFDData function as above repeatedly with the same operation code as before, but adding the AFD_GET_NEXT constant to it to obtain subsequent records. These can be added to a list box as above or processed as required. You should call AFDData in a lookup to retrieve these records allowing the user to cancel the lookup should it take some time or they realise they have entered something incorrectly.

Example VB code for an Address Management Lookup:

```
  Dim details As AFDAddressData
  Dim retVal As Long
  Static running As Boolean

  ' Prevent corruption of list box from button being clicked twice
  If running Then Exit Sub
  running = True

  ' Replace lstResult with the name of your list box for the results
  With lstResult

  ' Clear out any existing items in the list
  .Clear

  ' Reset Cancel flag
  cancelFlag = False

  ' Set the lookup
  details.Lookup = txtLookup.Text ' Change txtLookup to your lookup entry textbox

  ' Carry out the lookup (no need to alter the line below, unless you want to add a
sector skip option - see constants)
  retVal = AFDData(afdFieldSpec, AFD_FASTFIND_LOOKUP + AFD_SECTOR_SKIP, details)

  ' Abort with Message if error or user cancelled
  If retVal < 0 Then
    MsgBox AFDErrorText(retVal)
    running = False
    Exit Sub
  End If

  ' Display any changed postcode if applicable
  If Trim(details.PostcodeFrom) <> "" Then
    MsgBox "Postcode has changed from " + Trim(details.PostcodeFrom) + " to " +
Trim(details.Postcode)
  End If

  ' Now add matching records to the list box
  Do While retVal >= 0
    If retVal <> AFD_RECORD_BREAK Then
      ' Add the item to the list box with hidden key at the end
      .AddItem details.List + details.Key
    End If
    ' Give user the chance to cancel and allow list box to update
    DoEvents
    ' Check if user cancelled
    If cancelFlag Then
      MsgBox "Lookup Cancelled"
      running = False
      Exit Sub
    End If
    retVal = AFDData(afdFieldSpec, AFD_GET_NEXT + AFD_FASTFIND_LOOKUP, details)
  Loop

  ' Check results have been returned
  If .ListCount = 0 Then
```

```
    MsgBox "No Results Found"
  Else
    .ListIndex = 0 ' Select First item in the list
  End If

  End With

  running = False
```

Example C++ Code For an Address Management Lookup (Visual C++)

```
  HINSTANCE afdDLL = (HINSTANCE)NULL;
  AFDDATA afdData = (AFDDATA)NULL;
  static bool running = false;
  afdAddressData details;
  char listItem[2055];
  char msgTxt[255];
  long retVal;
  CListBox* listBox;
  MSG msg;

  // Check if we are already running to prevent crossing over items in the listbox
  if (running) return;
  running = true;

  // Load DLL
  if (!afdInitDLL(&afdDLL, &afdData)) {
    MessageBox("Error Loading afddata.dll", "Error", 0);
    return;
  }

  // Replace m_lstResult with the name given to a variable assigned to your list box
control for the results
  listBox = &m_lstResult;

  // Clear out any existing items in the list
  listBox->ResetContent();

  // Reset Cancel flag
  cancelFlag = false;

  // Update Data so we can read the lookup variable
  UpdateData(TRUE);

  // Set the lookup
  strcpy(details.Lookup, m_txtLookup); // Change this to your lookup entry textbox
value variable

  // Carry out the lookup (no need to alter the line below, unless you want to add a
sector skip option - see constants)
  retVal = (afdData)(afdFieldSpec, AFD_FASTFIND_LOOKUP, (char*)&details);

  // Abort with Message if error or user cancelled
  if (retVal < 0) {
    AFDErrorText(retVal, msgTxt);
    MessageBox(msgTxt, "Error", 0);
    running = false;
    return;
  }

  // Display any changed postcode if applicable
  if (details.PostcodeFrom[0] != '\0') {
    strcpy(msgTxt, "Postcode has changed from ");
    strcat(msgTxt, details.PostcodeFrom);
    strcat(msgTxt, " to ");
    strcat(msgTxt, details.Postcode);
    MessageBox(msgTxt, "Changed Postcode", 0);
  }

  // Now add matching records to the list box
  while (retVal >= 0) {
    if (retVal != AFD_RECORD_BREAK) {
      // make up list item with hidden key at the end
      strncpy(listItem, details.List, sizeof(details.List));
      strncpy(listItem + sizeof(details.List), details.Key, sizeof(details.Key));
      listItem[sizeof(details.List) + sizeof(details.Key)] = '\0';
```

```
      // Add the item to the list box
      listBox->AddString(listItem);
    }
    // Give user the chance to cancel and allow list box to update
    if(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }
    // Check if user cancelled
    if (cancelFlag) {
      MessageBox("Search Cancelled", "Cancelled", 0);
      return;
    }
    retVal = (afdData)(afdFieldSpec, AFD_GET_NEXT + AFD_FASTFIND_LOOKUP,
(char*)&details);
  }

  // Check results have been returned
  if (listBox->GetCount() == 0)
    MessageBox("No Results Found", "Error", 0);
  else {
    listBox->SetCurSel(0); // Select First item in the list

    OnSelchangeLstResult(); // Set this to your list change method to simulate
selecting the first list item

  }

  // free DLL instance
  FreeLibrary(afdDLL);
  afdDLL = (HINSTANCE)NULL;

  running = false;
```

3.3. Search Function

The search function allows records to be located by searching using specific fields rather than a general lookup string. It allows any of the Fields to be searched that are specified as being searchable for the AFD product that you are using in Appendix A (for Address Management products) or Appendix B (for BankFinder). All fields in your database are searchable in the case of Nearest.

To carry out a search you will first need to declare an instance of the AFD structure you have declared in your general declarations module or class (see Section 4.1).

You will then need to set the fields that you wish to search on to the criteria that you wish to use. Note that if you specify a field that is not searchable in the product that you are using it will be ignored.

If you are using International data you should also set the CountryISO or Country field to specify the country to carry out the lookup for.

You then call the AFDData function with the following three parameters:

1. The Field Specification String (as detailed in Section 4.1)
2. The operation code (AFD_SEARCH constant)
3. The instance of the structure or type that you declared.

If you would prefer not to use your own list box in your application, you may wish to add to the operation constant the AFD_LIST_BOX option. This causes the DLL to display a list box for you returning the record that the user selects, rather than returning all matching records to your application. This is only suitable for desktop applications as it displays the list box on-screen. Similarly adding AFD_SHOW_ERROR causes the DLL to display any error message to the user itself.

Should you wish to use one of the skip options in Address Management, for example returning the first record per sector only you can also add any of the Skip constants listed in the declarations (see Section 4.1).

When using BankFinder you may wish to add the clearing system you wish to restrict records to as well. Using AFD_UK_CLEARING restricts records to those on the UK (BACS) clearing system only. Using AFD_IRISH_CLEARING restricts records to those on the Irish (IPSO) clearing system only. If you can only clear through the UK system it is important to use the AFD_UK_CLEARING constant.

The AFDData function will return a negative value (less than zero) in the case of an error. Unless you have used the AFD_SHOW_ERROR option to ask the DLL to present any error to the user, you should display an error for the user before aborting the lookup. The AFDErrorText function will help you obtain a string which can be useful for displaying to the user to describe the error.

If the return value from the AFDData function is AFD_SUCCESS then a matching result has been returned and you can access the fields in the structure or type instance supplied to obtain full details for it. Included in this is a List property that can be used to provide a formatted item for adding to a list box to allow the user to select the desired option if desired. The Key property should also be stored as this allows quick retrieval of the record should it be selected using the ListFetch method described in Section 4.4.

If you have used the M and T options in the field specification to return all records at once from the API then you will have all matching records in the array you specified so the search is complete. If you have specified the AFD_LIST_BOX option then the user will have selected the required item and so you can access the fields in the supplied structure or type instance and the search is complete.

Otherwise, you will have retrieved the first record which you can add to a list box if desired. If the return value was AFD_RECORD_BREAK then no result has yet been returned but the search is taking some time (would not occur with a postcode or property, postcode lookup) and so the user is being given the chance to cancel.

To retrieve the rest of the records you should call the AFDData function as above repeatedly with the same operation code as before, but adding the AFD_GET_NEXT constant to it to obtain subsequent records. These can be

added to a list box as above or processed as required. You should call AFDData in a lookup to retrieve these records allowing the user to cancel the lookup should it take some time or they realise they have entered something incorrectly.

Example VB code for an Address Management Search:

```
  Dim details As AFDAddressData
  Dim retVal As Long

  Static running As Boolean

  ' Prevent corruption of list box from button being clicked twice
  If running Then Exit Sub
  running = True

  ' Replace lstResult with the name of your list box for the results
  With lstResult

  ' Clear out any existing items in the list
  .Clear

  ' Reset Cancel flag
  cancelFlag = False

  ' Clear Structure
  ClearAFDAddressData details

  ' Set the fields you wish to search on (look at the other properties of the
structure)
  details.Organisation = txtSearchOrganisation.Text
  details.Property = txtSearchProperty.Text
  details.Street = txtSearchStreet.Text
  details.Locality = txtSearchLocality.Text
  details.Town = txtSearchTown.Text
  details.Postcode = txtSearchPostcode.Text

  ' Carry out the search (no need to alter the line below, unless you want to add a
sector skip option - see constants)
  retVal = AFDData(afdFieldSpec, AFD_SEARCH + AFD_SECTOR_SKIP, details)

  ' Abort with Message if error or user cancelled
  If retVal < 0 Then
    MsgBox AFDErrorText(retVal)
    running = False
    Exit Sub
  End If

  ' Now add matching records to the list box
  Do While retVal >= 0
    If retVal <> AFD_RECORD_BREAK Then
      ' Add the item to the list box with hidden key at the end
      .AddItem details.List + details.Key
    End If
    DoEvents
    If cancelFlag Then
      MsgBox "Search Cancelled"
      running = False
      Exit Sub
    End If
    retVal = AFDData(afdFieldSpec, AFD_GET_NEXT + AFD_SEARCH, details)
  Loop

  ' Check results have been returned
  If .ListCount = 0 Then
    MsgBox "No Results Found"
  Else
    .ListIndex = 0 ' Select First item in the list
  End If

  End With

  running = False
```

Example C++ Code For an Address Management Search (Visual C++)

```
  HINSTANCE afdDLL = (HINSTANCE)NULL;
  AFDDATA afdData = (AFDDATA)NULL;
  static bool running = false;
  afdAddressData details;
  char listItem[2055];
  char msgTxt[255];
  long retVal;
  CListBox* listBox;
  MSG msg;

  // Check if we are already running to prevent crossing over items in the listbox
  if (running) return;
  running = true;

  // Load DLL
  if (!afdInitDLL(&afdDLL, &afdData)) {
    MessageBox("Error Loading afddata.dll", "Error", 0);
    return;
  }

  // Replace m_lstResult with the name given to a variable assigned to your list box
control for the results
  listBox = &m_lstResult;

  // Clear out any existing items in the list
  listBox->ResetContent();

  // Reset Cancel flag
  cancelFlag = false;

  // Update Data so we can read the search variables
  UpdateData(TRUE);

  // Set the search parameters (look at the other properties of the structure)
  strcpy(details.Organisation, m_txtSearchOrganisation);
  strcpy(details.Property, m_txtSearchProperty);
  strcpy(details.Street, m_txtSearchStreet);
  strcpy(details.Locality, m_txtSearchLocality);
  strcpy(details.Town, m_txtSearchTown);
  strcpy(details.Postcode, m_txtSearchPostcode);

  // Carry out the search (no need to alter the line below, unless you want to add a
sector skip option - see constants)
  retVal = (afdData)(afdFieldSpec, AFD_SEARCH, (char*)&details);

  // Abort with Message if error or user cancelled
  if (retVal < 0) {
    if (retVal != 99) { // User Cancelled
      AFDErrorText(retVal, msgTxt);
      MessageBox(msgTxt, "Error", 0);
      return;
    }
    running = false;
    return;
  }

  // Now add matching records to the list box
  while (retVal >= 0) {
    if (retVal != AFD_RECORD_BREAK) {
      // make up list item with hidden key at the end
      strncpy(listItem, details.List, sizeof(details.List));
      strncpy(listItem + sizeof(details.List), details.Key, sizeof(details.Key));
      listItem[sizeof(details.List) + sizeof(details.Key)] = '\0';
      // Add the item to the list box
      listBox->AddString(listItem);
    }
    // Give user the chance to cancel and allow list box to update
    if(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }
    // Check if user cancelled
```

```
    if (cancelFlag) {
      MessageBox("Search Cancelled", "Cancelled", 0);
      return;
    }
    retVal = (afdData)(afdFieldSpec, AFD_GET_NEXT + AFD_SEARCH, (char*)&details);
  }

  // Check results have been returned
  if (listBox->GetCount() == 0)
    MessageBox("No Results Found", "Error", 0);
  else {
    listBox->SetCurSel(0); // Select First item in the list

    OnSelchangeLstResult(); // Set this to your list change method to simulate
selecting the first list item

  }

  // free DLL instance
  FreeLibrary(afdDLL);
  afdDLL = (HINSTANCE)NULL;

  running = false;
```

3.4. List Fetch Function

Unless you are using the DLL's internal list box (i.e. specified the AFD_LIST_BOX constant at the time of your lookup or search) you may well have added each of the results from a lookup or search to a list box from which the user will select the required result. To retrieve the record they select you should use the Key Field which will have been returned with each result, and which you should have stored with the list items.

To fetch the record, you will first need to declare an instance of the AFD structure you have declared in your general declarations module or class (see Section 4.1). You should then set the Key Field to the value returned for the list item the user has selected.

If you are using International data you should also set the CountryISO or Country field to specify the country to carry out the lookup for.

You then call the AFDData function with the following three parameters:

1. The Field Specification String (as detailed in Section 4.1)
2. The operation code (AFD_RETRIEVE_RECORD constant)
3. The instance of the structure or type that you declared.

The AFDData function will return a negative value (less than zero) in the case of an error. It is unlikely that an error will occur at this stage, unless your key was in some way corrupted, but for completeness you can use the AFDErrorText function to help you obtain a string which can be useful for displaying to the user to describe the error.

You will now have the requested record and can use any of the fields in the structure to display or otherwise process the record details as desired.

You should note that with Nearest and the Multiple Fastfind Lookup operation if a location is returned you will obtain a Key starting “LOC:” followed by a grid reference. This should be looked up as a new lookup to get the Nearest results rather than retrieving a record.

Example VB code to fetch an item selected in the list for Address Management products:

```
  Dim details As AFDAddressData
  Dim pos As Long, retVal As Long

  ' Replace lstResult with the name of your list box for the results
  With lstResult

  ' Check a valid item is selected
  If .ListIndex = -1 Then
    MsgBox "No Item Selected"
    Exit Sub
  End If

  ' Set DLL parameters to retrieve the selected record
  details.Key = Mid(lstResult, 513) ' Replace lstResult with the name of your list box
for the results

  ' Finished with the list box
  End With

  ' Carry out the lookup (no need to alter the line below, unless you want to add a
sector skip option - see constants)
  retVal = AFDData(afdFieldSpec, AFD_RETRIEVE_RECORD, details)

  ' Abort with Message if error or user cancelled
  If retVal < 0 Then
    MsgBox AFDErrorText(retVal)
    Exit Sub
  End If

  ' Now Assign required fields to your application
  ' These are any of the members of the details. type (Use Trim to remove whitespace)
  txtName.Text = Trim(details.Name)
  txtOrganisation.Text = Trim(details.Organisation)
  txtProperty.Text = Trim(details.Property)
  txtStreet.Text = Trim(details.Street)
  txtLocality.Text = Trim(details.Locality)
  txtTown.Text = Trim(details.Town)
  txtPostcode.Text = Trim(details.Postcode)
```

Example C++ code to fetch an item selected in the list for Address Management products (Visual C++):

```
  HINSTANCE afdDLL = (HINSTANCE)NULL;
  AFDDATA afdData = (AFDDATA)NULL;
  afdAddressData details;
  bool foundSel = false;
  long retVal;
  CListBox* listBox;
  char lstStr[2055];
  char msgTxt[255];

  // Load DLL
  if (!afdInitDLL(&afdDLL, &afdData)) {
    MessageBox("Error Loading afddata.dll", "Error", 0);
    return;
  }

  // Replace m_lstResult with the name given to a variable assigned to your list box
control for the results
  listBox = &m_lstResult;

  // Set DLL parameters to retrieve the selected record
  listBox->GetText(listBox->GetCurSel(), lstStr);
  strncpy(details.Key, lstStr + sizeof(details.List), sizeof(details.Key));
```

```
  // Carry out the lookup (no need to alter the line below, unless you want to add a
sector skip option - see constants)
  retVal = (afdData)(afdFieldSpec, AFD_RETRIEVE_RECORD, (char*)&details);

  // Abort with Message if error
  if (retVal < 0) {
    AFDErrorText(retVal, msgTxt);
    MessageBox(msgTxt, "Error", 0);
    return;
  }

  // Now Assign required fields to your application
  // These are any of the members of the details. structure
  m_txtName = details.Name;
  m_txtOrganisation = details.Organisation;
  m_txtProperty = details.Property;
  m_txtStreet = details.Street;
  m_txtLocality = details.Locality;
  m_txtTown = details.Town;
  m_txtPostcode = details.Postcode;
  // Update Fields

  UpdateData(FALSE);

  // free DLL instance
  FreeLibrary(afdDLL);
  afdDLL = (HINSTANCE)NULL;
```

3.5. Account Number Validation – BankFinder Only

This function provides the ability to validate a sort code and account number. This checks that the account number is valid for the branch of the bank which the sortcode belongs to. This does not guarantee that the account number exists or sufficient funds exist for any transaction, but greatly cuts down on errors due to incorrectly entered numbers. The function will also translate any non-standard account numbers (e.g. a 10-digit account number).

To carry out a validation, you will first need to declare an instance of the AFDBankData structure you have declared in your general declarations module or class (see Section 4.1). You should then set the SortCode and AccountNumber Fields to the sort code and account number that you wish to validate (or instead the IBAN if validating an account number in that International standardised format). Optionally with Building Society credits you may also require a Roll Number.

You then call the AFDData function with the following three parameters:

1. The Field Specification String (as detailed in Section 4.1)
2. The operation code (AFD_ACCOUNT_VALIDATE constant)
3. The instance of the structure or type that you declared.

If you would prefer the DLL to display any error message that may occur to the user, rather than having to display this yourself, you should add the AFD_SHOW_ERROR constant to the operation parameter. This is only suitable for desktop applications as it displays any error message on-screen.

You may also need to add the clearing system you wish to restrict records to as well. Using AFD_UK_CLEARING restricts records to those on the UK (BACS) clearing system only. Using AFD_IRISH_CLEARING restricts

records to those on the Irish (IPSO) clearing system only. If you can only clear through the UK system it is important to use the AFD_UK_CLEARING constant.

The AFDData function will return a negative value (less than zero) in the case of an error. Unless you have used the AFD_SHOW_ERROR option to ask the DLL to present any error to the user, you should display an error for the user before aborting the lookup. The AFDErrorText function will help you obtain a string which can be useful for displaying to the user to describe the error.

Otherwise the account number is valid and you should use the SortCode, AccountNumber and TypeOfAccount fields returned in the supplied type or structure instance to process the account number (and optionally roll number with some building societies) as they may be updated should account number translation have been necessary.

If the return value is AFD_SUCCESS then the account number has been validated, if the return value is AFD_SUCCESS_NO_VALIDATION then account numbers on this sortcode cannot be validated and so the number should still be treated as valid. This return code is provided so you can carry out an additional check on the account number, e.g. asking a customer on the phone to repeat it, checking it has been entered from a paper form correctly etc. if you wish to do so.

If you are processing account numbers on both clearing systems and wish to check which one the branch at which the account number that was entered resides on, you can do this by checking the value of the ClearingSystem field:

Clearing System Field Value	**Meaning**
United Kingdom (BACS)	The branch at which this account is held is on the UK clearing system
Ireland (IPSO)	The branch at which this account is held is on the Irish Payment Services Organisation Clearing System
Both UK and Irish	The branch at which this account is held is on both the UK and Irish clearing systems. The actual account may only clear through one of these systems but it is not possible to determine which one so you should clarify that with the customer.

Should you also wish to check the branch details match those that the customer has supplied, check the transaction types allowed at this branch, or obtain the address to use for this branch (may not be the branch physical location) then you can carry out a lookup for the sortcode (see Section 4.1) to obtain the branch information.

Example VB code to validate an account number:

```
  Dim details As AFDBankData
  Dim retVal As Long

  ' Set the Sort Code and Account Number
  details.SortCode = txtValidateSortcode.Text ' Change txtValidateSortCode to your
sortcode entry textbox
```

```
  details.AccountNumber = txtValidateAccountNo.Text ' Change txtValidateAccountNo to
your account number entry textbox

  ' Carry out the validation (you can change the AFD_BOTH_CLEARINGS option to
AFD_UK_CLEARING or AFD_IRISH_CLEARING as desired)
  retVal = AFDData(afdBankFieldSpec, AFD_ACCOUNT_VALIDATE + AFD_BOTH_CLEARINGS,
details)

  ' Abort with Message if error
  If retVal < 0 Then
    MsgBox AFDErrorText(retVal)
    Exit Sub
  End If

  ' Display validation result - with details to submit for payment - note non-standard
account number's will be translated
  MsgBox "Account Number Valid: " + vbCrLf + vbCrLf + "Sortcode: " +
Trim(details.SortCode) + vbCrLf + "Account Number: " + Trim(details.AccountNumber) +
vbCrLf + "Type of Account Code: " + Trim(details.TypeOfAccount) + vbCrLf + "Clearing
System: " + Trim(details.ClearingSystem)
```

Example C++ code to validate an account number (Visual C++):

```
  HINSTANCE afdDLL = (HINSTANCE)NULL;
  AFDDATA afdData = (AFDDATA)NULL;
  afdBankData details;
  char msgTxt[255];
  long retVal;
  // Load DLL
  if (!afdInitDLL(&afdDLL, &afdData)) {
    MessageBox("Error Loading afddata.dll", "Error", 0);
    return;
  }

  // Update Data so we can read the sortcode and account number variables
  UpdateData(TRUE);

  // Set the Sort Code and Account Number
  strcpy(details.SortCode, m_txtValidateSortcode); // Change this to your sort code
textbox value variable
  strcpy(details.AccountNumber, m_txtValidateAccountNo); // Change this to your
account number textbox value variable

  // Carry out the validation (you can change the AFD_BOTH_CLEARINGS option to
AFD_UK_CLEARING or AFD_IRISH_CLEARING as desired)
  retVal = (afdData)(afdBankFieldSpec, AFD_ACCOUNT_VALIDATE + AFD_BOTH_CLEARINGS,
(char*)&details);

  // Abort with Message if error or user cancelled
  if (retVal < 0) {
    AFDErrorText(retVal, msgTxt);
    MessageBox(msgTxt, "Error", 0);
    return;
  }

  // Display validation result - with details to submit for payment - note non-
standard account number's will be translated
  strcpy(msgTxt, "Account Number Valid:\n\nSortcode: ");
  strcat(msgTxt, details.SortCode);
  strcat(msgTxt, "\nAccount Number: ");
  strcat(msgTxt, details.AccountNumber);
  strcat(msgTxt, "\nType of Account Code: ");
  strcat(msgTxt, details.TypeOfAccount);
  strcat(msgTxt, "\nClearing System: ");
  strcat(msgTxt, details.ClearingSystem);
  MessageBox(msgTxt, "Validation Successful", 0);

  // free DLL instance
  FreeLibrary(afdDLL);
  afdDLL = (HINSTANCE)NULL;
```

3.6. Card Number Validation – BankFinder Only

This function provides the ability to validate a card number, and optionally check that an expiry date indicates that the card is in-date. This checks that the card number is a valid one for the type of card and can indicate the card type. This does not guarantee that the card exists or that a transaction will be authorized, but greatly cuts down on errors due to incorrectly entered numbers.

To carry out a validation, you will first need to declare an instance of the AFDBankData structure you have declared in your general declarations module or class (see Section 4.1). You should then set the CardNumber and, if you wish, the ExpiryDate Fields for the card that you wish to validate.

You then call the AFDData function with the following three parameters:

1. The Field Specification String (as detailed in Section 4.1)
2. The operation code (AFD_CARD_VALIDATE constant)
3. The instance of the structure or type that you declared.

If you would prefer the DLL to display any error message that may occur to the user, rather than having to display this yourself, you should add the AFD_SHOW_ERROR constant to the operation parameter. This is only suitable for desktop applications as it displays any error message on-screen.

The AFDData function will return a negative value (less than zero) in the case of an error. Unless you have used the AFD_SHOW_ERROR option to ask the DLL to present any error to the user, you should display an error for the user before aborting the lookup. The AFDErrorText function will help you obtain a string which can be useful for displaying to the user to describe the error.

Otherwise the card number is valid. If you wish to determine the card type, the CardType field will hold this information.

Example VB code to validate a card number:

```
  Dim details As AFDBankData
  Dim retVal As Long

  ' Set the Card Number and Expiry Date (Optional)
  details.CardNumber = txtValidateCardNo.Text ' Change txtValidateCardNo to your card
number entry textbox
  details.ExpiryDate = txtValidateExpiry.Text ' Change txtValidateExpiry to your
expiry date entry textbox

  ' Carry out the validation (you can change the AFD_BOTH_CLEARINGS option to
AFD_UK_CLEARING or AFD_IRISH_CLEARING as desired)
  retVal = AFDData(afdBankFieldSpec, AFD_CARD_VALIDATE, details)

  ' Abort with Message if error
  If retVal < 0 Then
    MsgBox AFDErrorText(retVal)
    Exit Sub
  End If

  ' Display validation result
  MsgBox "Card Valid: " + Trim(details.CardType)
```

Example C++ code to validate a card number (Visual C++):

```
  HINSTANCE afdDLL = (HINSTANCE)NULL;
  AFDDATA afdData = (AFDDATA)NULL;
  afdBankData details;
  char msgTxt[255];
  long retVal;
  // Load DLL
  if (!afdInitDLL(&afdDLL, &afdData)) {
    MessageBox("Error Loading afddata.dll", "Error", 0);
    return;
  }

  // Update Data so we can read the card number and expiry date variables
  UpdateData(TRUE);

  // Set the Card Number and Expiry date (Optional)
  strcpy(details.CardNumber, m_txtValidateCardNo); // Change this to your card number
textbox value variable
  strcpy(details.ExpiryDate, m_txtValidateExpiry); // Change this to your expiry date
textbox value variable

  // Carry out the validation (no need to alter the line below)
  retVal = (afdData)(afdBankFieldSpec, AFD_CARD_VALIDATE, (char*)&details);

  // Abort with Message if error or user cancelled
  if (retVal < 0) {
    AFDErrorText(retVal, msgTxt);
    MessageBox(msgTxt, "Error", 0);
    return;
  }

  // Display validation result
  strcpy(msgTxt, "Card Valid: ");
  strcat(msgTxt, details.CardType);
  MessageBox(msgTxt, "Validation Successful", 0);

  // free DLL instance
  FreeLibrary(afdDLL);
  afdDLL = (HINSTANCE)NULL;
```

3.7. List Functions – Address Management Only

With Postcode Plus, Names & Numbers and TraceMaster products you can use the list functions to obtain a list of alias localities for the postcode sector that a postcode or result is contained in. These are non-postally required localities held by Royal Mail which can or may be included on an address if desired. An example of this would be including Wimbledon for an address in London. You should note that these are stored at postal sector level (e.g. SW19 1) and there are often multiple entries for an address so a locality being returned does not mean it is necessarily the best one for the particular address you are viewing.

For Names & Numbers and TraceMaster products only it is also possible to obtain a list of possible values for most fields, e.g. all the Mailsort codes present, business descriptions, etc. You can also specify the start value of the field, e.g. return all surnames present starting with “Smith”.

When using International data you can also use the List functions to obtain a list of all available countries (names or ISO codes).

To carry out a list operation, you first need to declare an instance of the AFDListData structure you have declared in your general declarations module

or class (see Section 4.1). For an alias locality lookup, you should then set the Lookup field to the postcode or record key that you wish to lookup the alias localities for. When retrieving field lists from Names & Numbers you can set this to specify that only entries starting with your specified string are returned (this is essential for long lists like surname to be useful, but is generally not so useful with shorter lists like Household Composition).

The operation parameter passed to the AFDData function determines the List function carried out:

Constant	**Description**
AFD_LIST_ALIAS_LOCALITY	Returns all alias localities for the sector that the specified postcode or key resides in.
The following are applicable when using International data only:	
AFD_LIST_COUNTRY_ISO	Will return the ISO codes of all available countries.
AFD_LIST_COUNTRY	Will return the names of all available countries.
The following are applicable to Names & Numbers and TraceMaster Products Only: These all return a list of all entries of the data item specified in the data Setting the lookup parameter will restrict matches to only those items starting with the specified string.	
AFD_LIST_FORENAME	Returns Forenames (first names).
AFD_LIST_SURNAME	Returns Surnames
AFD_LIST_ORGANISATION	Returns Organisations
AFD_LIST_PROPERTY	Returns Properties
AFD_LIST_STREET	Returns Streets
AFD_LIST_LOCALITY	Returns Localities
AFD_LIST_TOWN	Returns Postal Towns
AFD_LIST_COUNTY	Returns Counties (This includes Postal, Traditional and Administrative County names)
AFD_LIST_MAILSORT_CODE	Returns Mailsort codes
AFD_LIST_URBAN_RURAL_CODE	Returns Urban Rural Codes
AFD_LIST_URBAN_RURAL_NAME	Returns Urban Rural Names
AFD_LIST_WARD_CODE	Returns Ward Codes
AFD_LIST_WARD_NAME	Returns Ward Names
AFD_LIST_CONSTITUENCY	Returns Constituencies
AFD_LIST_EER_CODE	Returns EER Codes (European Electoral Region Codes)
AFD_LIST_EER_NAME	Returns EER Names
AFD_LIST_AUTHORITY_CODE	Returns Local / Unitary Authority Codes
AFD_LIST_AUTHORITY	Returns Authority Names
AFD_LIST_LEA_CODE	Returns LEA Codes (Local Education Authority)
AFD_LIST_LEA_NAME	Returns LEA Names
AFD_LIST_TV_REGION	Returns TV Regions
AFD_LIST_NHS_CODE	Returns NHS Codes
AFD_LIST_NHS_NAME	Returns NHS Names
AFD_LIST_NHS_REGION_CODE	Returns NHS Region Codes
AFD_LIST_NHS_REGION_NAME	Returns NHS Region Names
AFD_LIST_PCT_CODE	Return PCT Codes
AFD_LIST_PCT_NAME	Return PCT Names
AFD_LIST_CENSATION_CODE	Returns Censation Codes
AFD_LIST_AFFLUENCE	Returns Censation Affluence Codes with descriptions

AFD_LIST_LIFESTAGE	Returns Censation Lifestage Codes with descriptions
AFD_LIST_ADDITIONAL_CENSUS_INFO	Returns Censation Additional Information with descriptions.
AFD_LIST_HOUSEHOLD_COMPOSITION	Returns Household composition codes with descriptions.
AFD_LIST_BUSINESS	Returns Business descriptions
AFD_LIST_SIZE	Returns Company Size catagories
AFD_LIST_SIC_CODE	Returns SIC Codes
AFD_LIST_COUNCIL_TAX_BAND	Returns Council Tax Bands

You then call the AFDData function with the following three parameters:

1. The List Field Specification String (as detailed in Section 4.1)
2. The operation code (See above for options).
3. The instance of the structure or type that you declared.

If you would prefer not to use your own list box in your application, you may wish to add to the operation constant the AFD_LIST_BOX option. This causes the DLL to display a list box for you returning the record that the user selects, rather than returning all matching list records to your application. This is only suitable for desktop applications as it displays the list box on-screen. Similarly adding AFD_SHOW_ERROR causes the DLL to display any error message to the user itself.

The AFDData function will return AFD_SUCCESS for most operations or AFD_NO_RESULTS_FOUND if there were no matching list items. Other errors may be returned if the product is not correctly licensed (i.e. AFD_ERROR_OPENING_FILES, AFD_ERROR_FILE_READ, or AFD_DATA_LICENSE_ERROR). So, unless you have used the AFD_SHOW_ERROR option to ask the DLL to present any error to the user, you may wish to call AFDErrorText in these circumstances to obtain a string to display to the user describing the error.

If the return value from the AFDData function is AFD_SUCCESS then a matching result has been returned and you can access the fields in the structure or type instance supplied to obtain full details for it. The resulting string will be found in the List Field of the structure.

If you have specified the AFD_LIST_BOX option then the user will have selected the required item and you can access the selected result in the List Field of the structure.

Otherwise, you will have retrieved the first record. To retrieve the rest of the records you should call the AFDData function as above repeatedly with the same operation code as before, but adding the AFD_GET_NEXT constant to it to obtain subsequent records. These can be added to a list box as above or processed as required.

Example VB code for a List operation to retrieve alias localities:

```
  Dim details As AFDListData
  Dim retVal As Long
```

```
  Static running As Boolean

  ' Prevent corruption of list box from button being clicked twice
  If running Then Exit Sub
  running = True

  ' Replace lstResult with the name of your list box for the results
  With lstResult

  ' Clear out any existing items in the list
  .Clear

  ' Reset Cancel flag
  cancelFlag = False

  ' Set the lookup
  details.Lookup = txtLookup.Text ' Change txtLookup to the postcode or record key you
wish to lookup

  ' Carry out the lookup (Can alter the operation to retrieve N&N list items if
desired)
  retVal = AFDData(afdFieldSpec, AFD_LIST_ALIAS_LOCALITY, details)

  ' Abort with Message if error or user cancelled
  If retVal < 0 Then
    MsgBox AFDErrorText(retVal)
    running = False
    Exit Sub
  End If

 ' Now add matching records to the list box
  Do While retVal >= 0
    ' Add the item to the list box with hidden key at the end
    .AddItem Trim(details.List)
    ' Give user the chance to cancel and allow list box to update
    DoEvents
    ' Check if user cancelled
    If cancelFlag Then
      MsgBox "Lookup Cancelled"
      running = False
      Exit Sub
    End If
    retVal = AFDData(afdFieldSpec, AFD_GET_NEXT + AFD_LIST_ALIAS_LOCALITY, details)
  Loop

  ' Check results have been returned
  If .ListCount = 0 Then
    MsgBox "No Results Found"
  Else
    .ListIndex = 0 ' Select First item in the list
  End If

  End With

  running = False
```

Example C++ Code For an Address Management Lookup (Visual C++)

```
  HINSTANCE afdDLL = (HINSTANCE)NULL;
  AFDDATA afdData = (AFDDATA)NULL;
  static bool running = false;
  afdListData details;
  char listItem[2055];
  char msgTxt[255];
  long retVal;
  CListBox* listBox;
  MSG msg;

  // Check if we are already running to prevent crossing over items in the listbox
  if (running) return;
  running = true;

  // Load DLL
  if (!afdInitDLL(&afdDLL, &afdData)) {
    MessageBox("Error Loading afddata.dll", "Error", 0);
    return;
```

```
  }

  // Replace m_lstResult with the name given to a variable assigned to your list box
control for the results
  listBox = &m_lstResult;

  // Clear out any existing items in the list
  listBox->ResetContent();

  // Reset Cancel flag
  cancelFlag = false;

  // Update Data so we can read the lookup variable
  UpdateData(TRUE);

  // Set the lookup
  strcpy(details.Lookup, m_txtLookup); // Change this to the postcode or record key
you wish to lookup

  // Carry out the lookup (Can alter the operation to retrieve N&N list items if
desired)
  retVal = (afdData)(afdFieldSpec, AFD_LIST_ALIAS_LOCALITY, (char*)&details);

  // Abort with Message if error or user cancelled
  if (retVal < 0) {
    AFDErrorText(retVal, msgTxt);
    MessageBox(msgTxt, "Error", 0);
    running = false;
    return;
  }

  // Now add matching records to the list box
  while (retVal >= 0) {
    // make up list item
    strncpy(listItem, details.List, sizeof(details.List));
    listItem[sizeof(details.List)] = '\0';
    // Add the item to the list box
    listBox->AddString(listItem);
    // Give user the chance to cancel and allow list box to update
    if(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }
    // Check if user cancelled
    if (cancelFlag) {
      MessageBox("Search Cancelled", "Cancelled", 0);
      return;
    }
    retVal = (afdData)(afdFieldSpec, AFD_GET_NEXT + AFD_LIST_ALIAS_LOCALITY,
(char*)&details);
  }

  // Check results have been returned
  if (listBox->GetCount() == 0)
    MessageBox("No Results Found", "Error", 0);
  else {
    listBox->SetCurSel(0); // Select First item in the list

    OnSelchangeLstResult(); // Set this to your list change method to simulate
selecting the first list item

  }

  // free DLL instance
  FreeLibrary(afdDLL);
  afdDLL = (HINSTANCE)NULL;

  running = false;
```

3.8. String Utility Functions – Depreciated and Unsupported

These are provided for compatibility with existing applications which may depend on them but for new developments we would recommend you use in-

built functions which are included with most modern development environments.

To carry out a string operation, you will first need to declare an instance of the AFDStringData structure you have declared in your general declarations module or class (see Section 4.1). You should then set the Lookup Field to the string that you wish to clean. If you wish to carry out a Search and Replace operation then you should also set the Search and Replace fields to the appropriate strings.

The operation parameter passed to the AFDData function determines the String operation which is carried out, and this should be one of the following:

Constant	Description
AFD_ STRING_SEARCH_REPLACE	All occurrences in the string specified in the Lookup field of the string specified in the Search field are replaced with the string in the Replace field.
AFD_STRING_SEARCH_REPLACE_CASE	This is the same as AFD_STRING_SEARCH_REPLACE but is case sensitive.
AFD_STRING_CAPITALISE	This corrects the capitalisation of the string specified in the Lookup field. For example 'commercial STREET' would become 'Commercial Street'.
AFD_STRING_CLEAN_LINE	This cleans the string specified in the Lookup field by removing spurious characters that should not be in an address line, e.g. a trailing comma.
AFD_STRING_CHECK_POSTCODE	This checks if the string specified in the Lookup field looks like a postcode.
AFD_STRING_CLEAN_POSTCODE	This cleans the postcode specified in the Lookup field to tidy up the postcode specified.
AFD_STRING_ABBREVIATE_COUNTY	This provides the Royal Mail Approved county abbreviation for the county specified in the Lookup field if one exists.

You then call the AFDData function with the following three parameters:

4. The String Field Specification String (as detailed in Section 4.1)
5. The operation code (See above for options).
6. The instance of the structure or type that you declared.

The AFDData function will return AFD_SUCCESS for most operations. If you are using AFD_STRING_CLEAN_POSTCODE then AFD_NO_RESULTS_FOUND will be returned if the string does not look like a postcode. For AFD_STRING_ABBREVIATE_COUNTY the constant AFD_NO_RESULTS_FOUND will also be returned if there is no Royal Mail approved abbreviation available for the specified county name.

The resulting string will be found in the Lookup Field of the structure. When using the AFD_STRING_CLEAN_POSTCODE function the Outcode and Incode portions of the postcode (portion before and after the space) will also be avaliable in the separate Outcode and Incode Fields.

Example VB code for a Search/Replace String Operation:

```
  Dim details As AFDStringData
  Dim retVal as Long

  ' Set the Lookup, Search and Replace parameters
  details.Lookup = txtLookup.Text ' Change txtLookup.Text to your string entry textbox
  details.Search = txtSearch.Text ' Change txtSearch.Text to your search entry textbox
  details.Replace = txtReplace.Text ' Change txtReplace.Text to your replace textbox

  ' Carry out the String operation
  retVal = AFDData(afdStringFieldSpec, AFD_STRING_SEARCH_REPLACE, details)

  ' Check if success
  If retVal >= 0 Then
    ' details.Lookup holds the updated string
  End If
```

Example C++ code for a Search/Replace String Operation (Visual C++):

```
  HINSTANCE afdDLL = (HINSTANCE)NULL;
  AFDDATA afdData = (AFDDATA)NULL;
  afdStringData details;
  long retVal;
  // Load DLL
  if (!afdInitDLL(&afdDLL, &afdData)) {
    MessageBox("Error Loading afddata.dll", "Error", 0);
    return;
  }

  // Update Data so we can read the lookup, search and replace variables
  UpdateData(TRUE);

  // Set the String to lookup, and the string to Search for and Replace with
  strcpy(details.Lookup, m_txtLookup); // Change this to your string textbox value
variable
  strcpy(details.Search, m_txtSearch); // Change this to your search textbox value
variable
  strcpy(details.Replace, m_txtReplace); // Change this to your replace textbox value
variable

  // Carry out the String operation
  retVal = (afdData)(afdStringFieldSpec, AFD_STRING_SEARCH_REPLACE, (char*)&details);

  // Check if success
  if (retVal >= 0) {
    // details.Lookup holds the updated string
  }

  // free DLL instance
  FreeLibrary(afdDLL);
  afdDLL = (HINSTANCE)NULL;
```

3.9. Grid Utility Functions – UK Address Management Only

These functions are used to carry out operations related to grid references and latitude and longitude values. You can convert between GB and Irish based grid references and also convert to and from latitude and longitude values. The facility to convert a value in kilometers to miles and vice-versa, return an approximate grid reference for a location and also calculate the distance between two geographical locations is also included.

To carry out a grid operation, you will first need to declare an instance of the AFDGridData structure you have declared in your general declarations module or class (see Section 4.1).

The operation parameter passed to the AFDData function determines the String operation which is carried out, and this should be one of the following:

Constant	Value	Description
AFD_GRID_CONVERT	512	Converts a GB or NI based grid reference, or latitude and longitude value to all other grid reference types and latitude and longitude values. You should set the location in the Fields of your structure or type instance, for example set the GBGridE and GBGridN fields and the function will return the NIGridE and NIGridN variants along with the latitude and longitude values etc. (This uses a 1m resolution (6 digit). Using a constant of 0 rather than 512 uses 5 digit grids).
AFD_GRID_LOOKUP_LOCATION	513	Looks up a town, locality, or partial postcode specified in the Lookup field and provides an approximate grid reference and latitude and longitude values for the location if a match is found. Can return multiple records if the location is ambiguous. (This uses a 1m resolution (6 digit). Using a constant of 1 rather than 513 uses 5 digit grids).
AFD_GRID_DISTANCE	514	Calculates the distance between a pair of grid references or latitude and longitude values specified. You will need to set a grid or latitude and longitude value in both the normal fields and those prefixed with “From” to find the distance in both Miles and Km. (This uses a 1m resolution (6 digit). Using a constant of 2 rather than 514 uses 5 digit grids).

You then call the AFDData function with the following three parameters:

1. The Grid Field Specification String (as detailed in Section 4.1)
2. The operation code (See above for options).
3. The instance of the structure or type that you declared.

The AFDData function will return AFD_SUCCESS on success. If the operation fails, for example a location looked up does not exist or a grid reference specified is out of range then AFD_NO_RESULTS_FOUND will be returned.

You can then read the resulting grid reference, latitude and longitude values, or Km and Miles values as appropriate for the operation you have carried out and the data that you require.

Example VB code for converting a GB based grid reference:

```
  Dim details As AFDGridData
  Dim retVal as Long

  ' Set the GBGridE and GBGridN parameters
  details.GBGridE = "406600" ' Change 406600 to the grid easting value you wish to
convert
  details.GBGridN = "286500" ' Change 286500 to the grid northing value you wish to
convert

  ' Carry out the Grid operation
```

```
  retVal = AFDData(afdGridFieldSpec, AFD_GRID_CONVERT, details)

  ' Check if success
  If retVal >= 0 Then
    ' Other elements of details hold converted values, e.g. Latitude and Longitude
  End If
```

Example C++ code for converting a GB based grid reference (Visual C++):

```
  HINSTANCE afdDLL = (HINSTANCE)NULL;
  AFDDATA afdData = (AFDDATA)NULL;
  afdGridData details;
  long retVal;
  // Load DLL
  if (!afdInitDLL(&afdDLL, &afdData)) {
    MessageBox("Error Loading afddata.dll", "Error", 0);
    return;
  }

  // Set the GBGridE and GBGridN parameters
  strcpy(details.GBGridE, "406600"); // Change 406600 to the grid easting value you
wish to convert
  strcpy(details.GBGridN, "286500"); // Change 286500 to the grid northing value you
wish to convert

  // Carry out the Grid operation
  retVal = (afdData)(afdGridFieldSpec, AFD_GRID_CONVERT, (char*)&details);

  // Check if success
  if (retVal >= 0) {
    // Other elements of details hold converted values, e.g. Latitude and Longitude
  }

  // free DLL instance
  FreeLibrary(afdDLL);
  afdDLL = (HINSTANCE)NULL;
```

3.10. Email Utility Function

This function is used to carry out validation of an email address. This verifies that the address is in the correct format for an email address and also that the domain exists to help minimise errors in data entry.

To carry out an email operation, you will first need to declare an instance of the AFDEmailData structure you have declared in your general declarations module or class (see Section 4.1).

The operation parameter passed to the AFDData function determines the level of validation which is carried out, and this should be one of the following:

Constant	Value	Description
AFD_EMAIL_FULL	0	Full email validation including live domain lookup
AFD_EMAIL_FORMAT	2	Validate email addres format is correct only
AFD_EMAIL_TLD	3	Validate email format is correct and the top level domain exists
AFD_EMAIL_LOCAL	4	Validate email format, top level domain and for well known domains carry out additional checks of the local portion of the address

You then call the AFDData function with the following three parameters:

4. The Email Field Specification String (as detailed in Section 4.1)
5. The operation code (See above for options).

6. The instance of the structure or type that you declared.

The AFDData function will return AFD_SUCCESS on success. If the operation fails, for example the email address format is not valid then AFD_NO_RESULTS_FOUND will be returned.

Example VB code for validating an email address:

```
  Dim details As AFDEmailData
  Dim retVal as Long

  ' Set the Email parameter
  details.Email = "support@afd.co.uk" ' Change support@afd.co.uk to the email address
you wish to validate

  ' Carry out the Email operation
  retVal = AFDData(afdEmailFieldSpec, AFD_EMAIL_FULL, details)

  ' Check if success
  If retVal >= 0 Then
    ' Email address is valid
  End If
```

Example C++ code for validating an email address (Visual C++):

```
  HINSTANCE afdDLL = (HINSTANCE)NULL;
  AFDDATA afdData = (AFDDATA)NULL;
  afdEmailData details;
  long retVal;
  // Load DLL
  if (!afdInitDLL(&afdDLL, &afdData)) {
    MessageBox("Error Loading afddata.dll", "Error", 0);
    return;
  }

  // Set the GBGridE and GBGridN parameters
  strcpy(details.Email, "support@afd.co.uk"); // Change support@afd.co.uk to the email
address you wish to validate

  // Carry out the Email operation
  retVal = (afdData)(afdEmailFieldSpec, AFD_GRID_CONVERT, (char*)&details);

  // Check if success
  if (retVal >= 0) {
    // Email Address is Valid
  }

  // free DLL instance
  FreeLibrary(afdDLL);
  afdDLL = (HINSTANCE)NULL;
```

3.11. Clean Function – UK Address Management Only

Requires a Refiner API License

The clean function allows an address, for example from a database, to be cleaned, i.e. where possible matched to Postcode Plus and therefore given a correct deliverable address.

To clean an address will first need to declare an instance of the AFD structure you have declared in your general declarations module or class

You will then need to set address fields in your structure to specifiy the address to be cleaned. These do not need to match up to the actual fields, for example if you have Address Line 1, Address Line 2, Address Line 3 and

Postcode in your database you could set these to Property, Street, Locality and Postcode fields in the structure and they will be cleaned and returned in the correct named fields when matched. Note that if you set any non-address fields they will be ignored (Please see Appendix G for the list of fields that Refiner will use).

You then call the AFDData function with the following three parameters:

4. The Field Specification String (as detailed in Section 4.1)
5. The operation code (AFD_CLEAN constant)
6. The instance of the structure or type that you declared.

The AFDData function will return a negative value (less than zero) in the case where an address cannot be fully matched. This could be because the address was unmatchable, International, or an ambiguous result was found (see Section 4.1.9 for details of these return codes). An address will still be returned as this will include the address with Field Placement correction which you can use if you desire.

Where the function returns a positive value (greater than zero) this means that the address has been uniquely matched. You may still like to examine the return value as this will give details as to the level to which the address was matched (see Section 4.1.9 for details of these return codes). Many other fields are also avaliable with additional (non-address data) which you may require.

In the case of an ambiguous or suggested result (return code is -102, -103, or -104) the first address returned from the function will be the original address with field placement. For non-batch processes you may wish to present a list of addresses for the user to choose from and in this case you can continue to call the AFDData function as above repeatedly with the same operation code as before, but adding the AFD_GET_NEXT constant to it to obtain subsequent records. These can be added to a list box as above or processed as required. You should call AFDData in a lookup to retrieve these records allowing the user to cancel the lookup should it take some time or they realise they have entered something incorrectly.

Example VB code to clean an Address:

```
  Dim details As AFDAddressData
  Dim retVal as Long

  ' Replace lstResult with the name of your list box if you wish to display ambiguous
results
  With lstResult

  ' Clear out any existing items in the list
  .Clear

  ' Clear Structure
  ClearAFDAddressData details

  ' Set the fields to specify the address that you wish to clean
  details.Organisation = txtSearchOrganisation.Text
  details.Property = txtSearchProperty.Text
  details.Street = txtSearchStreet.Text
  details.Locality = txtSearchLocality.Text
```

```
  details.Town = txtSearchTown.Text
  details.Postcode = txtSearchPostcode.Text

  ' Clean the Address
  retVal = AFDData(afdFieldSpec, AFD_CLEAN, details)

  ' Show the resulting address
  ' These are any of the members of the details. type (Use Trim to remove whitespace)
  txtName.Text = Trim(details.Name)
  txtOrganisation.Text = Trim(details.Organisation)
  txtProperty.Text = Trim(details.Property)
  txtStreet.Text = Trim(details.Street)
  txtLocality.Text = Trim(details.Locality)
  txtTown.Text = Trim(details.Town)
  txtPostcode.Text = Trim(details.Postcode)

  ' Show Cleaning Status
  Msgbox AFDRefinerCleaningText(retVal)

  ' If ambiguous then add matching records to the list box for user selection
  '  - This is optional and not normally useful for batch processes
  If retVal = AFD_REFINER_AMBIGUOUS_POSTCODE Or retVal = AFD_REFINER_AMBIGUOUS_MATCH
Or retVal = AFD_REFINER_SUGGEST_RECORD Then
    Do While retVal <> AFD_ERROR_END_OF_SEARCH
      ' Add the item to the list box with hidden key at the end
      .AddItem details.List + details.Key
      retVal = AFDData(afdFieldSpec, AFD_GET_NEXT  + AFD_CLEAN, details)
    Loop

  End If

  End With
```

Example C++ Code to clean an address (Visual C++)

```
  HINSTANCE afdDLL = (HINSTANCE)NULL;
  AFDDATA afdData = (AFDDATA)NULL;
  afdAddressData details;
  char listItem[2055];
  char msgTxt[255];
  long retVal;
  CListBox* listBox;

  // Load DLL
  if (!afdInitDLL(&afdDLL, &afdData)) {
    MessageBox("Error Loading afddata.dll", "Error", 0);
    return;
  }

  // Replace lstResult with the name of your list box if you wish to display ambiguous
results
  listBox = &m_lstResult;

  // Clear out any existing items in the list
  listBox->ResetContent();

  // Update Data so we can read the search variables
  UpdateData(TRUE);

  // Set the fields to specify the address that you wish to clean
  strcpy(details.Organisation, m_txtSearchOrganisation);
  strcpy(details.Property, m_txtSearchProperty);
  strcpy(details.Street, m_txtSearchStreet);
  strcpy(details.Locality, m_txtSearchLocality);
  strcpy(details.Town, m_txtSearchTown);
  strcpy(details.Postcode, m_txtSearchPostcode);

  // Clean the Address
  retVal = (afdData)(afdFieldSpec, AFD_CLEAN, (char*)&details);

  // Show the resulting address
  // These are any of the members of the details. structure
  m_txtName = details.Name;
  m_txtOrganisation = details.Organisation;
  m_txtProperty = details.Property;
  m_txtStreet = details.Street;
```

```
  m_txtLocality = details.Locality;
  m_txtTown = details.Town;
  m_txtPostcode = details.Postcode;

  // Update Fields
  UpdateData(FALSE);

  // Show Cleaning Status
  AFDRefinerCleaningText(retVal, msgTxt);
  MessageBox(msgTxt, "Cleaning Status", 0);

  // If ambigious then add matching records to the list box for user selection
  //  - This is optional and not normally useful for batch processes
  if ((retVal == AFD_REFINER_AMBIGUOUS_POSTCODE) || (retVal ==
AFD_REFINER_AMBIGUOUS_MATCH) || (retVal == AFD_REFINER_SUGGEST_RECORD)) {
    while (retVal != AFD_ERROR_END_OF_SEARCH) {
      // make up list item with hidden key at the end
      strncpy(listItem, details.List, sizeof(details.List));
      strncpy(listItem + sizeof(details.List), details.Key, sizeof(details.Key));
      listItem[sizeof(details.List) + sizeof(details.Key)] = '\0';
      // Add the item to the list box
      listBox->AddString(listItem);
      retVal = (afdData)(afdFieldSpec, AFD_GET_NEXT + AFD_CLEAN, (char*)&details);
    }
  }

  // free DLL instance
  FreeLibrary(afdDLL);
  afdDLL = (HINSTANCE)NULL;
```

4. Other Features

4.1. Selecting TraceMaster Datasets

AFD Names & Numbers TraceMaster includes historic datasets for Address Management Data going back to 1998. These provide previous year's electoral rolls and business data. Lookup and Search operations function as described in sections 4.2 and 4.3 of this documentation function with TraceMaster in the same way as with all other products.

By default these operations will operate using the Current (latest) dataset. However to use a historic dataset simply specify the dataset name (year) in the DataSet Field of the AFDData structure. The product will automatically carry out your Lookup or Search operation using the specified dataset.

To retrieve records from all datasets, you can call the AFDData function in a loop specifying each dataset in turn.

4.2. Determining the Product in Use

When integrating with Address Management products the same code will work with any of our Address Management products (AFD Postcode, AFD Postcode Plotter, AFD Postcode Plus, AFD Names & Numbers and AFD Names & Numbers TraceMaster).

It is not normally necessary to determine which product has been used as you can integrate with one, e.g. Names & Numbers and the user can use any of our address management products – they will just have less data returned depending on the product they have. However, if for any reason, such as disabling/enabling features of your product - you can use the Product field if you wish to determine which product the user has and that has been used by the Common API.

Note that in the case of multiple address management products being installed the AFD Common API will use the highest level product available. For example, AFD Names & Numbers would be used in preference to AFD Postcode.

The Product field will contain one of the following values depending on the product being used:

- AFD Postcode
- AFD Postcode Plotter
- AFD Postcode Plus
- AFD Names & Numbers
- AFD Names & Numbers TraceMaster

Note that when carrying out a BankFinder operation AFD BankFinder will always be the product name returned.

4.3. Using Welsh Data in Postcode Plus

Welsh data is available for Postcode Plus on request. It works alongside the existing English language PAF data and provides Welsh language equivalents for streets, localities and towns in Wales were such equivalents are available.

To obtain address details using the Welsh variant simply set the DataSet property of the address structure to “Welsh” prior to making your call to the API. Any operation including lookup’s, searches and retrieving records can be done using either dataset. Note that the data returned when using either dataset will be the same if no Welsh language alternative is available.

You can also retrieve the same record in both Welsh and English simply by calling the API to retrieve the record once with the DataSet property set to an empty string (or English if you prefer) and once set to “Welsh”. For example, if you carry out a lookup for a postcode, as specified in Section 4.2 of this documentation, and add the items to a list box, when the user selects an item from the list you can retrieve the same address in both Welsh and English language variants by using the List Fetch operation described in Section 4.4 twice for the same record, once with the DataSet parameter set to Welsh and once with it not set.

Example VB code to fetch an item selected in the list in both English and Welsh:

```
  Dim details As AFDAddressData
  Dim welshDetails As AFDAddressData
  Dim pos As Long, retVal As Long

  ' Replace lstResult with the name of your list box for the results
  With lstResult

  ' Check a valid item is selected
  If .ListIndex = -1 Then
    MsgBox "No Item Selected"
    Exit Sub
  End If

  ' Set DLL parameters to retrieve the selected record
  details.Key = Mid(lstResult, 513) ' Replace lstResult with the name of your list box
for the results

  ' We will want the same record in Welsh too
  welshDetails.Key = details.Key

  ' Finished with the list box
  End With

  ' Set DataSet to Welsh for welshDetails
  details.DataSet = ""
  welshDetails.DataSet = "Welsh"

  ' Carry out the lookup for English language data and then Welsh language data
  retVal = AFDData(afdFieldSpec, AFD_RETRIEVE_RECORD, details)
  retVal = AFDData(afdFieldSpec, AFD_RETRIEVE_RECORD, welshDetails)

  ' Abort with Message if error
  If retVal < 0 Then
    MsgBox AFDErrorText(retVal)
    Exit Sub
  End If
```

```
  ' Now Assign required fields to your application
  ' These are any of the members of the details. type (Use Trim to remove whitespace)
  txtEnglishName.Text = Trim(details.PostalCounty)
  txtEnglishOrganisation.Text = Trim(details.AbbreviatedPostalCounty)
  txtEnglishProperty.Text = Trim(details.OptionalCounty)
  txtEnglishStreet.Text = Trim(details.AbbreviatedOptionalCounty)
  txtEnglishLocality.Text = Trim(details.TraditionalCounty)
  txtEnglishTown.Text = Trim(details.AdministrativeCounty)
  txtEnglishPostcode.Text = Trim(details.Postcode)
  txtWelshName.Text = Trim(welshDetails.PostalCounty)
  txtWelshOrganisation.Text = Trim(welshDetails.AbbreviatedPostalCounty)
  txtWelshProperty.Text = Trim(welshDetails.OptionalCounty)
  txtWelshStreet.Text = Trim(welshDetails.AbbreviatedOptionalCounty)
  txtWelshLocality.Text = Trim(welshDetails.TraditionalCounty)
  txtWelshTown.Text = Trim(welshDetails.AdministrativeCounty)
  txtWelshPostcode.Text = Trim(welshDetails.Postcode)
```

Example C++ code to fetch an item selected in the list for Address Management products (Visual C++):

```
  HINSTANCE afdDLL = (HINSTANCE)NULL;
  AFDDATA afdData = (AFDDATA)NULL;
  afdAddressData details;
  afdAddressData welshDetails;
  bool foundSel = false;
  long retVal;
  CListBox* listBox;
  char lstStr[2055];
  char msgTxt[255];

  // Load DLL
  if (!afdInitDLL(&afdDLL, &afdData)) {
    MessageBox("Error Loading afddata.dll", "Error", 0);
    return;
  }

  // Replace m_lstResult with the name given to a variable assigned to your list box
control for the results
  listBox = &m_lstResult;

  // Set DLL parameters to retrieve the selected record
  listBox->GetText(listBox->GetCurSel(), lstStr);
  strncpy(details.Key, lstStr + sizeof(details.List), sizeof(details.Key));

  // We will want the same record in Welsh too
  strncpy(welshDetails.Key, details.Key, sizeof(details.Key));

  ' Set DataSet to Welsh for welshDetails
  strcpy(details.DataSet, "");
  strcpy(welshDetails.DataSet, "Welsh");

  // Carry out the lookup for English language data and then Welsh language data
  retVal = (afdData)(afdFieldSpec, AFD_RETRIEVE_RECORD, (char*)&details);
  retVal = (afdData)(afdFieldSpec, AFD_RETRIEVE_RECORD, (char*)&welshDetails);

  // Abort with Message if error
  if (retVal < 0) {
    AFDErrorText(retVal, msgTxt);
    MessageBox(msgTxt, "Error", 0);
    return;
  }

  // Now Assign required fields to your application
  // These are any of the members of the details. structure
  m_txtEnglishName = details.Name;
  m_txtEnglishOrganisation = details.Organisation;
  m_txtEnglishProperty = details.Property;
  m_txtEnglishStreet = details.Street;
  m_txtEnglishLocality = details.Locality;
  m_txtEnglishTown = details.Town;
  m_txtEnglishPostcode = details.Postcode;
  m_txtWelshName = welshDetails.Name;
```

```
m_txtWelshOrganisation = welshDetails.Organisation;
m_txtWelshProperty = welshDetails.Property;
m_txtWelshStreet = welshDetails.Street;
m_txtWelshLocality = welshDetails.Locality;
m_txtWelshTown = welshDetails.Town;
m_txtWelshPostcode = welshDetails.Postcode;

// Update Fields
UpdateData(FALSE);

// free DLL instance
FreeLibrary(afdDLL);
afdDLL = (HINSTANCE)NULL;
```

4.4. DX Member Data

DX Members can have access to DX data from within Postcode Plus and the Common API. This enables you to lookup and search for DX addresses just as you can do with Royal Mail postal addresses. Uniquely, the Common API also allows you to easily identify DX addresses associated with a PAF address to route your mail through a DX member's box wherever possible resulting in savings over Royal Mail.

If you run the Wizard to generate a code sample with DX data installed declarations will be included for the DXNumber (10 characters), DXExchange (30 characters) and DXProfession (30 Characters). You can also manually add these to your field specification string and structure. Postcode Everywhere users will automatically have these fields returned in the XML if they have the DX data installed.

Fast-find functionality works with DX data as well as postal data. For example, as well as looking up a postcode you can also carry out a fast-find for a DX number and searching for an organisaiton name with fast-find will search both postal and DX data. This allows you to easily combine your lookup's. When searching you can either search the standard postal fields or specify the DX Number, organisation, exchange or profession to search theDX data instead. (If you only want to specify one set of search fields in your application then placing DX followed by the DX number in the normal street field will work too – town can then be used to specify the exchange if desired).

When results are returned following any lookup or search if the address is also a DX Member the DXNumber, DXExchange and DXProfession fields will also be returned to indicate this. You can format a DX address as follows for printing:

<Organisation>	e.g.	Pannone LLP
DX <DXNumber>		DX 14314
<DXExchange>		MANCHESTER

See Appendix K for a current list of available DX Professions and Exchanges.

5. Appendices

Appendix A. Address Management Product Fields

This currently includes Postcode, Plotter, Postcode Plus, Names & Numbers (including TraceMaster), and ZipAddress.

■ Field returned by this product and fully searchable
☐ Field returned by this product, but not searchable.

Note: The API Wizard will add one to the default size for development environments that normally use null terminated strings, e.g. C++ and C# to accommodate the null terminator.

Also note that the alternative address formats provided do share some of the same fields where there data is identical, but you should not mix and match other fields between the different formats as this could lead to address corruption. For example with Standard Address Fields the Street or Locality field could include a street number, whereas with Raw PAF Fields the number would be in the separate Number field.

Field Name	Default Size	Description	Postcode	Plotter	Postcode Plus	Names & Numbers	ZipAddress
General Fields							
Lookup	255	Specify postcode (or zipcode) and fast-find lookup string's here for lookup operations.	■	■	■	■	■
Key	255	Provides a key which can be used to easily retrieve the record again, e.g. when a user clicks on an item in the list box.	■	■	■	■	■
List	512	Provides a list item formatted to be added to a list box for this record.	☐	☐	☐	☐	☐
Product	40	Indicates the product name used [10]	☐	☐	☐	☐	☐
Occupant Fields							
Name	120	Full name (includes title, first name, middle initial and surname).				☐	
Gender	6	The gender (M or F) of the resident if known.				■	
Forename	30	The first name of the resident				■	
MiddleInitial	6	The middle initiate of the resident.				■	
Surname	30	The surname/last name of the resident.				■	
OnEditedRoll	6	Indicates if the resident is on the edited electoral roll (i.e. they have not opted out). Set to Y if the are on the Edited Roll, N if not, blank for Organisation and				■	

		other records). To search set to #Y to return only records on the electoral roll, #N only for those not on the electoral roll or !N for all records including Organisations but excluding those not on the Edited Roll.					
DateOfBirth	10	The residents date of birth if known (electoral roll attainers in the last 10 years only).				■	
Residency	6	Gives time in years that the occupant has lived at this address.				■	
HouseholdComposition	106	Describes the household composition of the selected address [6]				■	
Standard Address Fields (Formatted as an address would appear on an envelope)							
Organisation	120	Full business name (includes any department)			■	■	■
Property	120	Property (building-includes any sub-building).			■	■	■
Street	120	Delivery Street (includes any sub-street)	■	■	■	■	■
Locality	70	Locality (sometimes a village name – in ZipAddress used for Urbanization)	■	■	■	■	■
Town	30	Postal Delivery Town (or City)	■	■	■	■	■
Postcode	10	The Royal Mail Postcode for this address (or ZipCode)	■	■	■	■	■
Raw PAF Fields (Formatted closer to how they appear on Raw PAF, useful if your database stores fields this way)							
OrganisationName	60	Business Name			■	■	■
Department	60	Department Name			☐	■	
Sub Building	60	Sub Building Name			■	■	
Building	60	Building Name			■	■	■
Number	10	House Number			■	■	■
DependentThoroughfare	60	Sub-Street Name	■	■	■	■	
Thoroughfare	60	Street Name	■	■	■	■	■
DoubleDependentLocality	35	Sub-Locality Name	■	■	■	■	
DependentLocality	35	Locality Name (Urbanization in ZipAddress)	■	■	■	■	■
Town	30	Postal Delivery Town (City)	■	■	■	■	■
Postcode	10	The Royal Mail Postcode for this address (or Zipcode)	■	■	■	■	■
BS7666 Fields (Fields to help provide addresses which conform to BS 7666-5:2006)							
Identifier	8	Provides a unique identifier for the address (the Royal Mail UDPRN)			■	■	
BuildDate	10	Provides the build date, which can be used as the start date, entry date, and update date fields for BS7666.			☐	☐	☐
Administrator	20	Provides the administrator of the gazetteer (AFD).			☐	☐	☐
Language	5	Provides the language (ENG)			☐	☐	☐
Department	60	The name of a department within an organization where required.			■	■	
Organization	60	The Organization Name			■	■	■
SubUnit	60	Sub-Unit of a building where needed			■	■	
BuildingName	60	Building Name where present			■	■	■
BuildingNumber	10	Building Number, including 17A, 17-19, etc.			■	■	■

SubStreet	60	Sub-Street where needed			■	■	
DeliveryStreet	60	Designated Street Name	■	■	■	■	■
SubLocality	60	Sub-Locality where required	■	■	■	■	
DeliveryLocality	60	Locality name (or Urbanization)	■	■	■	■	■
DeliveryTown	30	Postal Town name (or City)	■	■	■	■	■
Code	10	The Postcode (or ZipCode)	■	■	■	■	■
County Fields *(Counties are Optional for addressing and AFD provide different types of county to meet your needs – all supply State Abbreviation in ZipAddress)*							
Postal County	30	Royal Mail supplied postal county	■	■	■	■	■
AbbreviatedPostalCount y	30	Royal Mail approved abbreviation is used where available for the postal county	■	■	■	■	■
OptionalCounty	30	Postal counties including optional ones for most addresses which would otherwise not have a county name.	■	■	■	■	■
AbbreviatedOptionalCou nty	30	Royal Mail approved abbreviation is used where available for the optional county	■	■	■	■	■
TraditionalCounty	30	The traditional county name for this postcode	■	■	☐	■	■
AdministrativeCounty	30	The administrative county name for this postcode	■	■	☐	■	■
Alternative Postcode Fields (Can be used in-place of the Postcode field to provide it as separate parts)							
Outcode	4	The Outcode portion of the Postcode (the portion before the space)	■	■	■	■	
Incode	3	The Incode portion of the Postcode (the portion after the space).	■	■	■	■	
Additional Postal Data Fields							
DPS	2	The Delivery Point Suffix which along with the postcode uniquely identifies the letterbox.			☐	■	
PostcodeFrom	8	Used with Postcode field to provide a range for searching. Also returns any changed postcode from a lookup.	☐	☐	■	■	
PostcodeType	6	L for Large User Postcode, S for Small User.	☐	☐	☐	■	
MailsortCode	5	Used for obtaining bulk mail discounts.	☐	☐	☐	■	
UDPRN	8	Royal Mail Unique Delivery Point Reference Number assigned to this letter box.			■	■	
JustBuilt	10	AFDJustBuilt - Contains the date of inclusion on PAF for properties thought to be recently built. The date is stored numerically in descending format in the form YYYYMMDD. YYYY is the year, MM is the month and DD is the day. For example 20080304 is 04/03/2008.			■	■	
USA Format Address Fields							
Recipient	120	The Recipient name were held (usually Organisation).			■	■	■
Secondary	120	Secondary address details (usually a building name or apartment)			■	■	■
Primary	120	Primary address details (usually a street	■	■	■	■	■

		number and name)					
Urbanization	60	Urbanization (applies only to Puerto Rico – returns Locality for UK addresses)	■	■	■	■	■
City	30	Official City name for the address (Town for UK addresses)	■	■	■	■	■
State	30	State Abbreviation (e.g. WA, returns Postal County for UK addresses)	■	■	■	■	■
ZipCode	20	Full Zip+4 Code for this address (Postcode for UK addresses)	■	■	■	■	■
International Address Fields							
Country	30	Specifies the name of the country to search for when using International data and returns the name of the country that the result returned was for.	■	■	■	■	■
CountryISO	3	Specifies the ISO code of the country to search for when using International data and returns the code for the country the result returned was for.	■	■	■	■	■
Address1…7	120	These fields provide a formatted address ready to print on an address label and so eliminate the need to format the address afterwards (as the rules differ from country to country).	■	■	■	■	■
Principality	60	This is the principality for the address if applicable.	■	■	■	■	
Region	60	This is the region for the address if applicable.	■	■	■	■	
Cedex	60	This specifies the Cedex if applicable to the address.	■	■	■	■	
Phone Number Related Fields							
Phone	20	STD Code or Phone Number	■ [3]	■ [3]	■ [3]	■	
Geographical Fields							
GridE	10	Grid Easting as a 6 digit reference		☐	■	■	
GridN	10	Grid Northing as a 6/7 digit reference		☐	■	■	
Latitude	10	Latitude representation of Grid Reference in Decimal Format (WGS84)		☐	■	■	
GBGridE	10	UK Based Grid Easting as a 6 digit reference. Always returns the UK based grid even for Northern Ireland addresses.		☐	■	■	
GBGridN	10	UK Based Grid Northing as a 6/7 digit reference.		☐	■	■	
NIGridE	10	Irish Grid Based Grid Easting as a 6 digit reference. Always returns the Irish base grid even for mainland UK addresses.		☐	■	■	
NIGridN	10	Irish Grid Based Grid Northing as a 6/7 digit reference.		☐	■	■	
Longitude	10	Longitude representation of Grid Reference in Decimal Format (WGS84)		☐	■	■	
Miles	6	Distance from supplied grid reference			■	■	
Km	6	Distance from supplied grid reference			■	■	
UrbanRuralCode	2	Provides a code which indicates if an area is mainly urban or rural and how			☐	■	

		sparsely populated those area's are. [11]					
UrbanRuralName	60	Provides a description which goes along with the UrbanRuralCode.			☐	■	
SOALower	9	Lower level Super Output Area (Data Zone in Scotland, Super Output Area in Northern Ireland)			☐	■	
SOAMiddle	9	Middle level Super Output Area (Intermediate Geography in Scotland, not applicable for Northern Ireland).			☐	■	
SubCountryName	20	Provides the devolved or non-UK country name (e.g. England, Scotland, Wales etc.)			☐	■	
Administrative / Electoral Division Fields							
WardCode	9	Code identifying the electoral ward for this postcode			☐	■	
WardName	50	Name identifying the electoral ward for this postcode			☐	■	
AuthorityCode	9	Local/Unitary Authority for this Postcode (same as the start of the ward code).			☐	■	
Authority	50	Local / Unitary Authority for this postcode			☐	■	
ConstituencyCode	9	Parliamentary Constituency Code for this postcode			☐	■	
Constituency	50	Parliamentary Constituency for this postcode			☐	■	
DevolvedConstituencyCode	9	Devolved Constituency Code for this postcode (currently covers Scotland)			☐	■	
DevolvedConstituencyName	50	Devolved Constituency Name for this postcode (currently covers Scotland)			☐	■	
EERCode	9	Code identifying the European Electoral Region for this postcode			☐	■	
EERName	40	Name identifying the European Electoral Region for this postcode			☐	■	
LEACode	3	Code identifying the Local Education Authority for this postcode			☐	■	
LEAName	50	Name identifying the Local Education Authority for this postcode			☐	■	
TVRegion	30	ISBA TV Region (not TV Company)			☐	■	
Postcode Level Property Indicator Fields							
Occupancy	6	Indication of the type of occupants of properties found on the selected postcode [4]	☐	☐	☐	■	
OccupancyDescription	30	Description matching the Occupancy [4]	☐	☐	☐	☐	
AddressType	6	Indication of the type of property level data to capture to have the full address for a property on the selected postcode. [5]	☐	☐	☐	■	
AddressTypeDescription	55	Description matching the Address Type [5]	☐	☐	☐	☐	
NHS Fields							
NHSCode	6	National Health Service Area Code			☐	■	
NHSName	50	National Health Service Area Name			☐	■	

PCTCode	9	National Heath Service Clinical Commisioning Group Code for England (Local Health Board Code in Wales, Community Health Partnership in Scotland, Local Commissioning Group in Northern Ireland, Primary Healthcare Directorate in the Isle of Man)			☐	■	
PCTName	50	Name matching the PCT Code field			☐	■	
Censation Data Fields (See Main product Manual for full details of Censation Codes and there meaning).							
CensationCode	10	Censation Code assigned to this Postcode	☐	☐	☐	■	
CensationLabel	50	Provides a handle for the Censation Code	☐	☐	☐	■	
Affluence	30	Affluence description	☐	☐	☐	■	
Lifestage	100	LifeStage description	☐	☐	☐	■	
AdditionalCensusInfo	200	Additional information from the Census.	☐	☐	☐	■	
Additional Organisation Information Fields							
Business	100	Provides a description of the type of business				■	
SICCode	10	Standard Industry Classification Code for an organisation record.				■	
Size	6	Gives an indication of the number of employees of an organisation at this particular office. [7]				■	
LocationType	6	The type of Business Location, e.g. Head Office or Branch Office				■	
BranchCount	6	The number of branches for this business				■	
GroupID	6	An ID of the Group were a business is part of a wider group				■	
ModelledTurnover	15	The modelled annual turnover for the business				■	
NationalSize	6	Gives an indication of the number of employees of an organisation covering all sites. [7]				■	
Alias Localities (Non-postally required Localities)							
AliasLocalities	4	Returns the number of alias records present for the postcode sector in which this result resides.			☐	☐	
AliasLocality	35	Returns an alias (non-postal) locality that resides in the postcode sector that this address is contained in. Note that many postcode sectors have multiple alias localities and as such you can include this field multiple times to return multiple localities.			☐	☐	
USA Specific Fields							
RecordType	30	Returns a description for the type of address record returned. [12]					☐
CarrierRouteID	4	Required for bulk mailings					☐
LACSStatus	2	Indicates if the address is available on the LACSLink system for obtaining new addresses.					☐
FinanceNumber	7	The USPS Finance Number for this					☐

		location					
CongressionalDistrict	3	The congressional district of this address					
CountyNumber	4	The USPS assigned number for this county					☐
CountyName	26	The name of the county for this address					☐
CityAbbreviation	14	Provides a postally acceptable abbreviation for long city names.					☐
Advanced / Premium Fields							
DataSet	10	With Postcode Plus and Welsh data can be set to 'Welsh" to obtain the Welsh language version of an address in Wales where available. If not set then the English language version will be returned. With TraceMaster this indicates an historic dataset to use [9]				■	
CouncilTaxBand	6	Provides the Council Tax Band for the selected property. *Requires Names & Numbers*				■	

Notes:

[3] STD Code Only – No Phone Number present

[4] Possible Occupancy values and descriptions are as follows (information in brackets not part of the description):

1. Large User Organisation (Single Organisation on this postcode)
2. Small User Organisation (All the properties on this postcode are likely to be businesses)
3. Mostly Organisations (Most of the properties on this postcode are organisations)
4. Mixed (This postcode contains a mixture of business and residential addresses)
5. Mostly Residential (Most of the properties on this postcode are residential)
6. Residential (All the properties on this postcode are likely to be residential)

[5] Possible Address Type values and descriptions are as follows (information in brackets not part of the description):

1. Numbered (Only a property number needs to be captured)
2. Numbered and Named (This postcode contains a mixture of properties needing a property number and those needing a property name including properties such as 16b)
3. Numbered and Named, Likelihood of Multiple Occupancy (This postcode contains a mixture of properties needing a property number and those needing a property name. Some of the properties on this postcode are likely to contain multiple occupants, e.g. flats).
4. Named (This postcode only contains properties needing a property name).
5. Non-Standard Address Format (This refers to addresses which do not have a street field at all, or have multiple street names on the same postcode. This also includes addresses with numbered localities (no street but a house number which goes in with the locality field). It is in-effect a warning to be careful in capturing the property information as it is not in one of the most common address formats).
6. PO Box (This postcode has a PO Box number)
7. No Property Information (Addresses on this postcode have no property information - i.e. capture an Organisation or Resident name only)

[6] The household composition field includes both a number and description and can have any of the following values.

1. 1 Male and 1 Female occupant with different surnames
2. 1 Male and 1 Female occupant with the same surname (married couples)
3. Mixed household
4. More than 2 persons with the same surname (e.g. older families).
5. 1 Male Occupant Only
6. 1 Female Occupant Only
7. More than 7 persons (e.g. old peoples home).

[7] The Size property can have any of the following values:

A. 1 to 9 employees
B. 10 to 19 employees
C. 20 to 49 employees
D. 50 to 99 employees
E. 100 to 199 employees
F. 200 to 499 employees
G. 500 to 999 employees
H. 1000+

(If blank then this is unknown or not applicable).

[8] The phone match type will be set to F if the phone number has been matched to the full name of this resident, or S if just to the surname. This can be useful for identifying the bill payer among multiple residents.

[9] DataSet property when used with the Names & Numbers TraceMaster product can currently be any of the following years: 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005 or Current (for the current data). Only one year can be specified at a time and searches/lookup's will fail if the specified year has not been installed. New years are automatically accessible when they become available if installed with no change required to the DLL or your application.

[10] The Product field can have any of the following values:
AFD Postcode
AFD Postcode Plotter
AFD Postcode Plus
AFD Names & Numbers
AFD Names & Numbers TraceMaster

[11] The Urban Rural Code differs from England and Wales, Scotland and Northern Ireland. The possible codes and there meanings are as follows:

England & Wales
1. Urban (Sparse): Falls within Urban settlements with a population of 10,000 or more and the wider surrounding area is sparsely populated
2. Town and Fringe (Sparse): Falls within the Small Town and Fringe areas category and the wider surrounding area is sparsely populated.
3. Village (Sparse): Falls within the Village category and the wider surrounding area is sparsely populated.
4. Hamlet and Isolated Dwelling (Sparse): Falls within the Hamlet and Isolated Dwelling category and thee wider surrounding area is sparsely populated.
5. Urban (Less Sparse): Falls within urban settlements with a population of 10,000 or more and the wider surrounding area is less sparsely populated.
6. Town and Fringe (Less Sparse): Falls within the Small Town and Fringe areas category and the wider surrounding area is less sparsely populated.
7. Village (Less Sparse): Falls within the village category and the wider surrounding area is less sparsely populated.
8. Hamlet and Isolated Dwelling (Less Sparse): Falls within the Hamlet & Isolated Dwelling category and the wider surrounding area is less sparsely populated

Scotland
S1. Large Urban Area: Settlement of over 125,000 people.

S2. Other Urban Area: Settlement of 10,000 to 125,000 people.
S3. Accessible Small Town: Settlement of 3,000 to 10,000 people, within 30 minutes drive of a settlement of 10,000 or more.
S4. Remote Small Town: Settlement of 3,000 to 10,000 people, with a drive time of 30 to 60 minutes to a settlement of 10,000 or more.
S5. Very Remote Small Town: Settlement of 3,000 to 10,000 people, with a drive time of over 60 minutes to a settlement of 10,000 or more.
S6. Accessible Rural: Settlement of less than 3,000 people, within 30 minutes drive of a settlement of 10,000 or more.
S7. Remote Rural: Settlement of less than 3,000 people, with a drive time of 30 to 60 minutes to a settlement of 10,000 or more.
S8. Very Remote Rural: Settlement of less than 3,000 people, with a drive time of over 60 minutes to a settlement of 10,000 or more.

Northern Ireland
A - E (Urban):
A. Belfast Metropolitan Urban Area
B. Derry Urban Area
C. Large Town: 18,000 and under 75,000 people
D. Medium Town: 10,000 and under 18,000 people
E. Small Town: 4,500 and under 10,000 people
F - H (Rural):
F. Intermediate Settlement: 2,250 and under 4,500 people
G. Village: 1,000 and under 2,250 people
H. Small Village, Hamlet or Open Countryside: Less than 1,000 people

[12] The record type will be one of the following:

General Delivery
Highrise
Firm
Street
PO Box
Rural Route/Highway Contract
Multi-Carrier Route

Appendix B. BankFinder Fields

■ Field returned by this product and fully searchable
☐ Field returned by this product, but searchable only through the Search Text field only.

Note: The API Wizard will add one to the default size for development environments that normally use null terminated strings, e.g. C++ and C# to accommodate the null terminator.

Also note that the alternative address formats provided do share some of the same fields where there data is identical, but you should not mix and match other fields between the different formats as this could lead to address corruption. For example with Standard Address Fields the Street or Locality field could include a street number, whereas with Raw PAF Fields the number would be in the separate Number field.

Field Name	**Default Size**	**Description**	**BankFinder**
General Fields			
Lookup	255	Specify sort code, postcode and fast-find lookup string's here for lookup operations.	■
ClearingSystem	25	Clearing system for this record [3]	☐
Key	40	Provides a key which can be used to easily retrieve the record again, e.g. when a user clicks on an item in the list box.	■
List	512	Provides a list item formatted to be added to a list box for this record.	☐
Product	40	Provides the product name used [10]	☐
SearchText	255	Specify text to search for within any of the BankFinder fields	■
General Bank Fields			
SortCode	6	Bank's Sortcode	■
BankBIC	8	Bank BIC Code [1]	■
BranchBIC	3	Branch BIC Code [1]	■
SubBranchSuffix	2	Allows a branch to be uniquely identified where there is a cluster of branches sharing the same Sort Code [1]	☐
ShortBranchTitle	27	The official title of the branch	■
FullBranchTitle	105	Extended title for the institution	■
CentralBankCountryCode	2	The ISO Country code for beneficiary banks in other countries	☐
CentralBankCountryName	20	The country name corresponding to the ISO code given.	☐
SupervisoryBodyCode	1	Indicates the supervisory body for an institution that is an agency in any of the clearings. [2]	☐
SupervisoryBodyName	50	The name of the supervisory body [2]	☐
DeletedDate	10	Specifies the date the branch was closed if it is not active	☐

BranchType	20	The branch type - Main Branch, Sub or NAB Branch, Linked Branch	☐
MainBranchSortCode	6	Set for linked branches in a cluster. It identifies the main branch for the cluster. For IPSO records this is set to the branch that would handle transactions for this sortcode when the branch has been amalgamated with another.	☐
Location	60	Where present helps indicate the physical location of the branch.	■
BranchName	35	Defines the actual name or place of the branch	■
AlternativeBranchName	35	An alternative name or place for the branch where applicable.	■
OwnerBankShortName	20	Short version of the name of the Owning Bank	■
OwnerBankFullName	70	Full version of the name of the Owning Bank	■
OwnerBankCode	4	The four digit bank code of the Owning Bank [1]	☐
Standard Address Fields *(Formatted as an address would appear on an envelope)*			
Organisation	120	Owner Bank Full Name	■
Property	65	Bank Postal Address: Property (Building)	☐
Street	60	Bank Postal Address: Street	☐
Locality	60	Bank Postal Address: Locality	☐
Town	30	Bank Postal Address: Town	■
County	30	Bank Postal Address: County (Optional)	☐
Postcode	8	The Royal Mail Postcode for this address	■
Raw PAF Fields *(Formatted closer to how they appear on Raw PAF, useful if your database stores fields this way)*			
OrganisationName	60	Owner Bank Full Name	■
SubBuilding	60	Bank Postal Address: Sub-Building Name	☐
Building	60	Bank Postal Address: Building Name	☐
Number	10	Bank Postal Address: House Number	☐
DependentThoroughfare	60	Bank Postal Address: Sub-Street Name	☐
Thoroughfare	60	Bank Postal Address: Street Name	☐
Double DependentLocality	35	Bank Postal Address: Sub-Locality Name	☐
DependentLocality	35	Bank Postal Address: Locality Name	☐
Town	30	Bank Postal Address: Postal Delivery Town	■
County	30	Bank Postal Address: County (Optional)	☐
Postcode	8	The Royal Mail Postcode for this address	■
Alternative Postcode Fields *(Can be used in-place of the Postcode field to provide it as separate parts)*			
Outcode	4	The portion of the postcode before the space	■
Incode	3	The portion of the postcode after the space	■
Phone Number Related Fields			
Phone	20	Phone Number for this bank	■
Fax	20	Fax Number for this bank (IPSO only)	☐
BACS Related Fields *(Not applicable to IPSO Records)*			
BACSStatus	5	Indicates the BACS Clearing Status [4]	☐
BACSStatusDescription	60	Provides a description for the status [4]	☐
BACSLastChange	10	Date on which BACS data was last amended	☐
BACSClosedClearing	10	Indicates the date the branch is closed in BACS clearing if applicable.	☐
BACSRedirectedFromFlag	1	Set to R if other branches are redirected to this sort code.	☐
BACSRedirectedToSortCode	6	This specifies the sort code to which BACS should redirect payments addressed to this sort code if	☐

		applicable.	
BACSSettlementBankCode	4	BACS Bank Code of the bank that will settle payments for this branch.	☐
BACSSettlementBankShortName	20	Short form name of the settlement bank	☐
BACSSettlementBankFullName	70	Full form name of the settlement bank	☐
BACSSettlementBankSection	2	Numeric data required for BACS to perform it's settlement.	☐
BACSSettlementBankSubSection	2	Numeric data required for BACS to perform it's settlement.	☐
BACSHandlingBankCode	4	BACS Bank Code of the member that will take BACS output from this branch.	☐
BACSHandlingBankShortName	20	Short form name of the handling bank	☐
BACSHandlingBankFullName	70	Full form name of the handling bank	☐
BACSHandlingBankStream	2	Numeric code defining the stream of output within the Handling Bank that will be used or payments to this branch.	☐
BACSAccountNumbered	1	Set to 1 if numbered bank accounts are used	☐
BACSDDIVoucher	1	Set to 1 if Paper Vouchers have to be printed for Direct Debit Instructions.	☐
BACSDirectDebits	1	Set to 1 if branch accepts Direct Debits	☐
BACSBankGiroCredits	1	Set to 1 if branch accepts Bank Giro Credits	☐
BACSBuildingSocietyCredits	1	Set to 1 if branch accepts Building Society Credits.	☐
BACSDividendInterestPayments	1	Set to 1 if branch accepts Dividend Interest Payments.	☐
BACSDirectDebitInstructions	1	Set to 1 if branch accepts Direct Debit Instructions.	☐
BACSUnpaidChequeClaims	1	Set to 1 if branch accepts Unpaid Cheque Claims.	☐
CHAPS Related Fields *(Not applicable to IPSO Records)*			
CHAPSPStatus	1	Indicates the CHAPS Sterling clearing Status [5]	☐
CHAPSPStatusDescription	80	Provides a description for the status [5]	☐
CHAPSPLastChange	10	Date on which CHAPS Sterling data was last amended	☐
CHAPSPClosedClearing	10	Indicates the date the branch is closed in CHAPS Sterling clearing if applicable.	☐
CHAPSPSettlementBankCode	3	CHAPS ID of the bank that will settle payments for this branch,	☐
CHAPSPSettlementBankShortName	20	Short form of the name of the settlement bank	☐
CHAPSPSettlementBankFullName	70	Full form of the name of the settlement bank	☐
CHAPSEStatus	1	Indicates the CHAPS Euro clearing Status [6]	☐
CHAPSEStatusDescription	80	Provides a description for the status [6]	☐
CHAPSELastChange	10	Date on which CHAPS Euro data was last amended	☐
CHAPSEClosedClearing	10	Indicates the date the branch is closed in CHAPS Euro clearing if applicable.	☐
CHAPSEEuroRoutingBICBank	8	Specifies the SWIFT closed user group Bank BIC to which CHAPS Euro payments for this branch should be routed.	☐
CHAPSEEuroRoutingBICBranch	3	Specifies the SWIFT closed user group Branch BIC to which CHAPS Euro payments for this branch should be routed.	☐
CHAPSESettlementBankCode	3	CHAPS ID of the bank that will settle payments for this branch.	☐
CHAPSESettlementBankShortName	20	Short form of the name of the settlement bank	☐
CHAPSESettlementBankFullName	70	Full form of the name of the settlement bank	☐
CHAPSEReturnIndicator	1	Set to R if this is the branch to which CHAPS Euro	☐

		payments should be sent.	
C&CCC Related Fields *(Not applicable to IPSO Records)*			
CCCCStatus	1	Indicates the C&CCC clearing Status [7]	☐
CCCCStatusDescription	40	Provides a description for the status [7]	☐
CCCCLastChange	6	Date on which C&CCC data was last amended	☐
CCCCClosedClearing	30	Indicates the date the branch is closed in C&CCC clearing if applicable.	☐
CCCCSettlementBankCode	3	BACS generated code of the bank that will settle payments for this branch.	☐
CCCCSettlement BankShortName	20	Short form of the name of the settlement bank	☐
CCCCSettlement BankFullName	70	Full form of the name of the settlement bank	☐
CCCCDebitAgencySortCode	50	When the Status field is set to 'D' this specifies where cheque clearing is handled for this branch.	☐
CCCCReturnIndicator	6	Set if this is the branch that other banks should return paper to. It will only be set for a sort code of a Member.	☐
Validation Related Fields			
AccountNumber	12	The account number to validate (set along with the sort code field for account number validation).	■
TypeOfAccount	1	The type of account field required for transmitting data to BACS when the account number has been translated.	☐
RollNumber	20	For some building society credit accounts a roll number is required. This can be specified here for validation.	■
IBAN	50	The International Bank Account Number. This contains the sort code and account number in a standardised format for cross-border transactions.	■
BuildingSocietyName	70	For building society accounts requiring a roll number this will contain the name of the recieving building society as this sometimes differs from the bank branch that the payment passes through.	☐
CardNumber	20	Used to specify a card number to validate	■
ExpiryDate		Optional field to specify an expiry date to validate along with the card number.	■
CardType	30	Indicates the card type following validation [8]	☐

Notes:

[1] Does not apply to records in the IPSO (Irish Payment Services Organisation) clearing system.

[2] The supervisory body code and name can be any of the following:
A. Bank of England
B. Building Society Commission
C. Jersey, Guernsey or Isle of Man authorities
D. Other

[3] The clearing system property can have one of the following values

United Kingdom (BACS) – For branch records for the UK clearing system
Ireland (IPSO) – For branch records on the Irish Payment Services Organisation Clearing System
Both UK and Irish – Returned by Account Number Validation only when a branch is on both systems.

Note, that you should only accept account numbers validated on the Irish system if you can clear through both the Irish (IPSO) system as well as the UK (BACS) system.

[4] Possible values for the BACS Status and Description fields are as follows:

M. Branch of a BACS Member
A. Branch of an Agency Bank
I. Member of the Irish Clearing Services (IPSO)
Does not accept BACS Payments

[5] Possible values for the CHAPS Sterling Status and Description fields are as follows:

M. Direct Branch of a CHAPS £ Member that Accepts CHAPS £ Payments
A. Branch of an Agency Bank that Accepts CHAPS £ Payments
I. Indirect Branch of a Member or Agency Bank that Accepts CHAPS £ Payments
Does not accept CHAPS £ Payments

[6] Possible values for the CHAPS Euro Status and Description fields are as follows:

D. Direct Branch of a CHAPS € Member that Accepts CHAPS € Payments
I. Indirect Branch of a Member or Agency Bank that Accepts CHAPS € Payments
Does not accept CHAPS € Payments

[7] Possible values for the C&CCC Status and Description fields are as follows:

M. Branch of a C&CCC Member
F. Full Agency Bank Branch
D. Debit Agency Branch Only
Not Part of the C&CCC Clearing

[8] Possible values for the card type field are as follows:

MasterCard
Visa
American Express
Visa Debit
Electron
Visa Purchasing
UK Maestro
International Maestro
Solo and Maestro
JCB
Charities Aid Foundation
MasterCard Debit

[10] The Product field would have the value ‘AFD BankFinder’.

Appendix C. Nearest Fields

■ Field returned by this product and can be used in a Lookup
☐ Field returned by this product, but not searchable.

Field Name	Default Size	Description	Nearest
General Fields			
Lookup	255	Used to specify the string to process	■
GBGridE	10	Provides the Grid Easting value for the nearest record or a Grid Easting to lookup if no lookup string is supplied.	■
GBGridN	10	Provides the Grid Northing value for the nearest record or a Grid Easting to lookup if no lookup string is supplied.	■
NIGridE	10	Provides the Grid Easting value on the Irish Grid System.	■
NIGridN	10	Provides the Grid Northing value on the Irish Grid System.	■
Latitude	10	Latitude representation of Grid Reference in Decimal Format (WGS84)	■
Longitude	10	Longitude representation of Grid Reference in Decimal Format (WGS84)	■
TextualLatitude	15	A textual representation of the Latitude field	☐
TextualLongitude	15	A textual representation of the Longitude field	☐
Km	10	Distance of this record from supplied grid reference in kilometres, or the maximum distance to return records for.	■
Miles	10	Distance of this record from supplied grid reference in miles, or the maximum distance to return record for.	■
List	512	Provides a list item formatted to be added to a list box for this record.	☐
Key	40	Provides a key which can be used to easily retrieve the record again, e.g. when a user clicks on an item in the list box.	☐
Product	40	Indicates the product name used	☐
MaxRecords	5	Specifies the maximum number of records to return.	☐

In addition to these all the fields contained in the database table that you are using with Nearest are also returned and are fully searchable when using the Search operation.

Appendix D. List Fields

■ Field returned by this product and fully searchable
☐ Field returned by this product, but not searchable.

All List Operations:

Field Name	Default Size	Description	String
General Fields			
Lookup	255	For an alias locality lookup this specifies the postcode or record key to find the alias localities for. With Names & Numbers Field Lists this specifies that only those entries starting with this string should be returned.	■
List	255	Returns each list entry	☐
Product	40	Indicates the product name used	☐

Appendix E. Utility Fields

■ Field returned by this product and fully searchable
□ Field returned by this product, but not searchable.

Grid Utility:

Field Name	Default Size	Description	String
General Fields			
Lookup	255	Used to specify the string to process	■
GBGridE	10	Grid Easting Reference - GB Grid	■
GBGridN	10	Grid Northing Reference - GB Grid	■
NIGridE	10	Grid Easting Reference - Irish Grid	■
NIGridN	10	Grid Northing Reference - Irish Grid	■
Latitude	10	Latitude representation of Grid Reference in Decimal Format (WGS84)	■
Longitude	10	Longitude representation of Grid Reference in Decimal Format (WGS84)	■
TextualLatitude	15	Latitude representation of Grid Reference in Direction, Degrees, Minutes and Seconds	■
TextualLongitude	15	Latitude representation of Grid Reference in Direction, Degrees, Minutes and Seconds	■
Km	6	Specifies the string to search for	■
Miles	6	Specifies the string to replace occurrences of Search with.	■
GBGridEFrom	10	Grid Easting Reference - GB Grid – used to specify two points for a distance calculation	■
GBGridNFrom	10	Grid Northing Reference - GB Grid – used to specify two points for a distance calculation	■
NIGridEFrom	10	Grid Easting Reference - Irish Grid – used to specify two points for a distance calculation	■
NIGridNFrom	10	Grid Northing Ref. - Irish Grid – used to specify two points for a distance calculation	■
LatitudeFrom	10	Latitude representation of Grid Reference in Decimal Format (WGS84) – used to specify two points for a distance calculation	■
LongitudeFrom	10	Longitude representation of Grid Reference in Decimal Format (WGS84) – used to specify two points for a distance calculation	■
TextualLatitudeFrom	15	Latitude representation of Grid Reference in Direction, Degrees, Minutes and Seconds – used to specify two points for a distance calculation	■
TextualLongitudeFrom	15	Latitude representation of Grid Reference in Direction, Degrees, Minutes and Seconds – used to specify two points for a distance calculation	■

Email Utility:

Field Name	Default Size	Description	Email
Email Field			

Email	255	The Email Address to validate	■

String Utility – Depreciated and Unsupported:

Field Name	Default Size	Description	String
General Fields			
Lookup	255	Used to specify the string to process	■
Postcode	10	Provides the postcode output where applicable	☐
Outcode	4	Provides the outcode portion of the postcode where applicable.	☐
Incode	3	Provides the incode portion of the postcode where applicable.	☐
County	30	Provides the abbreviated county name where applicable.	☐
Search	255	Specifies the string to search for	☐
Replace	255	Specifies the string to replace occurrences of Search with.	☐

Appendix F. Refiner Address Fields

When cleaning an address, you can use any of the address fields in the structure to specify the address. These do not need to match up to the actual fields, for example if you have Address Line 1, Address Line 2, Address Line 3 and Postcode in your database you could set these to Property, Street, Locality and Postcode fields in the structure and they will be cleaned and returned in the correct named fields when matched.

The address fields that you can use to specify the address to be cleaned are as follows:

Standard Fields
Organisation
Property
Street
Locality
Town
PostalCounty
AbbreciatedPostalCounty
OptionalCounty
AbbreviatedOptionalCounty
TraditionalCounty
AdministrativeCounty
Postcode

Raw Fields
Department
OrganisationName
SubBuilding
Building
Number
DependantThoroughfare
Thoroughfare
DoubleDependantLocality
DependantLocality
Town
PostalCounty
AbbreciatedPostalCounty
OptionalCounty
AbbreviatedOptionalCounty
TraditionalCounty
AdministrativeCounty
Postcode

BS7666 Fields
Department
Organization
SubUnit

BuildingName
BuildingNumber
SubStreet
DeliveryStreet
SubLocality
DeliveryLocality
DeliveryTown
PostalCounty
AbbreciatedPostalCounty
OptionalCounty
AbbreviatedOptionalCounty
TraditionalCounty
AdministrativeCounty
Code

Large Number of Address Lines

If you have more address lines then a normal address will allow you can still send these to the Refiner API simply by making use of all the county fields in order, for example with standard fields:

Address Line 1	=	Organisation
Address Line 2	=	Property
Address Line 3	=	Street
Address Line 4	=	Locality
Address Line 5	=	Town
Address Line 6	=	PostalCounty
Address Line 7	=	AbbreciatedPostalCounty
Address Line 8	=	OptionalCounty
Address Line 9	=	AbbreviatedOptionalCounty
Address Line 10	=	TraditionalCounty
Address Line 11	=	AdministrativeCounty
Address Line 12	=	Postcode

If your address lines are too long for the default lengths (e.g. County fields are only 30 characters long) you can increase these in your Field Specification as required.

Cleaned Address

Refiner will always output the cleaned data in the correct fields for the format you have selected. You can write the data back to your database in any form that you wish.

Appendix G. BS7666-5:2006

When using AFD Address Management products the Common API has the option to return an address in BS7666 format. You should note that this is the proposed BS7666 Part 5 (BS 7666-5:2006) and AFD will ensure that any necessary changes are made to conform to the final specification for Part 5 of BS7666 when it is released.

The proposed Part 5 of the BS7666 specification is the one which is intended for a delivery point gazetteer and so is designed to provide a standard for postal addresses of the type which AFD Software provides. Older standards such as BS 7666-3:2000 dealt with address specification but these were not specific to postal addresses and as such were not suitable for providing address data.

You should note that, strictly speaking, only AFD Postcode Plus is a delivery point gazetteer which can comply with this standard, as it contains addresses at delivery point level. However, both AFD Names & Numbers and AFD Postcode can also return fields in this format which can be used to aid BS7666 compliance in your address database. AFD Names & Numbers contains these compliant addresses along with additional Name data.

This appendix explains how to use the fields returned by the Common API in this address format to capture addresses compliant with this standard.

Gazetteer MetaData

You will require the following Gazetteer MetaData to use for compliance with the proposed BS7666-5:2006:

Field	Value
Name	“Postal Delivery Point Gazetteer”
Scope	“Premises receiving a postal delivery from Royal Mail”
Territory of Use	“Great Britain, Northern Ireland, Isle of Man and Channel Islands”
Gazetteer Owner	“AFD Software Ltd”
Custodian	“AFD Software Ltd”
Coordinate System	“National Grid of Great Britain”
Spatial Referencing System	“Postal Address”
Current Date	<Use the BuildDate Field from AFDData>
Language	“ENG”

Delivery Point Records

For each delivery point record returned from a Lookup or Search in the AFD data the following shows how the BS7666 fields returned correlate to those in the proposed BS7666-5:2006 standard:

Field	AFD Field To Use
Identifier	Identifier
Start Date	BuildDate – AFD products are a complete data refresh so they come into use at the build date.

Entry Date	BuildDate – You may wish to change this to the date you input an address onto your database.
Update Date	BuildDate – You may wish to change this to the date you update an address onto your database.
Position	
X	GBGridE
Y	GBGridN
Spatial Reference	
Identifier	Identifier
Language	Language
Department	Department
Organization	Organization
Sub-Unit	SubUnit
Building Name	BuildingName
Building Number	BuildingNumber
Sub-Street	SubStreet
Delivery Street	DeliveryStreet
Sub-Locality	SubLocality
Delivery Locality	DeliveryLocality
Delivery Town	DeliveryTown
County	*This is an optional attribute and if you wish to include a County you can use any of the County fields defined in Appendix A as you desire.*
Code	Code
Administrator	Administrator

Data Quality Report

For BS 7666-5:2006 compliance a data quality report is now also required.

Lineage

The delivery point information present in AFD Postcode Plus is derived from the Royal Mail Postcode Address File, with updates provided on a quarterly or annual basis depending on if you have purchased quarterly updates. The data in the Royal Mail Postcode Address File is included in it's entirety however is processed to conform to the BS 7666-5:2006 standard, which PAF in it's raw form does not comply with. In AFD Names & Numbers additional postal delivery points are also available which are obtained from Electoral Roll data but which do not appear on PAF.

Currency

This date is specified by the BuildDate Field of the AFDData structure.

Positional Accuracy

Using the default DataTalk GeoRef grid references, the co-ordinates of the gazetteer are to a 10m resolution. These provide the approximate location of the postcode for which the address falls.

If using Royal Mail Postzon grid references, the co-ordinates of the gazetteer are to 100m resolution and are provided by the Gridlink® consortium (of which the Office of National Statistics (ONS), Ordnance Survery (OS), Ordnance Survey of Northern Ireland (OSNI), the General Register Office for Scotland (GROS) and Royal Mail are all a part). These are generally given as the top

left of the 100m square for which the property at the centre of the postcode falls. They are constantly verified and updated and full details of their accuracy should be obtained from GridLink members if required.

If accuracy is important, and for a resolution within 1m of the postcode, Ordnance Survey CodePoint grids are an optional extra for address management products and are used in-place of GeoRef/Postzon grid references.

Attribute Accuracy
The data in AFD Postcode Plus is accurate in that it contains all postal delivery points held by Royal Mail. Royal Mail PAF is quoted as being 96.1% accurate in 2003 and a new accuracy measure is being developed which should be able to provide a better picture of its accuracy. AFD Names & Numbers contains Royal Mail PAF data to the same accuracy along with additional addresses present on the Electoral Roll but not in PAF.

Completeness
The data in AFD Postcode Plus is accurate and contains 100% of PAF records with no duplicates and tests against the original PAF data are carried out to verify this. Royal Mail PAF is quoted as being 96.1% accurate and that reflects it's completeness too – a better break down should be available once Royal Mail have developed a new accuracy measure. AFD Names & Numbers contains Royal Mail PAF data to the same accuracy, and additional addresses are presented as complete as is possible – no independent measure of that accuracy is available. They may well be duplicate addresses on AFD Names & Numbers as they cannot be matched to PAF but may identify the same delivery point.

Logical Consistency
The records in the data have been tested against the specification for the gazetteer to ensure that all are recorded in a consistent manual. This was done with both fully automated tests, and manually sampling addresses to ensure the format they appear is both consistent with the proposed BS 7666-5:2006 standard and with the other addresses present.

Appendix H. Grid References

AFD Postcode Plotter, AFD Postcode Plus, AFD Names & Numbers, and AFD Names & Numbers TraceMaster all contain grid references. DataTalk GeoRef is a Postcode level grid reference data supplied by AFD for distance calculations, nearest calculations and data / location analysis. It is made up of a six digit Easting and a six digit Northing. This reference relates the location of the Postcode to the National Grid (or Irish Grid for Northern Ireland Postcodes (start with BT).

DataTalk GeoRef is a good alternative to Postzon. However, Postzon, Code Point and Address Point data options are available at additional cost.

The grid references returned by these products are available in the following Fields:

GridE, GridN

This pair of grid references denotes the grid reference for the postcode on the National Grid of Great Britain for all postcodes other than those in Northern Ireland. For Northern Ireland the grid reference is returned in the Irish grid. The Grid References are provided as a 6 digit grid northing (7 digits for some northernmost parts of Scotland) and a 6 digit grid easting.

GBGridE, GBGridN

This pair of grid references denotes the grid reference for the postcode on the National Grid of Great Britain. This is used for all addresses in England, Scotland and Wales and can also be convenient to use for addresses in Northern Ireland in order to provide a common baseline with the rest of the UK. The Grid References are provided as a 6 digit grid northing (7 digits for some northernmost parts of Scotland) and a 6 digit grid easting.

NIGridE, NIGridN

This pair of grid references denotes the grid reference for the postcode on the Irish Grid System. This is used for all addresses in Northern Ireland, and a conversion is provided for addresses in the rest of the UK so it can be used as a common baseline if preferred for companies operating mainly in Northern Ireland. The Grid References are provided as a 6 digit grid northing and a 6 digit grid easting.

Latitude, Longitude

This pair of latitude and longitude values are provided in decimal format to four decimal places. These values are based on the WGS84 standard - the one in most common usage with GPS systems. However you should note that these references can only be as good as the grid references that they are converted from.

Textual Latitude, Textual Longitude

These also provide the latitude and longitude values as above but they are provided in a textual representation which, while being less useful as input to computer related applications, can be more readable to a user as they provide the direction, degrees, minutes and seconds components of the latitude and longitude value for this location.