Ordnance Survey®

Products and services **Getting started guide**

AddressBase® products

v1.2

Preface

Purpose and disclaimer

This getting started guide is provided for guidance only and does not constitute any warranty, representation, undertaking, commitment or obligation (express or implied) about the product or its suitability for any particular or intended purpose. Any warranties, representations, undertakings, commitments and obligations given by Ordnance Survey about the product and/or its suitability for any particular or intended purpose are set out in your customer contract. It is your responsibility to ensure that this product is suitable for your intended purpose.

Ordnance Survey does not accept any liability (whether for breach of contract, negligence or otherwise) for any loss or liability you or any third party may suffer in relying on this guide and any guidance, suggestion, advice or explanation provided in it. Any liability that Ordnance Survey has to you in relation to the product, its supply, use, accuracy, data supplied, functionality or any other liability arising out of or in connection with the product is limited as set out in your customer contract.

We may change the information in this guide at any time without notice.

We do not accept responsibility for the content of any third party websites referenced or accessed.

Data copyright and other intellectual property rights

The Crown (or, where applicable, Ordnance Survey's suppliers) owns the intellectual property rights in Ordnance Survey digital map data.

Full details of the terms and conditions under which Ordnance Survey digital map data may be processed and/or manipulated or copied by a customer – whether or not for use on PCs or workstations or for making hard copies – are available from the Customer Service Centre, please see Further Information. You should check the terms and conditions with us before using the data. It is also the responsibility of the holder of the digital map data to ensure that any plotted or printed output contains the required copyright and database acknowledgements in a conspicuous position.

The data in AddressBase is provided by GeoPlace, a limited liability partnership between the Local Government Group and Ordnance Survey.

1 AddressBase data applications

Three products have been designed to meet distinct customer requirements. All are derived from a single database, which captures data from multiple authoritative address sources. The source data is collated, verified and quality assured by GeoPlace. This database is then used to create the AddressBase products.

1.1 Examples of applications for AddressBase products

Product name	AddressBase Premium	AddressBase Plus	AddressBase
Suggested application	Planning, mailing, postal delivery, analysis, statistics, strategic decision-making, address matching and verification, customer relationship management (CRM), web mapping, live 'front line' operational use.	Planning, mailing, postal delivery, analysis, statistics, strategic decision-making, address-matching and verification, CRM, web mapping.	Mailing, postal delivery.
Benefits	Key building block for 'e-government': • reduces duplication on effort on maintaining address information across multiple departments; and • facilitates data-sharing between departmental systems. Provides objects without a postal address (OWPA) records and multiple address references. Connectivity – the Unique Property Reference Number (UPRN) as the key identifier for a property/ address that enables systems to share information about the same entities without the need to match multiple datasets. Created from a central hub managed by GeoPlace to bring all the address information together to ensure data management and update consistency.		

2 What do I need to use this data?

2.1 System requirements

AddressBase data is designed for use as a digital map within geographical information systems (GIS) and database systems. For details of Ordnance Survey's Licensed Partners, who can incorporate AddressBase in their systems, please see the systems/software page on the Ordnance Survey website.

Ordnance Survey does not recommend either suppliers or software products, as the most appropriate system will depend on many factors, such as the amount of data being taken, resources available within the organisation, the existing and planned information technology infrastructure and last but by no means least, the applications that AddressBase products can be used for.

However as a minimum, the following elements will be required in any system:

- a means of reading the data, either in its native format, or by translating it into a file format or for storage in a database;
- a means of storing and distributing the data, perhaps in a database or through a web-based service; and
- a way of visualising and querying the data, typically a GIS.

2.2 Backup provision of the product

You are advised to copy the supplied data to a backup medium.

2.3 Typical data volumes

For reading purposes it is recommended that users store the data on a single hard disc. This will speed up the ability of your computer to read the data.

Uncompressed file sizes for the full supply of England and Wales are as follows:

2.3.1 Uncompressed comma-separated values (CSV)

- AddressBase Premium is 29 Gb
- AddressBase Plus is 13 Gb
- AddressBase 4.9 Gb

2.3.2 Uncompressed Geography Markup Language (GML)

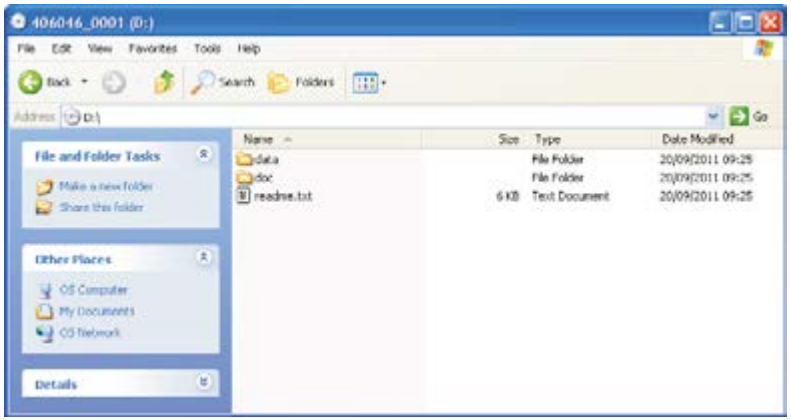
- AddressBase Premium is 133 Gb
- AddressBase Plus is 62 Gb
- AddressBase 25 Gb

3 What's on the CD/DVD or in the data download?

3.1 Supply options:

3.1.1 CD/DVD
When a customer receives an order via offline media, the following files will be supplied.

- data
- doc
- readme.txt



Within the data directory, data files will be found in their compressed format.

The doc directory contains both standard and product-specific document files that describe what has been supplied in the order, including:

- Medialist.txt – outlining the contents of the media.
- Disccare.txt – outlining how to care for your media.
- Report.txt – outlining the order details.
- readme.txt – this document provides guidance notes on matters such as the file name referencing used and the directory structure of the DVD.

3.1.1 File Transfer Protocol (FTP)

With an FTP order, the same information is supplied as in section 3.1.1; but the file names will be slightly different, reflecting the FTP order number.

3.1.2 Download

Public Sector Mapping Agreement (PSMA) customers can download their geographic chunk data via our download service. When you click 'Download data', you will be required to enter a password to access the PSMA members' area. On successful entry to the download service, you will be able to view all of your orders in the members' area and download your data.

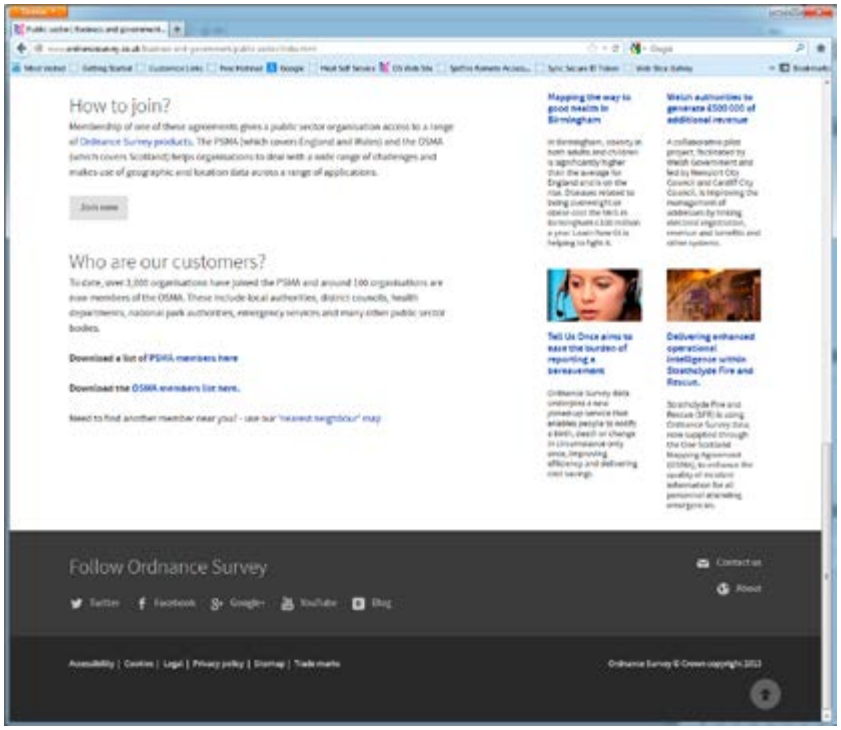


Figure 1: download data from our website

If you have ordered your data offline, you will be sent an email with a link to the download page.

Online Reference - 234831 - AddressBase

Thank you for placing your order for Ordnance Survey map data.

We are pleased to inform you that your order placed on Thursday 29th September 2011 is now available for download.

Please click here to download your data

Your order will be available to download for **seven days** from the date of this email.

Please note that larger files may take some time to download depending on your broadband/dial up connection speed.

Should you require any assistance with your download or have received this email in error, please contact our Business enquiries help-desk on 08453 757535 and we will endeavour to resolve your issue. Our office opening hours are Monday to Friday 08:30 - 17:30 except for Public Holidays.

Kind regards,

Figure 2: download data via an email link

Within the PSMA members' area, you can download the data that you require by clicking on the 'Download' button (on the right of the screen).

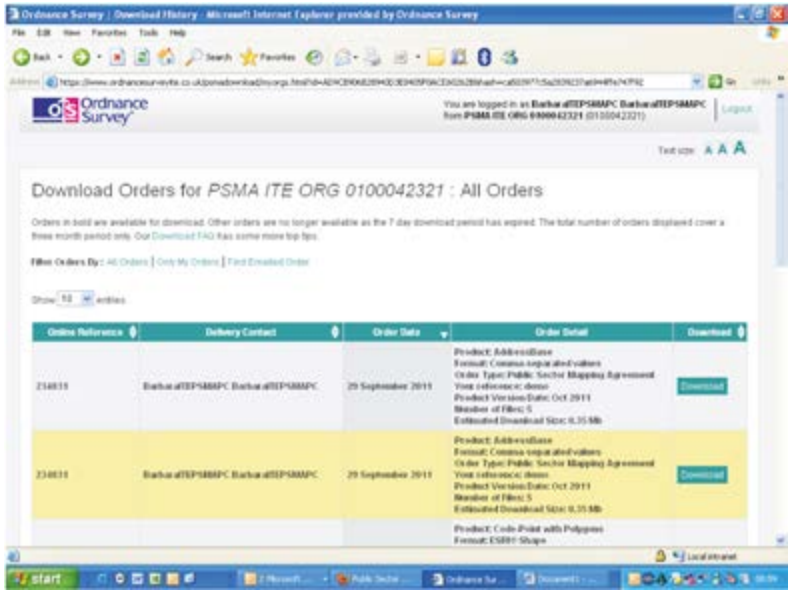


Figure 3: PSMA members' area

When you've downloaded the data, it will be available as a series of zipped data files. To unzip these files please see chapter 3.3, below.

Figure 4: zipped data

3.2 Chunked files

The data is supplied as a number of chunked files that cover your selected area. These files are named according to the convention detailed below.

When you open your data, you will see a series of zip folders.

3.2.1 Non-geographic chunks

For example:

AddressBasePremium_FULL_2011-07-29_001_csv.zip
(Full supply of CSV)

or

AddressBasePremium_COU_2011-07-29_001_gml.zip
(Change-only update supply of GML)

3.2.2 Geographic chunks

For example:

AddressBasePremium_FULL_2011-07-29_TQ2020_csv.zip
(Full supply of CSV)

or

AddressBasePremium_COU_2011-07-29_TQ2020_gml.zip
(Change-only update supply of GML)

3.3 How do I unzip the files?

The AddressBase GML and CSV data is supplied in a compressed form (zip). Some software can access these files directly; others will require it to be uncompressed. To uncompress the zipped data files (.zip extension), use an unzipping utility found on most PCs, for example, WinZip®. Open-source zipping/unzipping software can be downloaded from the Internet, for example, 7-Zip.

3.4 Unzipped

When the files are unzipped they will appear as follows:

3.4.1 Non-geographic chunks

AddressBasePremium_2011-07-29_001.csv

3.4.2 Geographic chunks

AddressBasePremium_2011-07-29_NC4040.csv

These CSV files are now ready for use.

4 Managing the CSV data

The technique for managing AddressBase CSV data is determined by which of the three products was taken: AddressBase, AddressBase Plus or AddressBase Premium.

If using AddressBase or AddressBase Plus data, refer to Chapter 4.1. If using AddressBase Premium data please refer to Chapter 4.2.

4.1 AddressBase and AddressBase Plus

4.1.1 Merging multiple AddressBase or AddressBase Plus CSV files

For AddressBase and AddressBase Plus data – using the technique described in Chapter 3.3, unzip all the CSV files into a single folder, for example, C:\AddressBase_Data or C:\AddressBase_Plus_Data.

It may be beneficial to merge all CSV files together to ensure that the user does not have to follow the import procedure for each of the individual files, which can be time-consuming and repetitive. The user can use any technique they feel comfortable with to merge all the individual CSV files in to a single file. This could include doing it manually using a text editor such as Notepad or TextPad (though this is very time-consuming), using a .bat batch file, or an MS-DOS® command.

To use the batch function:

- Copy the following text and paste it into a new Notepad document:

  *copy *.csv mergedABdata.csv*

  NOTE: mergedABdata.csv can be any user-defined file name with the extension .csv

- Save the Notepad document with the file extension .bat (for example, *mergedABdata.bat*) in the same directory as the CSV files unzipped in Chapter 3.3 (for example, *C:\AddressBase_Data*).
- Close the .bat file, and navigate to the directory where it was saved (for example, *C:\AddressBase_Data*). Double-click on the .bat file (for example, *mergedABdata.bat*) and an MS DOS window will appear. Once the process is complete, the MS-DOS screen will close automatically.
- If you look in the directory containing the AddressBase CSV files, and batch file (for example, *C:\ AddressBase_Data*), it can be seen that there is now an additional single file called *mergedABdata.csv*

4.1.2 Appending a header file to AddressBase and AddressBase Plus CSV

The three AddressBase CSV products, AddressBase, AddressBase Plus and AddressBase Premium all contain different attributes. As such, there is a separate header file for each of these. Header files for each product are supplied by Ordnance Survey and can be found on our **web page**.

Download the Header file that corresponds to the product that has been supplied:

- **AddressBase**
- **AddressBase Plus**

Paste the .csv header file contained within the zip folder into the same folder as the merged AddressBase.csv file created in Chapter 4.1.1.

Copy the appropriate version of the following text and paste it into a new Notepad document:

AddressBase

copy *AddressBase_Header.csv+mergedABdata.csv AB_Data.csv*

AddressBase Plus

copy *AddressBasePlus_Header.csv+ mergedAB_Plusdata.csv AB_Plus_Data.csv*

NOTE: mergedABdata.csv is the file that contains all of the AddressBase data merged into a single .csv file.

The order that the documents are referred to in the above text is important, as it states which file is appended to the other. In this instance the headers .csv file comes first, so that the column headers are the first line of the final AddressBase data file and the merged data file is appended to the column headers.

- Save the above Notepad document with the file extension .bat (for example, *append.bat*) in the same directory as where the column headers and the merged AddressBase data are located (for example, *C:\AddressBase_Data*).
- Close the .bat file and navigate to the directory where it was saved to (for example, *C:\AddressBase_Data*). Double-click on the new .bat file (for example, *append.bat*) and an MS-DOS window will appear. Once the process is complete, the MS-DOS screen will close automatically.
- If the user navigates to the directory where the column headers and the merged AddressBase data are located, it can be noticed that a new .csv file has been created, which is the merged column headers and AddressBase data (for example, *AB_Data.csv* or *AB_Plus_Data.csv*).

4.2 AddressBase Premium

Appending header files to AddressBase Premium can be done at the same time as splitting the record identifiers by following these instructions:

- Group all the AddressBase Premium CSV files into a folder with no other CSV files contained within.

  NOTE: this folder must contain no spaces in any of the file directory path, for example, C:\AddressBaseData\AddressBase_Premium

- Into this folder, add the **AddressBase Premium header files**:

Record_10_HEADER_Header.csv
Record_11_STREET_Header.csv
Record_15_STREETDESCRIPTOR_Header.csv
Record_21_BLPU_Header.csv
Record_23_XREF_Header.csv
Record_24_LPI_Header.csv
Record_28_DELIVERYPOINTADDRESS_Header.csv
Record_29_METADATA_Header.csv
Record_30_SUCCESSOR_Header.csv
Record_31_ORGANISATION_Header.csv
Record_32_CLASSIFICATION_Header.csv
Record_99_TRAILER_Header.csv

- Go to **http://omniplex.om.funpic.de/home/ltru/gawk-win.zip** and save the zip file in any location.
  - Unzip the file.
  - Copy the 'gawk.exe' file, contained within the 'gawk-win' folder, to the same folder as where all off the AddressBase Premium CSV and header files are located, for example, *C:\AddressBaseData\AddressBase_Premium*.

- Copy the following text and paste it into a new Notepad document:

```
FOR /F %%A IN ('dir *.csv /b/s') DO (CALL :process "%%A" "%%~NA")

@rem -- merge the individual record identifier files
copy *_10_Records.csv Master_10_Records.out
copy *_11_Records.csv Master_11_Records.out
copy *_15_Records.csv Master_15_Records.out
copy *_21_Records.csv Master_21_Records.out
copy *_23_Records.csv Master_23_Records.out
copy *_24_Records.csv Master_24_Records.out
copy *_28_Records.csv Master_28_Records.out
copy *_29_Records.csv Master_29_Records.out
copy *_30_Records.csv Master_30_Records.out
copy *_31_Records.csv Master_31_Records.out
copy *_32_Records.csv Master_32_Records.out
copy *_99_Records.csv Master_99_Records.out
del *_Records.csv
```

```
@rem -- add header records to the individual record identifier files
copy Record_10_HEADER_Header.csv+Master_10_Records.out ID10_Header_Records.csv
copy Record_11_STREET_Header.csv+Master_11_Records.out ID11_Street_Records.csv
copy Record_15_STREETDESCRIPTOR_Header.csv+Master_15_Records.out ID15_StreetDesc_Records.csv
copy Record_21_BLPU_Header.csv+Master_21_Records.out ID21_BLPU_Records.csv
copy Record_23_XREF_Header.csv+Master_23_Records.out ID23_XREF_Records.csv
copy Record_24_LPI_Header.csv+Master_24_Records.out ID24_LPI_Records.csv
copy Record_28_DELIVERYPOINTADDRESS_Header.csv+Master_28_Records.out ID28_DPA_Records.csv
copy Record_29_METADATA_Header.csv+Master_29_Records.out ID29_Metadata_Records.csv
copy Record_30_SUCCESSOR_Header.csv+Master_30_Records.out ID30_Successor_Records.csv
copy Record_31_ORGANISATION_Header.csv+Master_31_Records.out ID31_Org_Records.csv
copy Record_32_CLASSIFICATION_Header.csv+Master_32_Records.out ID32_Class_Records.csv
copy Record_99_TRAILER_Header.csv+Master_99_Records.out ID99_Trailer_Records.csv
del *.out
pause
exit

@rem -- split the source csv into individual files based on the record identifier
:process
SET tempvar1=%~1
SET tempvar2=%~2
gawk < %tempvar1% -F "," "{ if ($1 == \"10\") { print $0 } }" > %tempvar2%_10_Records.csv
gawk < %tempvar1% -F "," "{ if ($1 == \"11\") { print $0 } }" > %tempvar2%_11_Records.csv
gawk < %tempvar1% -F "," "{ if ($1 == \"15\") { print $0 } }" > %tempvar2%_15_Records.csv
gawk < %tempvar1% -F "," "{ if ($1 == \"21\") { print $0 } }" > %tempvar2%_21_Records.csv
gawk < %tempvar1% -F "," "{ if ($1 == \"23\") { print $0 } }" > %tempvar2%_23_Records.csv
gawk < %tempvar1% -F "," "{ if ($1 == \"24\") { print $0 } }" > %tempvar2%_24_Records.csv
gawk < %tempvar1% -F "," "{ if ($1 == \"28\") { print $0 } }" > %tempvar2%_28_Records.csv
gawk < %tempvar1% -F "," "{ if ($1 == \"29\") { print $0 } }" > %tempvar2%_29_Records.csv
gawk < %tempvar1% -F "," "{ if ($1 == \"30\") { print $0 } }" > %tempvar2%_30_Records.csv
gawk < %tempvar1% -F "," "{ if ($1 == \"31\") { print $0 } }" > %tempvar2%_31_Records.csv
gawk < %tempvar1% -F "," "{ if ($1 == \"32\") { print $0 } }" > %tempvar2%_32_Records.csv
gawk < %tempvar1% -F "," "{ if ($1 == \"99\") { print $0 } }" > %tempvar2%_99_Records.csv
GOTO :EOF
```

- Save the Notepad document with the file extension .bat (for example, process_AB_Premium.bat) in the same directory as the AddressBase Premium CSV files, header files and gawk.exe file (for example, *C:\AddressBaseData\AddressBase_Premium*).
- Close the .bat file, and navigate to the directory where it was saved (for example, *C:\AddressBaseData\AddressBase_Premium*). Double-click on the .bat file (for example, *process_AB_Premium.bat*) and an MS-DOS window will appear. Once the process is complete, the MS-DOS screen will close automatically.
- If you look in the directory containing the AddressBase Premium CSV files, header files, gawk.exe file, and batch file (for example, *C:\AddressBaseData\AddressBase_Premium*), it can be seen that there are now files that adhere to similar naming conventions as the header files, which contain the relevant AddressBase Premium data.

5 How do I load the CSV AddressBase products into a GIS?

AddressBase products can be loaded into several Geographic Information System (GIS). This chapter describes how to load AddressBase products into commonly used GIS. For more information on other GIS that AddressBase products are compatible with, please speak to your Account Manager.

5.1 Esri®

These instructions are for use with ArcGIS Desktop 9.3 Service Pack 1.

When using CSV data in ArcGIS, it is necessary to have column headings. The default headings given by ArcGIS are numeric and have no real relevance to AddressBase, hence it is more appropriate to use the headings provided by Ordnance Survey. Instructions on how to merge the appropriate header files and data can be found in Chapter 4.

5.1.1 AddressBase and AddressBase Plus

- Start ArcCatalog.
- Connect to a folder where the AddressBase data you wish to use can be accessed, for example, *C:\AddressBase_Data* by:
  - clicking File;
  - then Connect Folder, and navigate to the relevant folder;
  - select the folder that is to be connected to from the main window; then
  - click OK.
- The folder should now appear in the navigation window to the left of the screen.
- Using the file tree on the left, navigate to the directory where you wish to create the File Geodatabase, for example, *C:\AddressBase_Geodatabase\AddressBase_Plus*.
- Right-click on the folder that you wish to contain the File Geodatabase, and select 'New' and 'File Geodatabase'.
- Rename the File Geodatabase to something relevant, for example, *'AB_Plus_FileDb.gdb'*, and press Enter.
- Right-click on the newly-created File Geodatabase, for example, *'AB_Plus_FileDb.gdb'*, and select 'Import', then 'Table (single)…'
  - For 'Input Rows', navigate to the location of the CSV file that you wish to open, that is, the file that contains the merged header and AddressBase Plus data file.
  - The 'Output Location' option should automatically populated by the location of the File Geodatabase that is to be updated, that is, *'AddressBase_Plus_FileDb.gdb'*.
  - Insert a relevant name for the Output Table, for example, *AddressBase_Plus_data*. *NOTE: there can be no spaces in the table name*.
- Click 'OK'.

To create a map of the locations of the AddressBase records, they need to be geocoded. To do this:

- Right-click on the AddressBase table in the geodatabase that was created in the previous step and select Create Feature Class.

- Then, From XY Table…
  - Select the X_Coordinate attribute from the X Field: drop-down menu and the Y_Coordinate attribute from the Y Field: drop-down menu. Leave the Z Field: drop-down menu as <none>.
  - Then click on the Coordinate System of Input Coordinates… button, click Select…, and then navigate to and select the British National Grid – which should be located in the directory Coordinate Systems\Projected Coordinate Systems\National Grids. Click Add and then OK.
  - Click on the open folder button on the right of the text box referring to the Output, navigate to the location where you wish to save the output shapefile or feature class (it is suggested that this be within the geodatabase created above) and give the file a suitable name (for example, *AddressBase_Data_geocoded*), changing the Save as Type: option to File and Personal Geodatabase feature classes.
  - Click Save, leave the Configuration keyword: drop-down menu as Defaults, and press OK.
- So that the new feature class can be seen, right-click on the Personal Geodatabase where it was saved and select Refresh.

Now that all of the processing has been done, the data can be loaded into ArcMap:

- Start ArcMap.
- Select File > Add Data…
- Navigate to the folder where the AddressBase file Geodatabase was created.
- Double-click on the Geodatabase, and select all of the files contained within in one go.
- Click Add.

5.1.2 AddressBase Premium

Once the AddressBase Premium data has been split from the single CSV file that it is supplied in, into the CSV files for each of the individual record types, by following the steps outlined in Chapter 4, the files should be loaded into ArcCatalog:

- Start ArcCatalog.
- Connect to a folder where the AddressBase data you wish to use can be accessed, for example, C:\AddressBase_Data by:
  - clicking File;
  - then Connect Folder, and navigate to the relevant folder;
  - select the folder that is to be connected to from the main window; then
  - click OK.
- The folder should now appear in the navigation window to the left of the screen.
- Using the file tree on the left, navigate to the directory where you wish to create the File Geodatabase.
- Right click on the folder which you wish to contain the File Geodatabase, and select 'New' and 'File Geodatabase'.
- Rename the File Geodatabase to something relevant, for example, *'AddressBase_Premium.gdb'*, and press Enter.
- Right-click on the newly-created File Geodatabase, for example, *'AddressBase_Premium.gdb'*, and select 'Import', then 'Table (multiple)…'
  - For 'Input Table', navigate to the location of the .csv files that you wish to open, for example, the folder that contains the AddressBase data split into individual files by record type and select the files that you wish to add. Click 'Add'.
  - The 'Output Geodatabase' option should automatically be populated by the location of the File Geodatabase that is to be updated, for example, *'AddressBase_Premium.gdb'*.
  - Click OK.
- Once the process is complete, click Close.

To create a map of the locations of the AddressBase Premium records, they need to be geocoded. To do this:

- Double-click on the geodatabase that the AddressBase data was just imported into.
- Right-click on the table that was created from AddressBase Premium records with a record type of 21, for example, 'ID21_BLPU_Records', in the geodatabase and select Create Feature Class.
- Then, From XY Table…
  - Select the X_Coordinate attribute from the X Field: drop-down menu and the Y_Coordinate attribute from the Y Field: drop-down menu. Leave the Z Field: drop-down menu as <none>.
  - Click on the Coordinate System of Input Coordinates… button, click Select…, and then navigate to and select the British National Grid – which should be located in the directory Coordinate Systems\Projected Coordinate Systems\National Grids. Click Add and then OK.
  - Click on the open folder button on the right of the text box referring to the Output, navigate to the location where you wish to save the output shapefile or feature class (it is suggested that this be within the geodatabase created in 6.1.2) and give the file a suitable name (for example, *XYID21_BLPU_Records*), changing the Save as Type: option to File and Personal Geodatabase feature classes.
  - Click Save, leave the Configuration keyword: drop-down menu as Defaults, and press OK.
- So that the new feature class can be seen, right-click on the Personal Geodatabase where it was saved and select Refresh.

Now that all of the processing has been done, the data should be loaded into ArcMap so that the individual tables, split by record type, can be 'related'.

- Start ArcMap.
- Select File > Add Data…
- Navigate to the folder where the AddressBase Premium file Geodatabase was created.
- Double-click on the Geodatabase, and select all of the files contained within in one go.
- Click Add.

The way in which all of the individual tables are related/joined can be found within the AddressBase Premium technical specification.

However, for ease of reference, the following joins/relates should be made:

- BLPU (spatial data for record identifier 21)
  - UPRN – Application Cross Reference (record identifier 23) UPRN
  - UPRN – LPI (record identifier 24) UPRN
  - UPRN – Delivery Point Address (record identifier 28) UPRN
  - UPRN – Successor Record (record identifier 30) UPRN
  - UPRN – Organisation (record identifier 31) UPRN
  - UPRN – Classification (record identifier 32) UPRN
- LPI (record identifier 24) USRN – Street (record identifier 11) USRN
- Street (record identifier 11) USRN – Street Descriptor (record identifier 15) USRN

To create the relevant relates:

- With the Source tab selected in the left-hand navigation window, right-click on the first table you wish to relate to another:
  - select Joins and Relates; then
  - Relate…
- From the first drop-down, select the attribute from the first table that is going to be used to create the relate between the two tables (relationships stated above).
- From the second drop-down menu, select the table that is going to be related to (relationships stated above).
- From the third drop-down menu, select the attribute from the table that is being related to (relationships stated above).
- In the fourth box, input a relevant name for the relate, for example, *BLPU_to_Organisation*.
- Click OK.
- Repeat this process for all of the joins/relates.

5.1.3 Helpful tip

Once the data has been loaded into ArcMap, if the user wishes to display more relevant information in the 'Info' tool than the Esri defined Object ID, it is possible to change this by:

- double-click on the spatial dataset that you wish to change the Primary Display Field of;
- select the 'Fields' tab; then
- change the Primary Display Field to the desired field, for example, for AddressBase and AddressBase Plus data – UPRN.

5.2 MapInfo®

These instructions are based on the use of MapInfo Professional v10.5.2.

When using CSV data in MapInfo, it is not a critical requirement to have column headings; however, for ease of use of the data, it is recommended that the default headings supplied by
Ordnance Survey are used in conjunction with the data. Instructions on how to merge the appropriate header files and data can be found in chapter 4.

5.2.1 AddressBase and AddressBase Plus

- Start MapInfo.
- 'Cancel' the Quick Start prompt.
- Click 'File', then 'Open…', and navigate to the folder that contains the AddressBase data combined with the appropriate header file created in chapter 4.
- In the 'Files of Type' drop down menu, select 'Comma delimited CSV (*.csv)', and select the merged AddressBase data and header file to be loaded. Click 'Open'.
- The tick box next to 'Use First Line for Column Titles' should be checked. Then press OK.

5.2.2 Format attributes

When adding data this way, the field type classifications and field sizes of each column are done automatically – to fit the type of data that MapInfo believes is contained within the column and the largest value of that classification found within that column. The classifications and field sizes of some attributes may not match the field types and sizes stated in the AddressBase technical specification. These should be changed so that they match the values stated. It is only possible to change the classification types and values in an editable copy of a table – as the initial MapInfo table created simply references the CSV file that was opened and it is not possible to edit the classification of an attribute. Hence, a copy will have to be created so that the edits can be made.

- Go to File, Save Copy As…, select the AddressBase table that was loaded in, Save As…, name the table to be created, then click Save.
- Open the table that was just created. File, Open…, navigate to and select the copy of <copy_of_AddressBase>, click Open.

Once the copy has been saved and opened, to check and change any attribute classifications that do not match those stated in the AddressBase technical specification, navigate to: Table > Maintenance > Table Structure…then select the table to be edited and click OK.

On this screen, it is possible to change the Type and Width of each attribute to match that stated in the AddressBase technical specification. This should be adhered to for all attributes apart from the UPRN, which should be classified as Float, and all attributes that have a Field Type of 'Date' in the AddressBase technical specification, which should be classified as Character with a length of 10. These discrepancies are due to software-specific issues in handling the data. After all changes have been made, click OK.

5.2.3 Geocoding

In order to create a map of the location of the AddressBase records, they need to be geocoded. To do this, ensure that the table of AddressBase records that you wish to geocode is open, and then:

- Navigate to 'Table', Create Points…
- Select:
  - The table you wish to geocode from the Create Points for Table: drop-down menu.
  - The X_Coordinate attribute from the Get X Coordinates from Column: drop-down menu.
  - The Y_Coordinate attribute from the Get Y Coordinates from Column: drop-down menu.
- Then click on the Projection… button:
  - Select the British Coordinate Systems option from the Category drop-down menu and then the British National Grid [EPSG: 27700].
  - Click OK to close that screen, and once again, OK to close the next screen.
- To view the geocoded points, go to Window, New Map Window.

5.2.4 General considerations

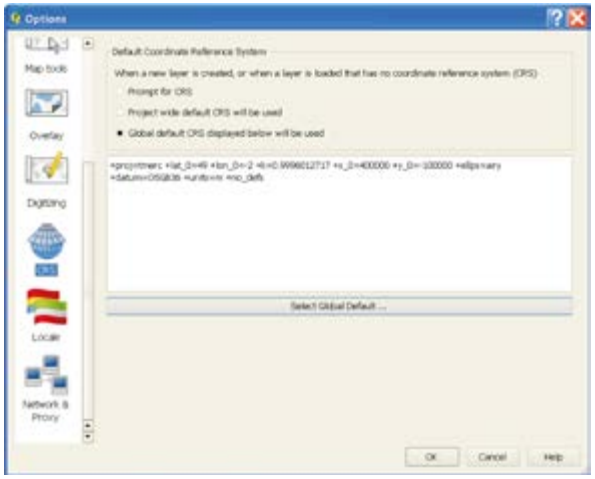
MapInfo has a size limit of 2 Gb on each table. This equates to a maximum number of approximately 4 million AddressBase records.

5.3 QGIS

This method is designed for use with QGIS 1.6.0.

5.3.1 Set projection

- Click Settings
  - Then Options…



Select 'CRS' from the left-hand side menu.
  - Ensure that 'Global default CRS displayed below will be used.
  - Click on 'Select Global Default…'
    - In the Search area:
- ensure Authority = All
- Search for = ID
- in the text box insert '27700' and press Find.
    - In the Coordinate Reference System Selector at the top of the screen, ensure that 'OSGB 1936 / British National Grid' is highlighted.
    - Click OK.
  - Click OK.

5.3.2 AddressBase and AddressBase Plus

- Go to Plugins.
  - Then Manage Plugins…
- Check the checkbox next to Add Delimited Text Layer.
  - Then OK.
- Return to Plugins.
  - Select Delimited text.
    - then Add delimited text layer.
- In the dialog box, click the '…' button and navigate to the location of the AddressBase CSV file that was created in Chapter 4 – containing the merged header files and AddressBase data.
  - Select the CSV file, and press Open.
- Accept the default or create a new layer name for the dataset.
- In the delimiter string text box, type ","
- Click the Parse button, which will then allow you to select the geometry attributes:
  - For the X field, select the X_Coordinate attribute from the drop-down option.
  - For the Y field select the Y_Coordinate attribute from the drop-down option.
- Press OK.

6 Importing AddressBase products into a database

AddressBase products can be loaded into several types of database. This chapter describes how to load AddressBase products into commonly used ones. For more information on other databases that AddressBase products are compatible with, please speak to your Account Manager.

It should be noted that ArcMap, ArcGIS Desktop and ArcGIS Server software do not support the BIGINT/NUMBER data type as an object id. The use of the BIGINT/NUMBER data type should therefore be considered if the expectation is to use the data directly with these ESRI products. An alternative method to facilitate using ESRI software is to store this data as a string and add a new serial id to act as the object id.

6.1 Microsoft® Access

This method is designed for use with Microsoft Access '97.

6.1.1 AddressBase and AddressBase Plus

It is possible to import AddressBase and AddressBase Plus data directly into Microsoft Access. You must first create a Microsoft Access Database by clicking:

- File.
- New Database…
- Double click on the blank database icon. Name and save the database somewhere convenient.

After creating the database, to import the AddressBase data:

- Click File.
  - Then Get External Data.
    - Then Import…
    - Change the selection of the 'Files of type:' to Text Files.
    - Navigate to the location of the AddressBase CSV file that contains the data and headers in a single file.
    - Select the file and click Import.
- In the next box that appears, ensure the Delimited option is selected.
  - Then click Next.
- Then 'Comma' from the delimiter options.
  - Tick the First Row Contains Filed Names option.
  - Select " from the Text Qualifier drop-down menu.
  - Then Advanced:
    - change the Date Order: option to "YMD".
    - tick the Four Digit Years box.
    - change the Date Delimited to be "-".
    - tick the Leading Zeros in Dates box.
  - Click OK.
  - Then Next.

Then In a New Table.
- Then Next.

This is where some more detailed manual work becomes necessary.

- On the next screen you are prompted to specify information about each of the fields that were imported. You must select and work through each field that you have imported, individually, and ensure that the 'Data_Type:' of each field matches the Field Type classification stated for that attribute in the AddressBase technical specifications.
  - This is true for all attributes apart from the UPRN and the OS_Address_TOID, which should be given the Field Type: Text – as there are issues with Microsoft Access handling values over 2,147,483,647.
- For UPRN, you should ensure that the Indexed: box has Yes (no duplicates) is selected.
- Once these steps have been carried out, click Next.
- Select the option to 'Choose my own Primary Key', selecting the UPRN.
- Click Next.
- Finally, give your table a name and click Finish.

6.1.2 AddressBase Premium

You must first create a Microsoft Access Database by clicking:

- File.
- New Database…
- Double-click on the Blank Database icon. Name and save the database somewhere convenient.

After creating the database, to import the AddressBase Premium data:

- Click File.
  - then Get External Data.
    - Then Import…
    - Change the selection of the 'Files of type:' to Text Files.
    - Navigate to the location of the AddressBase Premium CSV files that contain the data, split by record type, and headers in a single file.
    - Select the file and click Import.
- In the next box that appears, ensure the Delimited option is selected.
  - Click Next.
- Then 'Comma' from the delimiter options.
  - Tick the First Row Contains Filed Names option.
  - Select " from the Text Qualifier drop-down menu.
  - Then Advanced,
    - Change the Date Order: option to "YMD".
    - Tick the Four Digit Years box.
    - Change the Date Delimited to be "-"
    - Tick the Leading Zeros in Dates box.
  - Then click OK.
  - Then Next.

- Then In a New Table
  - Then Next.
- Click Next.
- Select the option to 'No Primary Key'.
- Click Next.
- Finally, give your table a relevant name and click Finish.

This process should be repeated for all AddressBase Premium datasets, split by record type – importing them all into the same Access database.

Once all the datasets have been imported, they can be linked via queries. To do this:

Click Insert.
  - Then Query.
- Select Design View and press OK.
- Highlight all of the tables that are to be joined in the main body of the dialogue window, and press Add, then Close.

Hint: make the query window as large as possible here so that creating the joins graphically is easier.

- To create the joins between the relevant tables:
- Select the attribute to be joined from the first table, and without letting go of the mouse button, drag it to the matching attribute in the table to be joined to.

These are the joins to be created:

- BLPU (record identifier 21)
  - UPRN – Application Cross Reference (record identifier 23) UPRN
  - UPRN – LPI (record identifier 24) UPRN
  - UPRN – Delivery Point Address (record identifier 28) UPRN
  - UPRN – Successor Record (record identifier 30) UPRN
  - UPRN – Organisation (record identifier 31) UPRN
  - UPRN – Classification (record identifier 32) UPRN
- LPI (record identifier 24) USRN – Street (record identifier 11) USRN
- Street (record identifier 11) USRN – Street Descriptor (record identifier 15) USRN
- Once these have all been created, save the database.

6.1.3 General considerations

It should be noted that Microsoft Access '97 has a maximum table size of 1 Gb. This equates to a maximum number of approximately three million AddressBase records.

6.2 PostGresSQL

These notes outline how to load AddressBase Premium into a PostGresSQL database using the text files created using the CSV file merge utility described in the previous chapters. They have been prepared using version 1.12.3 of PostGresSQL and with an assumption that you have set-up your database with the PostGIS spatial extension. The instructions only cover loading AddressBase Premium.

It is recommended that you have an understanding of database terminology before attempting to follow this guide.

6.2.1 Loading instructions

To Load AddressBase Premium use the following steps:

1. Prepare the text files as described in Chapter 4 of this guide.

2. Remove any carriage returns from the end of the output file as this will close the import to file.

3. Open PGAdmin tool (this can be found on the Windows Start Menu – PostGreSQL).

4. Either connect to an existing database or create a new database (it is recommended that the encoding is set to UTF-8).

5. Open the public schema (although in a production environment it is advised to use a different schema) and create the tables using the following steps.
   a. Open the SQL query tool
   b. Copy the SQL scripts in section 6.2.3 into the query window. These should be copied individually; you will need to create the following tables
      - BLPU
      - Classifications
      - Cross reference table
      - Delivery Point Address
      - LPI
      - Organisation
      - Streets
      - Street Descriptor
      - Successor Records

6. Once the tables have been created the data can be loaded into each table using the SQL COPY, adding the CSV option as the first line contains a header record for each table.

Note that the path and file name may need to be changed to reflect your data.

```
COPY abp_blpu FROM 'C:/Address/ID21_BLPU_Records.csv' DELIMITER ',' CSV HEADER;
COPY abp_delivery_point FROM 'C:/Address/ID28_DPA_Records.csv' DELIMITER ',' CSV HEADER;
COPY abp_lpi FROM 'C:/Address/ID24_LPI_Records.csv' DELIMITER ',' CSV HEADER;
COPY abp_crossref FROM 'C:/Address/ID23_XREF_Records.csv' DELIMITER ',' CSV HEADER;
COPY abp_classification FROM 'C:/Address/ID32_Class_Records.csv' DELIMITER ',' CSV HEADER;
COPY abp_street FROM 'C:/Address/ID11_Street_Records.csv' DELIMITER ',' CSV HEADER;
COPY abp_street_descriptor FROM 'C:/Address/ID15_StreetDesc_Records.csv' DELIMITER ',' CSV HEADER ;
COPY abp_organisation FROM 'C:/Address/ID31_Org_Records.csv' DELIMITER ',' CSV HEADER;
COPY abp_successor FROM 'C:/Address/ID30_Successor_Records.csv' DELIMITER ',' CSV HEADER;
```

7. Once loaded you may want to add primary and foreign keys to the data. However, these can only be added on columns where the data values are unique. Where there are no unique data values an index may be added which will aid searching. For the BLPU table, the UPRN provides a unique value and USRN in the Streets table. Primary Keys are added using the following steps.
   a. Right click on the table name and select New Object – New Primary Key
   b. Enter a Name to call the key and select the Columns tab
   c. From the drop-down at the foot of the window select UPRN
   d. Click on Add
   e. Click Add then OK

Repeat the procedure for the Streets table and USRN.

8. However, in the other tables these columns may contain duplicate values. In this case use the table key, for example, LPI_Key as the primary identifier. Alternative object identifiers (OID) can be added to each table (these are also required to use the data in some GIS including QGIS and MapInfo. The following SQL can be used for this:

   ALTER TABLE *insert_table_name* SET WITH OIDS

9. To help performance when querying across mutliple tables a foreign key may be added, however, as with a primary key only unique data columns can be used.
   a. Click on the table you wish to add the key to in pgAdmin
   b. Click on the + sign
   c. Right click on Constraints and select New Object > New Foreign Key
   d. Under the Properties Tab select the table to join to from the References drop down
   e. On the same tab enter a Name for the key (for example, FKey1)
   f. Click on Columns tab
   g. Click on the unique field for Local field and the same field from the Referencing drop down
   h. Click on Add and OK
   i. Click OK

10. You will need to repeat this for each table that contain suitable fields.

11. You can also index the data by following these steps.
    a. Click on the table in pgAdmin
    b. Click on the + sign
    c. Right click on Indexes > New Index
    d. On the Properties enter a Name (for example, Idx1)
    e. Click on the Columns tab
    f. Select Column (for example, UPRN or USRN)
    g. Click Add and then OK

6.2.2 Converting coordinates to geometry

Next load the data into your new geometry column using the following SQL –
UPDATE public.abp_blpu_record SET geom = ST_GeomFromText('POINT(' || x_coordinate || ' ' || y_coordinate || ')', 27700).

This sets the geom column in the BLPU table to equal the values from the x_coordinate and y_coordinate columns, with the spatial reference defined as 27700.

Finally create a spatial index on the data using –
CREATE INDEX idx_blpu_geom ON public.abp_blpu_record USING gist(geom)

This adds the index name idx_blpu_geom to the same table on the geom column.

6.2.3 SQL statements

--**BLPU**

```
CREATE TABLE abp_blpu (
        RECORD_IDENTIFIER SMALLINT,
        CHANGE_TYPE CHARACTER VARYING(1),
        PRO_ORDER BIGINT,
        UPRN BIGINT,
        LOGICAL_STATUS SMALLINT,
        BLPU_STATE SMALLINT,
        BLPU_STATE_DATE DATE,
        PARENT_UPRN BIGINT,
        RPC SMALLINT,
        LOCAL_CUSTODIAN_CODE SMALLINT,
        START_DATE DATE,
        END_DATE DATE,
        LAST_UPDATE_DATE DATE,
        ENTRY_DATE DATE,
        POSTAL_ADDRESS CHARACTER VARYING(1),
        POSTCODE_LOCATOR CHARACTER VARYING(8),
        MULTI_OCC_COUNT SMALLINT,
        SHAPE CHARACTER VARYING(80)
);
```

```
--Classification
CREATE TABLE abp_classification (
        RECORD_IDENTIFIER SMALLINT,
        CHANGE_TYPE CHARACTER VARYING(1),
        PRO_ORDER BIGINT,
        UPRN BIGINT,
        CLASS_KEY CHARACTER VARYING(14),
        CLASSIFICATION_CODE CHARACTER VARYING(6),
        CLASS_SCHEME CHARACTER VARYING(60),
        SCHEME_VERSION DOUBLE PRECISION,
        START_DATE DATE,
        END_DATE DATE,
        LAST_UPDATE_DATE DATE,
        ENTRY_DATE DATE
);

--Application Cross reference table
CREATE TABLE abp_crossref (
        RECORD_IDENTIFIER SMALLINT,
        CHANGE_TYPE CHARACTER VARYING(1),
        PRO_ORDER BIGINT,
        UPRN BIGINT,
        XREF_KEY CHARACTER VARYING(14),
        CROSS_REFERENCE CHARACTER VARYING(50),
        VERSION SMALLINT,
        SOURCE CHARACTER VARYING(6),
        START_DATE DATE,
        END_DATE DATE,
        LAST_UPDATE_DATE DATE,
        ENTRY_DATE DATE
);
```

```
--Delivery Point Address
CREATE TABLE abp_delivery_point (
        RECORD_IDENTIFIER SMALLINT,
        CHANGE_TYPE CHARACTER VARYING(1),
        PRO_ORDER BIGINT,
        UPRN BIGINT,
        PARENT_ADDRESSABLE_UPRN BIGINT,
        RM_UDPRN BIGINT,
        ORGANISATION_NAME CHARACTER VARYING(60),
        DEPARTMENT_NAME CHARACTER VARYING(60),
        SUB_BUILDING_NAME CHARACTER VARYING(30),
        BUILDING_NAME CHARACTER VARYING(80),
        BUILDING_NUMBER SMALLINT,
        DEPENDENT_THOROUGHFARE_NAME CHARACTER VARYING(80),
        THOROUGHFARE_NAME CHARACTER VARYING(80),
        DOUBLE_DEPENDENT_LOCALITY CHARACTER VARYING(35),
        DEPENDENT_LOCALITY CHARACTER VARYING(35),
        POST_TOWN CHARACTER VARYING(30),
        POSTCODE CHARACTER VARYING(8),
        POSTCODE_TYPE CHARACTER VARYING(1),
        WELSH_DEPENDENT_THOROUGHFARE_NAME CHARACTER VARYING(80),
        WELSH_THOROUGHFARE_NAME CHARACTER VARYING(80),
        WELSH_DOUBLE_DEPENDENT_LOCALITY CHARACTER VARYING(35),
        WELSH_DEPENDENT_LOCALITY CHARACTER VARYING(35),
        WELSH_POST_TOWN CHARACTER VARYING(30),
        RM_PO_BOX_NUMBER CHARACTER VARYING(6),
        RM_PROCESS_DATE DATE,
        START_DATE DATE,
        END_DATE DATE,
        LAST_UPDATE_DATE DATE,
        ENTRY_DATE DATE
);
```

```
--LPI
CREATE TABLE abp_lpi (
        RECORD_IDENTIFIER SMALLINT,
        CHANGE_TYPE CHARACTER VARYING(1),
        PRO_ORDER BIGINT,
        UPRN BIGINT,
        LPI_KEY CHARACTER VARYING(14),
        LANGUAGE CHARACTER VARYING(3),
        LOGICAL_STATUS SMALLINT,
        START_DATE DATE,
        END_DATE DATE,
        LAST_UPDATE_DATE DATE,
        ENTRY_DATE DATE,
        SAO_START_NUMBER SMALLINT,
        SAO_START_SUFFIX CHARACTER VARYING(2),
        SAO_END_NUMBER SMALLINT,
        SAO_END_SUFFIX CHARACTER VARYING(2),
        SAO_TEXT CHARACTER VARYING(90),
        PAO_START_NUMBER SMALLINT,
        PAO_START_SUFFIX CHARACTER VARYING(2),
        PAO_END_NUMBER SMALLINT,
        PAO_END_SUFFIX CHARACTER VARYING(2),
        PAO_TEXT CHARACTER VARYING(90),
        USRN INTEGER,
        USRN_MATCH_INDICATOR CHARACTER VARYING(1),
        AREA_NAME CHARACTER VARYING(35),
        LEVEL CHARACTER VARYING(30),
        OFFICIAL_FLAG CHARACTER VARYING(1)
);

--Organisation
CREATE TABLE abp_organisation (
        RECORD_IDENTIFIER SMALLINT,
        CHANGE_TYPE CHARACTER VARYING(1),
        PRO_ORDER BIGINT,
        UPRN BIGINT,
        ORG_KEY CHARACTER VARYING(14),
        ORGANISATION CHARACTER VARYING(100),
        LEGAL_NAME CHARACTER VARYING(60),
        START_DATE DATE,
        END_DATE DATE,
        LAST_UPDATE_DATE DATE,
        ENTRY_DATE DATE
);
```

```
--Streets
CREATE TABLE abp_street (
        RECORD_IDENTIFIER SMALLINT,
        CHANGE_TYPE CHARACTER VARYING(1),
        PRO_ORDER BIGINT,
        USRN INTEGER,
        RECORD_TYPE SMALLINT,
        SWA_ORG_REF_NAMING SMALLINT,
        STATE SMALLINT,
        STATE_DATE DATE,
        STREET_SURFACE SMALLINT,
        STREET_CLASSIFICATION SMALLINT,
        VERSION SMALLINT,
        STREET_START_DATE DATE,
        STREET_END_DATE DATE,
        LAST_UPDATE_DATE DATE,
        RECORD_ENTRY_DATE DATE,
        STREET_START_X DOUBLE PRECISION,
        STREET_START_Y DOUBLE PRECISION,
        STREET_END_X DOUBLE PRECISION,
        STREET_END_Y DOUBLE PRECISION,
        STREET_TOLERANCE SMALLINT
);

--Street Descriptor
CREATE TABLE abp_street_(
        RECORD_IDENTIFIER SMALLINT,
        CHANGE_TYPE CHARACTER VARYING(1),
        PRO_ORDER BIGINT,
        USRN INTEGER,
        STREET_DESCRIPTION CHARACTER VARYING(110),
        LOCALITY_NAME CHARACTER VARYING(35),
        TOWN_NAME CHARACTER VARYING(30),
        ADMINISTRATIVE_AREA CHARACTER VARYING(30),
        LANGUAGE CHARACTER VARYING(3)
);
```

```
--Successor Records
CREATE TABLE abp_successor (
        RECORD_IDENTIFIER SMALLINT,
        CHANGE_TYPE CHARACTER VARYING(1),
        PRO_ORDER BIGINT,
        UPRN BIGINT,
        SUCC_KEY CHARACTER VARYING(14),
        START_DATE DATE,
        END_DATE DATE,
        LAST_UPDATE_DATE DATE,
        ENTRY_DATE DATE,
        SUCCESSOR BIGINT
);
```

6.3 Loading AddressBase products into Oracle

The following set of instructions assumes a basic knowledge of Oracle databases and SQLLDR which is the package used to load the CSV files into the database. Other options are available for loading data into an Oracle databases.

Using SQLLDR it is not necessary to join all the files in single files, but it can simply load the data as it comes off the disk provided that it is uncompressed. The following steps describe one method for loading a full supply of the data, and sections of code in italics indicate the sections that need to be altered to your specific file locations, drives and so on.

6.3.1 AddressBase CSV

1. Copy the data files from the disk to an appropriate location. It is worth noting that the files will need to be unzipped and therefore you will need in the region of 5 Gb of free space.

2. Once the data is copied the next stage is to uncompress the *.zip files to *.csv. This can be done using a package such as Winzip, or 7Zip. Please see Chapter 3.3 for more information.

3. Now that all the files are uncompressed it will make the latter stages quicker and easier to create a file list of all the CSV files to be loaded. This can be done using a batch file that writes all the files out to a text file:

```
dir *.csv /b/s >filelisting.txt
pause
```

This file will form the basis for loading the control file in a later step.

4. Create the tables using the SQL below. Prior to running the script the tablespace that the tables are going to reside needs to be altered from <*TablespaceName*> to the tablespace that is being worked in.

```
-- Drop Existing Table if exists
DROP TABLE AddressBase CASCADE CONSTRAINTS;

-- Create Table for AB_CSV
CREATE TABLE AddressBase (
        MI_PRINX    NUMBER (38,0),
        UPRN NUMBER,
        OS_ADDRESS_TOID VARCHAR2(20),
        RM_UDPRN NUMBER,
        ORGANISATION_NAME VARCHAR2(60),
        DEPARTMENT_NAME VARCHAR2(60),
        PO_BOX_NUMBER VARCHAR2(6),
        BUILDING_NAME VARCHAR2(50),
        SUB_BUILDING_NAME VARCHAR2(30),
        BUILDING_NUMBER NUMBER,
        DEPENDENT_THOROUGHFARE_NAME VARCHAR2(80),
        THOROUGHFARE_NAME VARCHAR2(80),
        POST_TOWN VARCHAR2(30),
        DOUBLE_DEPENDENT_LOCALITY VARCHAR2(35),
        DEPENDENT_LOCALITY VARCHAR2(35),
        POSTCODE VARCHAR2(8),
        POSTCODE_TYPE VARCHAR2(1),
        X_COORDINATE FLOAT,
        Y_COORDINATE FLOAT,
        RPC NUMBER,
        CHANGE_TYPE VARCHAR2(1),
        START_DATE DATE,
        LAST_UPDATE_DATE DATE,
        ENTRY_DATE DATE,
        CLASS VARCHAR2(1),
        PROCESS_DATE DATE,
        GEOMETRY  MDSYS.SDO_GEOMETRY
) TABLESPACE <tablespace name>;

DROP SEQUENCE AddressBase_S;
CREATE SEQUENCE AddressBase_S START WITH 1 INCREMENT BY 1;

-- Update USER_SDO_GEOM_METADATA
DELETE FROM USER_SDO_GEOM_METADATA WHERE TABLE_NAME = 'AddressBase';

INSERT INTO USER_SDO_GEOM_METADATA VALUES('AddressBase','GEOMETRY',
MDSYS.SDO_DIM_ARRAY(MDSYS.SDO_DIM_ELEMENT('X',0,750000,0.001),MDSYS.SDO_DIM_ELEMENT(
'Y',0,1350000,0.001)),81989);
```

5. Create the SQLLDR control file using the file listing created at step 3

```
OPTIONS ( BINDSIZE=20971520, READSIZE=20971520, ROWS=2500, ERRORS=10, SILENT = FEEDBACK )

LOAD DATA
CHARACTERSET UTF8
INFILE 'filelistingLine1.csv'
INFILE 'filelistingLine2.csv'

APPEND
        INTO TABLE AddressBase
        FIELDS TERMINATED BY ","
        OPTIONALLY ENCLOSED BY '"'
        TRAILING NULLCOLS
        (
        MI_PRINX    EXPRESSION "AddressBase_S.NEXTVAL",
        UPRN,
        OS_ADDRESS_TOID,
        RM_UDPRN,
        ORGANISATION_NAME,
        DEPARTMENT_NAME,
        PO_BOX_NUMBER,
        BUILDING_NAME,
        SUB_BUILDING_NAME,
        BUILDING_NUMBER,
        DEPENDENT_THOROUGHFARE_NAME,
        THOROUGHFARE_NAME,
        POST_TOWN,
        DOUBLE_DEPENDENT_LOCALITY,
        DEPENDENT_LOCALITY,
        POSTCODE,POSTCODE_TYPE,
        X_COORDINATE FLOAT EXTERNAL,
        Y_COORDINATE FLOAT EXTERNAL,
        RPC,
        CHANGE_TYPE,
        START_DATE DATE "YYYY-MM-DD",
        LAST_UPDATE_DATE DATE "YYYY-MM-DD",
        ENTRY_DATE DATE "YYYY-MM-DD",
        CLASS,
        PROCESS_DATE DATE "YYYY-MM-DD",
        GEOMETRY COLUMN OBJECT
                (
                SDO_GTYPE CONSTANT '2001',
                SDO_SRID CONSTANT '27700',
                SDO_POINT COLUMN OBJECT
                (
                        X ":X_COORDINATE",
                        Y ":Y_COORDINATE"
                )
                )
)
```

Once this file is created it can be called from a .bat file to run it on the box that holds the database rather than a remote machine. If you wish to run it from a remote machine contact your Oracle Administrator who will be able to advise on the best way to do this within your environment. The contents of the .bat file should be similar to:

@sqlldr <username>/<password>@<service name> control= <name of ctl file created previously>
Pause

6. Once the load has completed the relevant indexes need to be built. If the table name has been altered make sure that it is also altered in the below script.

```
--============================================
-- Creates the attribute index and PK on UPRN and MI+_PRINX
-- respectively, and then the spatial index
--============================================
CREATE UNIQUE INDEX AddressBase_MIPRINX_PK ON AddressBase (MI_PRINX);
CREATE INDEX AddressBase_UPRN_IDX ON AddressBase (UPRN);

Drop INDEX AddressBase_IXS;

CREATE INDEX AddressBase_IXS on AddressBase(GEOMETRY)
INDEXTYPE IS MDSYS.SPATIAL_INDEX
PARAMETERS ( ' INITIAL =8M , NEXT =8M , PCTINCREASE = 0,
LAYER_GTYPE=MULTIPOINT, tablespace=<tablespace name>' );
```

6.3.2 AddressBase Plus CSV

The process for loading AddressBase Plus is the same as for AddressBase, in that the data gets copied to a drive, unzipped, and a file list created. The tables are then created, the control file generated, and the data loaded. Finally the indexes are built.

1. Copy the data from a disk to an appropriate drive. For AddressBase Plus uncompressed will occupy in the region of 13 Gb of disk space.

2. Once the data is copied the next stage is to uncompress the *.zip files to *.csv. This can be done using a package such as Winzip, or 7Zip. Please see Chapter 3.3 for more information.

3. Now that all the files are uncompressed it will make the latter stages quicker and easier to create a file list of all the CSV files to be loaded. This can be done using a batch file that writes all the files out to a text file:

```
dir *.csv /b/s >filelisting.txt
pause
```

This file will form the basis for loading the control file in a later step.

4. Create the tables using the SQL below. Prior to running the script the tablespace that the tables are going to reside needs to be altered from <TablespaceName> to the tablespace in that is being worked in.

```
-- Drop Existing Table if exists
DROP TABLE AB_Plus CASCADE CONSTRAINTS;

-- Create Table for ABPL_CSV
CREATE TABLE AB_Plus (
        MI_PRINX    NUMBER (38,0),
        UPRN NUMBER,
        RM_UDPRN NUMBER,
        CHANGE_TYPE VARCHAR2(1),
        STATE NUMBER,
        STATE_DATE DATE,
        CLASS VARCHAR2(6),
        PARENT_UPRN NUMBER,
        X_COORDINATE FLOAT,
        Y_COORDINATE FLOAT,
        RPC NUMBER,
        LOCAL_CUSTODIAN_CODE NUMBER,
        START_DATE DATE,
        END_DATE DATE,
        LAST_UPDATE_DATE DATE,
        ENTRY_DATE DATE,
        ORGANISATION_NAME VARCHAR2(60),
        ORGANISATION VARCHAR2(100),
        DEPARTMENT_NAME VARCHAR2(60),
        SCOTTISH_DEPARTMENT_NAME VARCHAR2(60),
        BUILDING_NAME VARCHAR2(50),
        SUB_BUILDING_NAME VARCHAR2(30),
        SAO_START_NUMBER NUMBER,
        SAO_START_SUFFIX VARCHAR2(2),
        SAO_END_NUMBER NUMBER,
        SAO_END_SUFFIX VARCHAR2(2),
        SAO_TEXT VARCHAR2(90),
        ALT_LANGUAGE_SAO_TEXT VARCHAR2(90),
        PAO_START_NUMBER NUMBER,
        PAO_START_SUFFIX VARCHAR2(2),
        PAO_END_NUMBER NUMBER,
        PAO_END_SUFFIX VARCHAR2(2),
        PAO_TEXT VARCHAR2(90),
        ALT_LANGUAGE_PAO_TEXT VARCHAR2(90),
        USRN NUMBER,
```

```
        USRN_MATCH_INDICATOR VARCHAR2(1),
        AREA_NAME VARCHAR2(35),
        AddLEVEL VARCHAR2(30),
        OFFICIAL_FLAG VARCHAR2(1),
        OS_ADDRESS_TOID VARCHAR2(20),
        OS_ADDRESS_TOID_VERSION NUMBER,
        OS_ROADLINK_TOID VARCHAR2(20),
        OS_ROADLINK_TOID_VERSION NUMBER,
        OS_TOPO_TOID VARCHAR2(20),
        OS_TOPO_TOID_VERSION NUMBER,
        VOA_CT_RECORD NUMBER,
        VOA_NDR_RECORD NUMBER,
        STREET_DESCRIPTION VARCHAR2(101),
        ALT_LANGUAGE_STREET_DESCRIPTOR VARCHAR2(110),
        DEP_THOROUGHFARE_NAME VARCHAR2(80),
        THOROUGHFARE_NAME VARCHAR2(80),
        WELSH_DEP_THOROUGHFARE_NAME VARCHAR2(80),
        WELSH_THOROUGHFARE_NAME VARCHAR2(80),
        DOU_DEP_LOCALITY VARCHAR2(35),
        DEP_LOCALITY VARCHAR2(35),
        LOCALITY_NAME VARCHAR2(35),
        WELSH_DEP_LOCALITY VARCHAR2(35),
        WELSH_DOU_DEP_LOCALITY VARCHAR2(35),
        TOWN_NAME VARCHAR2(30),
        ADMINISTRATIVE_AREA VARCHAR2(30),
        POST_TOWN VARCHAR2(35),
        POSTCODE VARCHAR2(8),
        POSTCODE_LOCATOR VARCHAR2(8),
        POSTCODE_TYPE VARCHAR2(1),
        POSTAL_ADDRESS VARCHAR2(1),
        PO_BOX_NUMBER VARCHAR2(6),
        WARD_CODE VARCHAR2(9),
        PARISH_CODE         VARCHAR2(9),
        PROCESS_DATE DATE,
        MULTI_OCC_COUNT NUMBER,
        VOA_NDR_P_DESC_CODE VARCHAR2(5),
        VOA_NDR_SCAT_CODE VARCHAR2(4),
        ALT_LANGUAGE VARCHAR2(3),
        GEOMETRY  MDSYS.SDO_GEOMETRY
) TABLESPACE <tablespace name>;
```

```
DROP SEQUENCE AB_Plus_S;
CREATE SEQUENCE AB_Plus_S START WITH 1 INCREMENT BY 1;

-- Update USER_SDO_GEOM_METADATA
DELETE FROM USER_SDO_GEOM_METADATA WHERE TABLE_NAME = 'AB_Plus';

INSERT INTO USER_SDO_GEOM_METADATA VALUES('AB_Plus','GEOMETRY',
MDSYS.SDO_DIM_ARRAY(MDSYS.SDO_DIM_ELEMENT('X',0,750000,0.001),MDSYS.SDO_DIM_ELEMENT(
'X',0,1350000,0.001)),81989);
```

5. Create the SQLLDR control file using the file listing created at step 3.

```
OPTIONS ( BINDSIZE=20971520, READSIZE=20971520, ROWS=2500, ERRORS=10, SILENT = FEEDBACK )

LOAD DATA
CHARACTERSET UTF8
INFILE 'filelistingLine1.csv'
INFILE 'filelistingLine2.csv'

APPEND
        INTO TABLE AB_Plus
        FIELDS TERMINATED BY ","
        OPTIONALLY ENCLOSED BY '"'
        TRAILING NULLCOLS
        (
        MI_PRINX    EXPRESSION "AB_Plus_S.NEXTVAL",
        UPRN,
        RM_UDPRN,
        CHANGE_TYPE,
        STATE,
        STATE_DATE DATE "YYYY-MM-DD",
        CLASS,
        PARENT_UPRN,
        X_COORDINATE FLOAT EXTERNAL,
        Y_COORDINATE FLOAT EXTERNAL,
        RPC,
        LOCAL_CUSTODIAN_CODE,
        START_DATE DATE "YYYY-MM-DD",
        END_DATE DATE "YYYY-MM-DD",
        LAST_UPDATE_DATE DATE "YYYY-MM-DD",
        ENTRY_DATE DATE "YYYY-MM-DD",
        ORGANISATION_NAME,
```

ORGANISATION,
DEPARTMENT_NAME,
SCOTTISH_DEPARTMENT_NAME,
BUILDING_NAME,
SUB_BUILDING_NAME,
SAO_START_NUMBER,
SAO_START_SUFFIX,
SAO_END_NUMBER,
SAO_END_SUFFIX,
SAO_TEXT,
ALT_LANGUAGE_SAO_TEXT,
PAO_START_NUMBER,
PAO_START_SUFFIX,
PAO_END_NUMBER,
PAO_END_SUFFIX,
PAO_TEXT,
ALT_LANGUAGE_PAO_TEXT,
USRN,
USRN_MATCH_INDICATOR,
AREA_NAME,
AddLEVEL,
OFFICIAL_FLAG,
OS_ADDRESS_TOID,
OS_ADDRESS_TOID_VERSION,
OS_ROADLINK_TOID,
OS_ROADLINK_TOID_VERSION,
OS_TOPO_TOID,
OS_TOPO_TOID_VERSION,
VOA_CT_RECORD,
VOA_NDR_RECORD,
STREET_DESCRIPTION,
ALT_LANGUAGE_STREET_DESCRIPTOR,
DEP_THOROUGHFARE_NAME,
THOROUGHFARE_NAME,
WELSH_DEP_THOROUGHFARE_NAME,
WELSH_THOROUGHFARE_NAME,
DOU_DEP_LOCALITY,
DEP_LOCALITY,
LOCALITY_NAME,
WELSH_DEP_LOCALITY,
WELSH_DOU_DEP_LOCALITY,
TOWN_NAME,

```
        ADMINISTRATIVE_AREA,
        POST_TOWN,
        POSTCODE,
        POSTCODE_LOCATOR,
        POSTCODE_TYPE,
        POSTAL_ADDRESS,
        PO_BOX_NUMBER,
        WARD_CODE,
        PARISH_CODE,
        PROCESS_DATE DATE "YYYY-MM-DD",
        MULTI_OCC_COUNT,
        VOA_NDR_P_DESC_CODE,
        VOA_NDR_SCAT_CODE,
        ALT_LANGUAGE,
        GEOMETRY COLUMN OBJECT
                        (
                        SDO_GTYPE CONSTANT '2001',
                        SDO_SRID CONSTANT '81989',
                        SDO_POINT COLUMN OBJECT
                        (
                                X ":X_COORDINATE",
                                Y ":Y_COORDINATE"
                        )
                        )
)
```

The last section of the control file creates the geometry using the X and Y coordinate that is held within the data. Save the file with the extension *.ctl.

Once this file is created it can be called from a .bat file to run it on the box that holds the database rather than a remote machine. If you wish to run it from a remote machine contact your Oracle Administrator who will be able to advise on the best way to do this within your environment. The contents of the .bat file should be similar to:

@sqlldr <username>/<password>@<service name> control= <name of ctl file created previously>
Pause

6. Once the load has completed the relevant indexes need to be built. If the table name has been altered make sure that it is also altered in the below script.

```
CREATE UNIQUE INDEX AB_Plus_MIPRINX_PK ON AB_Plus (MI_PRINX);
CREATE INDEX AB_Plus_UPRN_IDX ON AB_Plus(UPRN);

Drop INDEX AB_Plus_IXS;

CREATE INDEX AB_Plus_IXS on AB_Plus(GEOMETRY)
INDEXTYPE IS MDSYS.SPATIAL_INDEX
PARAMETERS ( ' INITIAL =8M , NEXT =8M , PCTINCREASE = 0,
LAYER_GTYPE=MULTIPOINT, tablespace=<tablespace name>' );
```

6.3.3 AddressBase Premium CSV

The process for loading AddressBase Premium is the same as for AddressBase, in that the data gets copied to a drive, unzipped, and a file list created. The tables are then created, the control file generated, and the data loaded. Finally the indexes are built.

1. Copy the data files from the disk to an appropriate location. It is worth noting that the files will need to be unzipped and therefore you will need in the region of 30 Gb of free space.

2. Once the data is copied the next stage is to uncompress the *.zip files to *.csv. This can be done using a package such as Winzip, or 7Zip. Please see Chapter 3.3 for more information.

3. Now that all the files are uncompressed it will make the latter stages quicker and easier to create a file list of all the CSV files to be loaded. This can be done using a batch file that writes all the files out to a text file:

```
dir *.csv /b/s >filelisting.txt
pause
```

This file will form the basis for loading the control file in a later step.

1. To load the data there needs to be a table, or in the case of the premium data a number of tables created to hold the data. The SQL code needs to be run in a package like, SQL*PLUS, SQL Developer, or TOAD. Prior to running the script the tablespace that the tables are going to reside needs to be altered from <TablespaceName> to the tablespace in that is being worked in.

```
Drop TABLE ABPrem_Street;
Drop TABLE ABPrem_StreetDescript;
Drop TABLE ABPrem_BLPU;
Drop TABLE ABPrem_AppXRef;
Drop TABLE ABPrem_LPI;
Drop TABLE ABPrem_DeliveryPtAddress;
Drop TABLE ABPrem_Organisation;
Drop TABLE ABPrem_Metadata;
Drop TABLE ABPrem_Header;
Drop TABLE ABPrem_Sucessor;
Drop TABLE ABPrem_Trailer;
Drop TABLE ABPrem_Classification;

--============================================
-- Create the tables for the AddressBase Premium load
--============================================

--The 10 record table
Create TABLE ABPrem_Header
        (RECORD_IDENTIFIER NUMBER,
         CUSTODIAN_NAME VARCHAR2(40),
         LOCAL_CUSTODIAN_NAME NUMBER,
         PROCESS_DATE DATE,
         VOLUME_NUMBER NUMBER,
         ENTRY_DATE DATE,
         TIME_STAMP DATE,
         VERSION VARCHAR2(7),
         FILE_TYPE VARCHAR2(1)
         )TABLESPACE <TablespaceName>;
```

```
--The 11 record table (Streets)
Create TABLE ABPrem_Street
        (RECORD_IDENTIFIER NUMBER,
        CHANGE_TYPE VARCHAR2(1),
        PRO_ORDER NUMBER,
        USRN NUMBER,
        RECORD_TYPE NUMBER,
        SWA_ORG_REF_NAMING NUMBER,
        STATE NUMBER,
        STATE_DATE DATE,
        STREET_SURFACE NUMBER,
        STREET_CLASSIFICATION NUMBER,
        VERSION NUMBER,
        STREET_START_DATE DATE,
        STREET_END_DATE DATE,
        LAST_UPDATE_DATE DATE,
        RECORD_ENTRY_DATE DATE,
        STREET_START_X FLOAT,
        STREET_START_Y FLOAT,
        STREET_END_X FLOAT,
        STREET_END_Y FLOAT,
        STREET_TOLERANCE NUMBER,
        MI_PRINX NUMBER(38,0)
        )TABLESPACE <TablespaceName>;

--The 15 record table (Street Descriptor)
Create TABLE ABPrem_StreetDescript
        (RECORD_IDENTIFIER NUMBER,
        CHANGE_TYPE VARCHAR2(1),
        PRO_ORDER NUMBER,
        USRN NUMBER,
        /*street description is set to 110 in this script to allow for Welsh characters to be loaded into Oracle*/
        STREET_DESCRIPTION VARCHAR2(110),
        LOCALITY_NAME VARCHAR2(35),
        TOWN_NAME VARCHAR2(30),
        ADMINISTRATIVE_AREA VARCHAR2(30),
        LANGUAGE VARCHAR2(3)
        )TABLESPACE <TablespaceName>;
```

```
--The 21 record table (BLPU)
Create TABLE ABPrem_BLPU
        (RECORD_IDENTIFIER NUMBER,
        CHANGE_TYPE VARCHAR2(1),
        PRO_ORDER NUMBER,
        UPRN NUMBER,
        LOGICAL_STATUS NUMBER,
        BLPU_STATE NUMBER,
        BLPU_STATE_DATE DATE,
        PARENT_UPRN NUMBER,
        X_COORDINATE FLOAT,
        Y_COORDINATE FLOAT,
        RPC NUMBER,
        LOCAL_CUSTODIAN_CODE NUMBER,
        START_DATE DATE,
        END_DATE DATE,
        LAST_UPDATE_DATE DATE,
        ENTRY_DATE DATE,
        POSTAL_ADDRESS VARCHAR2(1),
        POSTCODE_LOCATOR VARCHAR2(8),
        MULTI_OCC_COUNT NUMBER,
        MI_PRINX NUMBER(38,0),
        GEOMETRY  MDSYS.SDO_GEOMETRY
        )TABLESPACE <TablespaceName>;

-- Update USER_SDO_GEOM_METADATA
DELETE FROM USER_SDO_GEOM_METADATA WHERE TABLE_NAME = 'ABPrem_BLPU';

INSERT INTO USER_SDO_GEOM_METADATA VALUES('ABPrem_BLPU','GEOMETRY',
MDSYS.SDO_DIM_ARRAY(MDSYS.SDO_DIM_ELEMENT('X',0,750000,0.001),MDSYS.SDO_DIM_ELEMENT(
'Y',0,1350000,0.001)),81989);

DROP SEQUENCE ABPrem_BLPU_S;
CREATE SEQUENCE ABPrem_BLPU_S START WITH 1 INCREMENT BY 1;
```

```
--The 23 record table (Application Cross Reference)
Create TABLE ABPrem_AppXRef
        (RECORD_IDENTIFIER NUMBER,
        CHANGE_TYPE VARCHAR2(1),
        PRO_ORDER NUMBER,
        UPRN NUMBER,
        XREF_KEY VARCHAR2(14),
        CROSS_REFERENCE VARCHAR2(50),
        VERSION NUMBER,
        SOURCE VARCHAR2(6),
        START_DATE DATE,
        END_DATE DATE,
        LAST_UPDATE_DATE DATE,
        ENTRY_DATE DATE
        )TABLESPACE <TablespaceName>;

--The 24 record table (LPI)
Create TABLE ABPrem_LPI
        (RECORD_IDENTIFIER NUMBER,
        CHANGE_TYPE VARCHAR2(1),
        PRO_ORDER NUMBER,
        UPRN NUMBER,
        LPI_KEY VARCHAR2(14),
        LANGUAGE VARCHAR2(3),
        LOGICAL_STATUS NUMBER,
        START_DATE DATE,
        END_DATE DATE,
        LAST_UPDATE_DATE DATE,
        ENTRY_DATE DATE,
        SAO_START_NUMBER NUMBER,
        SAO_START_SUFFIX VARCHAR2(2),
        SAO_END_NUMBER NUMBER,
        SAO_END_SUFFIX VARCHAR2(2),
        SAO_TEXT VARCHAR2(90),
        PAO_START_NUMBER NUMBER,
        PAO_START_SUFFIX VARCHAR2(2),
        PAO_END_NUMBER NUMBER,
        PAO_END_SUFFIX VARCHAR2(2),
        PAO_TEXT VARCHAR2(90),
        USRN NUMBER,
        USRN_M_INDICATOR VARCHAR2(1),
        AREA_NAME VARCHAR2(35),
        addLEVEL VARCHAR2(30),
        OFFICIAL_FLAG VARCHAR2(1)
        ) TABLESPACE <TablespaceName>;
```

```
--The 28 record table Delivery Point Address
Create TABLE ABPrem_DeliveryPtAddress
        (RECORD_IDENTIFIER NUMBER,
        CHANGE_TYPE VARCHAR2(1),
        PRO_ORDER NUMBER,
        UPRN NUMBER,
        PARENT_ADD_UPRN NUMBER,
        RM_UDPRN NUMBER,
        ORGANISATION_NAME VARCHAR2(60),
        DEPARTMENT_NAME VARCHAR2(60),
        SUB_BUILDING_NAME VARCHAR2(30),
        --BUILDING_NAME VARCHAR2(80),
        BUILDING_NAME VARCHAR2(50),
        BUILDING_NUMBER NUMBER,
        DEP_THOROUGHFARE_NAME VARCHAR2(80),
        THOROUGHFARE_NAME VARCHAR2(80),
        DOU_DEP_LOCALITY VARCHAR2(35),
        DEP_LOCALITY VARCHAR2(35),
        POST_TOWN VARCHAR2(30),
        POSTCODE VARCHAR2(8),
        POSTCODE_TYPE VARCHAR2(1),
        WELSH_DEP_THOROUGHFARE_NAME VARCHAR2(80),
        WELSH_THOROUGHFARE_NAME VARCHAR2(80),
        WELSH_DOU_DEP_LOCALITY VARCHAR2(35),
        WELSH_DEP_LOCALITY VARCHAR2(35),
        WELSH_POST_TOWN VARCHAR2(30),
        PO_BOX_NUMBER VARCHAR2(6),
        PROCESS_DATE DATE,
        START_DATE DATE,
        END_DATE DATE,
        LAST_UPDATE_DATE DATE,
        ENTRY_DATE DATE
 ) TABLESPACE <TablespaceName>;
```

```
-- The 29 record table
Create TABLE ABPrem_Metadata
        (RECORD_IDENTIFIER NUMBER,
        GAZ_NAME VARCHAR2(60),
        GAZ_SCOPE VARCHAR2(60),
        TER_OF_USE VARCHAR2(60),
        LINKED_DATA VARCHAR2(100),
        GAZ_OWNER VARCHAR2(15),
        NGAZ_FREQ VARCHAR2(1),
        CUSTODIAN_NAME VARCHAR2(40),
        CUSTODIAN_UPRN NUMBER,
        LOCAL_CUSTODIAN_CODE NUMBER,
        CO_ORD_SYSTEM VARCHAR2(40),
        CO_ORD_UNIT VARCHAR2(10),
        META_DATE DATE,
        CLASS_SCHEME VARCHAR2(60),
        GAZ_DATE DATE,
        LANGUAGE VARCHAR2(3),
        CHARACTER_SET VARCHAR2(30)
        ) TABLESPACE <TablespaceName>;

--The 30 record table
Create TABLE ABPrem_Sucessor
        (RECORD_IDENTIFIER NUMBER,
        CHANGE_TYPE VARCHAR2(1),
        PRO_ORDER NUMBER,
        UPRN NUMBER,
        SUCC_KEY VARCHAR2(14),
        START_DATE DATE,
        END_DATE DATE,
        LAST_UPDATE_DATE DATE,
        ENTRY_DATE DATE,
        SUCCESSOR NUMBER
        )TABLESPACE <TablespaceName>;
```

```
--The 31 record table
Create TABLE ABPrem_Organisation
        (RECORD_IDENTIFIER NUMBER,
        CHANGE_TYPE VARCHAR2(1),
        PRO_ORDER NUMBER,
        UPRN NUMBER,
        ORG_KEY VARCHAR2(14),
        ORGANISATION VARCHAR2(100),
        LEGAL_NAME VARCHAR2(60),
        START_DATE DATE,
        END_DATE DATE,
        LAST_UPDATE_DATE DATE,
        ENTRY_DATE DATE
        )TABLESPACE <TablespaceName>;

--The 32 record table
Create TABLE ABPrem_Classification
        (RECORD_IDENTIFIER NUMBER,
        CHANGE_TYPE VARCHAR2(1),
        PRO_ORDER NUMBER,
        UPRN NUMBER,
        CLASS_KEY VARCHAR2(14),
        CLASSIFICATION_CODE VARCHAR2(6),
        CLASS_SCHEME VARCHAR2(60),
        SCHEME_VERSION FLOAT,
        START_DATE DATE,
        END_DATE DATE,
        LAST_UPDATE_DATE DATE,
        ENTRY_DATE DATE
        )TABLESPACE <TablespaceName>;

--The 99 record table
Create TABLE ABPrem_Trailer
        (RECORD_IDENTIFIER NUMBER,
        NEXT_VOLUME_NUMBER NUMBER,
        RECORD_COUNT NUMBER,
        ENTRY_DATE DATE,
        TIME_STAMP DATE
        )TABLESPACE <TablespaceName>;
```

2. Now that the tables have been created the next stage is to create the SQLLDR control file. This is the file that describes where SQLLDR can find the relevant files, the structure it will find in those files, and what it needs to do with the records that are within the data.

```
OPTIONS ( BINDSIZE=20971520, READSIZE=20971520, ROWS=2500, ERRORS=9999, SILENT = FEEDBACK )

LOAD DATA
CHARACTERSET UTF8
INFILE 'filelistingLine1.csv'
INFILE 'filelistingLine2.csv'

APPEND
        INTO TABLE ABPrem_Header WHEN (01) = '10'
        FIELDS TERMINATED BY ","
        OPTIONALLY ENCLOSED BY '"'
        TRAILING NULLCOLS
        (RECORD_IDENTIFIER,
        CUSTODIAN_NAME,
        LOCAL_CUSTODIAN_NAME,
        PROCESS_DATE DATE "YYYY_MM_DD",
        VOLUME_NUMBER,
        ENTRY_DATE DATE "YYYY_MM_DD",
        TIME_STAMP DATE "HH24:Mi:SS",
        VERSION,
        FILE_TYPE
         )
```

```
INTO TABLE ABPrem_Street WHEN (01) = '11'
        FIELDS TERMINATED BY ","
        TRAILING NULLCOLS
        (RECORD_IDENTIFIER POSITION (1) INTEGER EXTERNAL,
        CHANGE_TYPE         CHAR ENCLOSED BY '"',
        PRO_ORDER INTEGER EXTERNAL,
        USRN INTEGER EXTERNAL,
        RECORD_TYPE INTEGER EXTERNAL,
        SWA_ORG_REF_NAMING,
        STATE,
        STATE_DATE Date "YYYY-MM-DD",
        STREET_SURFACE,
        STREET_CLASSIFICATION,
        VERSION,RECORD_ENTRY_DATE Date "YYYY-MM-DD",
        LAST_UPDATE_DATE DATE "YYYY-MM-DD",
        STREET_START_DATE Date "YYYY-MM-DD",
        STREET_END_DATE Date "YYYY-MM-DD",
        STREET_START_X FLOAT EXTERNAL,
        STREET_START_Y FLOAT EXTERNAL,
        STREET_END_X FLOAT EXTERNAL,
        STREET_END_Y FLOAT EXTERNAL,
        STREET_TOLERANCE,
        MI_PRINX EXPRESSION "ABPrem_Street_S.NEXTVAL"
        )

INTO TABLE ABPrem_StreetDescript WHEN (01) = '15'
        FIELDS TERMINATED BY ","
        OPTIONALLY ENCLOSED BY '"'
        TRAILING NULLCOLS
        (RECORD_IDENTIFIER POSITION (1) INTEGER EXTERNAL,
        CHANGE_TYPE,
        PRO_ORDER,
        USRN,
        STREET_DESCRIPTION,
        LOCALITY_NAME,
        TOWN_NAME,
        ADMINISTRATIVE_AREA,
        LANGUAGE
        )
```

```
INTO TABLE ABPrem_BLPU WHEN (01) = '21'
        FIELDS TERMINATED BY ","
        OPTIONALLY ENCLOSED BY '"'
        TRAILING NULLCOLS
        (RECORD_IDENTIFIER POSITION (1) INTEGER EXTERNAL,
         CHANGE_TYPE,
         PRO_ORDER,
         UPRN,
         LOGICAL_STATUS,
         BLPU_STATE,
         BLPU_STATE_DATE DATE "YYYY-MM-DD",
         PARENT_UPRN,
         X_COORDINATE FLOAT EXTERNAL,
         Y_COORDINATE FLOAT EXTERNAL,
         RPC,
         LOCAL_CUSTODIAN_CODE,
         START_DATE DATE "YYYY-MM-DD",
         END_DATE DATE "YYYY-MM-DD",
         LAST_UPDATE_DATE DATE "YYYY-MM-DD",
         ENTRY_DATE DATE "YYYY-MM-DD",
         POSTAL_ADDRESS,
         POSTCODE_LOCATOR,
        MULTI_OCC_COUNT,
        MI_PRINX    EXPRESSION "ABPrem_BLPU_S.NEXTVAL",
            GEOMETRY        COLUMN OBJECT
                    (
                    SDO_GTYPE CONSTANT '2001',
                    SDO_SRID CONSTANT '81989',
                    SDO_POINT COLUMN OBJECT
                    (
                            X ":X_COORDINATE",
                            Y ":Y_COORDINATE"
                    )
                    )
          )
```

```
INTO TABLE ABPrem_AppXRef WHEN (01) = '23'
        FIELDS TERMINATED BY ","
        OPTIONALLY ENCLOSED BY '"'
        TRAILING NULLCOLS
        (RECORD_IDENTIFIER POSITION (1) INTEGER EXTERNAL,
        CHANGE_TYPE,
        PRO_ORDER,
        UPRN,
        XREF_KEY,
        CROSS_REFERENCE,
        VERSION,
        SOURCE,
        START_DATE DATE "YYYY-MM-DD",
        END_DATE DATE "YYYY-MM-DD",
        LAST_UPDATE_DATE DATE "YYYY-MM-DD",
        ENTRY_DATE DATE "YYYY-MM-DD"
        )

INTO TABLE ABPrem_LPI WHEN (01) = '24'
        FIELDS TERMINATED BY ","
        OPTIONALLY ENCLOSED BY '"'
        TRAILING NULLCOLS
        (RECORD_IDENTIFIER POSITION (1) INTEGER EXTERNAL,
        CHANGE_TYPE,
        PRO_ORDER,
        UPRN,
        LPI_KEY,
        LANGUAGE,
        LOGICAL_STATUS,
        START_DATE DATE "YYYY-MM-DD",
        END_DATE DATE "YYYY-MM-DD",
        LAST_UPDATE_DATE DATE "YYYY-MM-DD",
        ENTRY_DATE DATE "YYYY-MM-DD",
        SAO_START_NUMBER,
        SAO_START_SUFFIX,
        SAO_END_NUMBER,
        SAO_END_SUFFIX,
        SAO_TEXT,
        PAO_START_NUMBER,
        PAO_START_SUFFIX,
        PAO_END_NUMBER,
        PAO_END_SUFFIX,
        PAO_TEXT,
        USRN,
        USRN_M_INDICATOR,
        AREA_NAME,
        addLEVEL,
        OFFICIAL_FLAG
        )
```

```
INTO TABLE ABPrem_DeliveryPtAddress WHEN (01) = '28'
        FIELDS TERMINATED BY ","
        OPTIONALLY ENCLOSED BY '"'
        TRAILING NULLCOLS
        (RECORD_IDENTIFIER POSITION (1) INTEGER EXTERNAL,
         CHANGE_TYPE,
         PRO_ORDER,
         UPRN,
         PARENT_ADD_UPRN,
         RM_UDPRN,
        ORGANISATION_NAME,
        DEPARTMENT_NAME,
        SUB_BUILDING_NAME,
        BUILDING_NAME,
        BUILDING_NUMBER,
        DEP_THOROUGHFARE_NAME,
        THOROUGHFARE_NAME,
        DOU_DEP_LOCALITY,
        DEP_LOCALITY,
        POST_TOWN,
        POSTCODE,
        POSTCODE_TYPE,
        WELSH_DEP_THOROUGHFARE_NAME,
        WELSH_THOROUGHFARE_NAME,
        WELSH_DOU_DEP_LOCALITY,
        WELSH_DEP_LOCALITY,
        WELSH_POST_TOWN,
        PO_BOX_NUMBER,
        PROCESS_DATE Date "YYYY-MM-DD",
        START_DATE Date "YYYY-MM-DD",
        END_DATE Date "YYYY-MM-DD",
        LAST_UPDATE_DATE Date "YYYY-MM-DD",
        ENTRY_DATE Date "YYYY-MM-DD"
        )
```

```
INTO TABLE ABPrem_Metadata WHEN (01) = '29'
        FIELDS TERMINATED BY ","
        OPTIONALLY ENCLOSED BY '"'
        TRAILING NULLCOLS
        (RECORD_IDENTIFIER POSITION (1) INTEGER EXTERNAL,
        GAZ_NAME,
        GAZ_SCOPE,
        TER_OF_USE,
        LINKED_DATA,
        GAZ_OWNER,
        NGAZ_FREQ,
        CUSTODIAN_NAME,
        CUSTODIAN_UPRN,
        LOCAL_CUSTODIAN_CODE,
        CO_ORD_SYSTEM,
        CO_ORD_UNIT,
        META_DATE DATE "YYYY-MM-DD",
        CLASS_SCHEME,
        GAZ_DATE DATE "YYYY-MM-DD",
        LANGUAGE,
        CHARACTER_SET
        )

INTO TABLE ABPrem_Sucessor WHEN (01) = '30'
        FIELDS TERMINATED BY ","
        OPTIONALLY ENCLOSED BY '"'
        TRAILING NULLCOLS
        (RECORD_IDENTIFIER POSITION (1) INTEGER EXTERNAL,
        CHANGE_TYPE,
        PRO_ORDER,
        UPRN,
        SUCC_KEY,
        START_DATE DATE "YYYY-MM-DD",
        END_DATE DATE "YYYY-MM-DD",
        LAST_UPDATE_DATE DATE "YYYY-MM-DD",
        ENTRY_DATE DATE "YYYY-MM-DD",
        SUCCESSOR
        )
```

```
INTO TABLE ABPrem_Organisation WHEN (01) = '31'
        FIELDS TERMINATED BY ","
        OPTIONALLY ENCLOSED BY '"'
        TRAILING NULLCOLS
        (RECORD_IDENTIFIER POSITION (1) INTEGER EXTERNAL,
        CHANGE_TYPE,
        PRO_ORDER,
        UPRN,
        ORG_KEY,
        ORGANISATION,
        LEGAL_NAME,
        START_DATE DATE "YYYY-MM-DD",
        END_DATE DATE "YYYY-MM-DD",
        LAST_UPDATE_DATE DATE "YYYY-MM-DD",
        ENTRY_DATE DATE "YYYY-MM-DD"
        )

INTO TABLE ABPrem_Classification WHEN (01) = '32'
        FIELDS TERMINATED BY ","
        OPTIONALLY ENCLOSED BY '"'
        TRAILING NULLCOLS
         (RECORD_IDENTIFIER POSITION (1) INTEGER EXTERNAL,
        CHANGE_TYPE,
        PRO_ORDER,
        UPRN,
        CLASS_KEY,
        CLASSIFICATION_CODE,
        CLASS_SCHEME,
        SCHEME_VERSION FLOAT EXTERNAL,
        START_DATE DATE "YYYY-MM-DD",
        END_DATE DATE "YYYY-MM-DD",
        LAST_UPDATE_DATE DATE "YYYY-MM-DD",
        ENTRY_DATE DATE "YYYY-MM-DD"
        )

INTO TABLE ABPrem_Trailer WHEN (01) = '99'
        FIELDS TERMINATED BY ","
        OPTIONALLY ENCLOSED BY '"'
        TRAILING NULLCOLS
        (RECORD_IDENTIFIER POSITION (1) INTEGER EXTERNAL,
        NEXT_VOLUME_NUMBER,
        RECORD_COUNT,
        ENTRY_DATE DATE "YYYY-MM-DD",
        TIME_STAMP DATE "HH24:MI:SS"
         )
```

Populate the INFILE lines with the file listing from that was created in step 3, with one INFILE command for each file. This tells the process to open each of the files and carry out the other tasks listed below it.

The rest of the file tells the tool how to interpret the files that it is reading in. The INTO statement at the top of each of the tables tests the first column (01) of the row in the file that it is looking at, and if it meets the criteria the structure of the table that the line is to be loaded into is described below it. Save the completed file with the extension *.ctl.

Once this file is created it can be called from a .bat file to run it on the box that holds the database rather than a remote machine. If you wish to run it from a remote machine contact your Oracle Administrator who will be able to advise on the best way to do this within your environment. The contents of the .bat file should be similar to:

@sqlldr <username>/<password>@<service name> control= <name of ctl file created previously>
Pause

When the .bat file is run the data is loaded. Any records that do not meet the expected structure, or errors with the load are recorded in the *.bad, and *.log files respectively that are written out to the same drive location as the control file that is being used in to load the data. It is strongly recommended that the log file is checked once the load is completed to check that all of the data has loaded correctly before continuing.

3. Post load the indexes need to be built in order to be able to carry out spatial queries and other queries where the relationship between the tables need to be built within the query. For example, in order to return all the Delivery Point Addresses within a county, there needs to be a spatial index on the BLPU table which contains the geometry, as well as the UPRN in both the BLPU and Delivery Point Address table. The following SQL statements build the required indexes on each of the tables. If the table names have been changed from those contained within the scripts above, make sure to replicate that in the index scripts below.

```
--BLPU Indexes
DROP INDEX ABPrem_BLPU_MIPRINX_PK;
DROP INDEX ABPrem_BLPU_UPRN_IDX;
CREATE UNIQUE INDEX ABPrem_BLPU_MIPRINX_PK ON ABPrem_BLPU (MI_PRINX);
CREATE INDEX ABPrem_BLPU_UPRN_IDX ON ABPrem_BLPU (UPRN);

Drop INDEX ABPrem_BLPU_IXS;

CREATE INDEX ABPrem_BLPU_IXS on ABPrem_BLPU(GEOMETRY)
INDEXTYPE IS MDSYS.SPATIAL_INDEX
PARAMETERS ( ' INITIAL =8M , NEXT =8M , PCTINCREASE = 0,
LAYER_GTYPE=MULTIPOINT, tablespace=<tablespace name>' );

-- APP X REF Indexes
DROP INDEX ABPrem_AppXRef_XRef_IDX;
CREATE INDEX ABPrem_AppXRef_XRef_IDX ON ABPrem_APPXREF (UPRN);
```

```
-- Classification Indexes
DROP INDEX ABPrem_Classification_UPRN_IDX;
CREATE INDEX ABPrem_Classification_UPRN_IDX ON ABPrem_CLASSIFICATION (UPRN);

-- Organisation Indexes
DROP INDEX ABPrem_Organisation_UPRN_IDX;
CREATE INDEX ABPrem_Organisation_UPRN_IDX ON ABPrem_ORGANISATION(UPRN);

-- Delivery Point Address Indexes
DROP INDEX ABPrem_DP_PC_IDX;
DROP INDEX ABPrem_DP_UPRN_IDX;
CREATE INDEX ABPrem_DP_PC_IDX ON ABPrem_DeliveryPtAddress (POSTCODE);
CREATE INDEX ABPrem_DP_UPRN_IDX ON ABPrem_DeliveryPtAddress (UPRN);

--LPI Indexes
DROP INDEX ABPrem_LPI_UPRN_IDX;
DROP INDEX ABPrem_LPI_USRN_IDX;
CREATE INDEX ABPrem_LPI_UPRN_IDX ON ABPrem_LPI (UPRN);
CREATE INDEX ABPrem_LPI_USRN_IDX ON ABPrem_LPI (USRN);

--Street Indexes
DROP INDEX ABPrem_Street_USRN_IDX;
DROP INDEX ABPrem_STREET_MIPRINX_PK;
CREATE INDEX ABPrem_Street_USRN_IDX ON ABPrem_STREET(USRN);
CREATE UNIQUE INDEX ABPrem_STREET_MIPRINX_PK ON ABPrem_STREET(MI_PRINX);

--StreetDescriptor Indexes
DROP INDEX ABPrem_StreetDescript_USRN_IDX;
CREATE INDEX ABPrem_StreetDescript_USRN_IDX ON ABPrem_STREETDESCRIPT (USRN);
```

Once the indexes are complete the data loading process is complete and the data is ready to use. This process has included the MI_PRINX fields and indexes required to register the data with esri ArcSDE and use with an esri environment.

7 How do I load the GML data?

GML is an XML dialect which can be used to model geographic features. It was designed by the OGC as a means for people to share information regardless of the particular applications or technology that they use. In the first instance, GML was used to overcome the differences between different GIS applications by providing a neutral file format as an alternative to proprietary formats such as Esri SHAPE or MapInfo TAB files and so on. However, because it is independent of applications there is no reason to assume that data exchanged in GML is being exchanged between GIS – it can be moved between databases or other types of application. GML therefore has a wider application than just GIS data transfer.

GML data can be viewed using software such as Snowflake GML Viewer
http://www.snowflakesoftware.co.uk/products/gmlviewer/ or FME Data Inspector/Universal Viewer
http://docs.safe.com/fme/html/FME_UniversalViewer/welcome_to_the_fme_universal_viewer.htm

GML data can be loaded into a database using software such as Snowflake Go Loader
http://www.snowflakesoftware.com/products/goloader/schemas/ordnancesurvey/address_base.htm

As yet, we are unaware of any other solutions that are available from Licensed Partners to load the GML data.

If customers wish to use AddressBase products supplied in GML format, they will need to develop code that will enable the data to be used effectively.

8 How do I create a single line or multi-line address?

8.1 Introduction

The AddressBase® products contain a variety of data fields which allow a user to construct, for a given addressable object, different forms of an address dependent on how the address is to be used.

There are two types of address contained in the AddressBase products:

- Delivery Point Address
- Geographic Address

These two address types come from different sources and are matched together by GeoPlace.

The Delivery Point Address is sourced from Royal Mail's PAF (Postcode Address File) which is a non-geocoded list of addresses. These addresses are used primarily as a 'mailing list' for postal purposes.

Geographic Addresses are maintained by contributing Local Authorities. The structure of a geographic address is based on the British Standard BS7666. These addresses are used to provide an accurate geographic locator for an object to aid for example, service delivery, asset management, or command and control operations. They also represent the legal form of addresses as created under street naming and numbering legislation.

High level data model

The AddressBase Premium data model accommodates both the Delivery Point Address and the geographic address by linking them using the UPRN as the key.

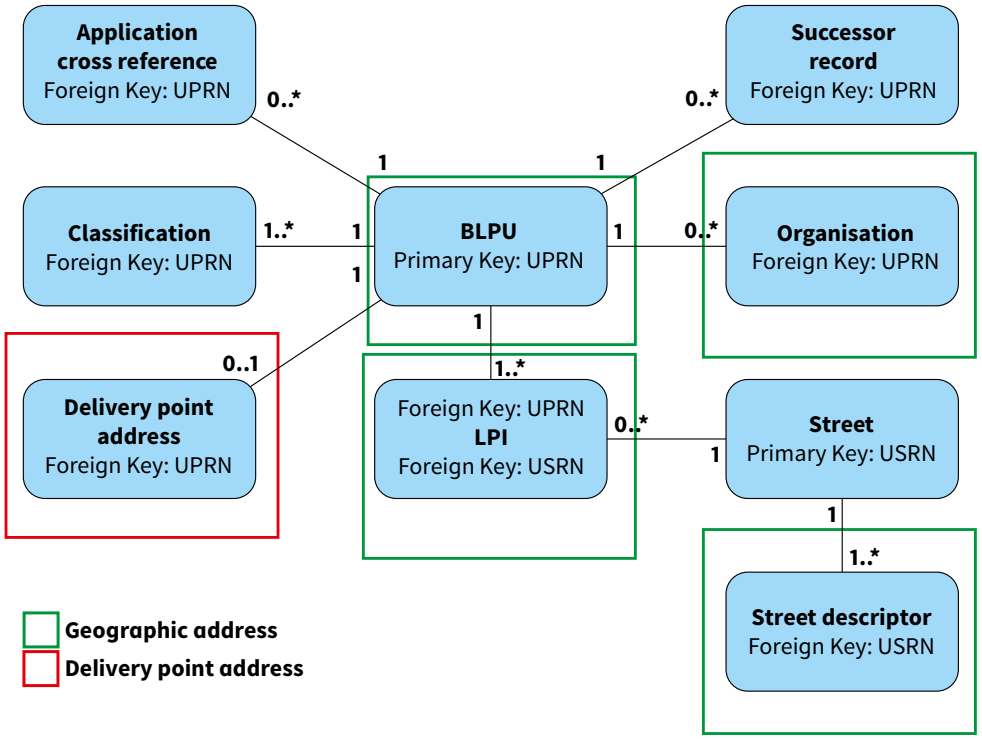


Figure 5: AddressBase Premium high level data model with Address 'types' highlighted

It is important to note the cardinality differences that the geographic and Delivery Point Address components have with the Basic Land and Property Unit (BLPU):

- The relationship between the DPA (Delivery Point Address) and the BLPU is 0..1 – 1

  This means that the DPA is an optional component, therefore a DPA address will only be created when it has been matched to the geographic address. Moreover, only 1 DPA can be matched to a BLPU.

- The relationship between the Land and Property Identifier (LPI) and the BLPU is 1..* – 1

  This means that the LPI component is mandatory; therefore at least 1 LPI must exist for each BLPU. Moreover, there can be more than 1 LPI linked to a single BLPU.

Together, these differences mean that there are more geographic addresses in the product than there are Delivery Point Addresses, because:

a) Not every BLPU has a Delivery Point (postal) Address – only those that have been matched to the Royal Mail PAF database.
b) A single BLPU can have only one Delivery Point Address.
c) A single BLPU can have more than one geographic address (because alternative and historical addresses are available in AddressBase Premium).

8.2 Background

A common requirement for customers using the AddressBase products is to build a single address label from core address elements.

There are two types of address label. The simplest is a full address on a single line with different elements separated by commas and spaces. This type of label is suited for displaying a full address within a tabular display, such as within an on-screen data grid or spread sheet, or where a single line printed address is most appropriate (such as within the text, header or footer of a letter):

ROSE COTTAGE, 5 MAIN STREET, ADDRESSVILLE, LONDON, SE99 9EX

The other type of formatted address is a multi-line address label. These are most often used on envelopes or at the tops of letters, where different parts of an address are separated onto different lines:

ROSE COTTAGE 5 MAIN STREET ADDRESSVILLE LONDON SE99 9EX

This document outlines a methodology for structuring and layering a single address label using AddressBase Premium. The methodology is largely applicable to AddressBase Plus for building geographic addresses (with the exception that all the fields are within a single table, so table joins are not necessary, and some of the status filters will not be present, such as "logical_status"). It is also applicable to AddressBase when building address labels for Delivery Point Addresses.

This guide provides the suggested logic to build both delivery point and geographic addresses.

The rules in this guidance are suggestions only and can be used for visual display of full addresses. It is strongly recommended that address components are stored in the relational format in which they are provided in order to allow maximum flexibility of use and derived value.

8.3 Delivery Point Address (PAF Address)

Description:

A Delivery Point Address contains information sourced from Royal Mail (PAF). Stringent rules are used to match these addresses to the geographic address and assign a common UPRN to link addresses from the two addressing sources together in the data model.

To construct a single address label based purely on the Royal Mail PAF address fields, the following attributes can be used to build a Delivery Point Address label:

Table 1 – Delivery Point Address Components

Delivery Point Address Component	Type
DEPARTMENT_NAME	Character
ORGANISATION_NAME	Character
SUB_BUILDING_NAME	Character
BUILDING_NAME	Character
BUILDING_NUMBER	Integer
PO_BOX_NUMBER	Integer
DEPENDENT_THOROUGHFARE_NAME *OR* WELSH_DEPENDENT_THOROUGHFARE_NAME	Character
THOROUGHFARE_NAME *OR* WELSH_THOROUGHFARE_NAME	Character
DOUBLE_DEPENDENT_LOCALITY *OR* WELSH_ DOUBLE_DEPENDENT_LOCALITY	Character
DEPENDENT_LOCALITY *OR* WELSH_ DEPENDENT_LOCALITY	Character
POST_TOWN *OR* WELSH_POST_TOWN	Character
POSTCODE	Character

These address components are listed in the correct order in which they should appear on an address label.

It should be noted that most of the PAF fields are optional and may contain null values (or zero, in the cases of "building number" and "po box number"). In these cases, those fields should be omitted. Where an '*OR*' exists, there may be a choice between English and Welsh languages (although if the preferred language field is empty, the other should be used if it contains a value).

The following (entirely fictional) example shows all of the PAF fields filled in (apart from the PO Box number), and how they should be ordered in a single address label:

Delivery Point Address Component	Example
DEPARTMENT_NAME	CUSTOMER SERVICE DEPARTMENT
ORGANISATION_NAME	JW SIMPSON LTD.
SUB_BUILDING_NAME	UNIT 3
BUILDING_NAME	THE OLD FORGE
BUILDING_NUMBER	7
PO_BOX_NUMBER	
DEPENDENT_THOROUGHFARE_NAME	RICHMOND TERRACE
THOROUGHFARE_NAME	MAIN STREET
DOUBLE_DEPENDENT_LOCALITY	HOOK
DEPENDENT_LOCALITY	WARSASH
POST_TOWN	SOUTHAMPTON
POSTCODE	SO99 9ZZ

In cases where a PO BOX number is present, it will only be described in the data as an integer. In order to properly format these addresses when generating an address label, these integers should be prefixed with the text 'PO BOX', as shown in the following example:

Delivery Point Address Component	Data Content	Formatted output
ORGANISATION_NAME	'JWS CONSULTING'	JWS CONSULTING
PO_BOX_NUMBER	5422	PO BOX 5422
THOROUGHFARE_NAME	'HIGH STREET'	HIGH STREET
POST_TOWN	'SPRINGFIELD'	SPRINGFIELD
POSTCODE	'SP77 0SF'	SP77 0SF

Where null or empty string values exist (for character fields) or zeros or nulls (for integer fields), those fields should be entirely omitted from the output; however, the order in which the fields should be concatenated always remains the same:

Delivery Point Address Component	Data content	Formatted output
DEPARTMENT_NAME	null	
ORGANISATION_NAME	'TM MOTORS'	TM MOTORS
SUB_BUILDING_NAME	null	
BUILDING_NAME	'THE OLD BARN'	THE OLD BARN
BUILDING_NUMBER	0 (or null)	
PO_BOX_NUMBER	0 (or null)	
DEPENDENT_THOROUGHFARE_NAME	null	
THOROUGHFARE_NAME	'HORSHAM LANE'	HORSHAM LANE
DOUBLE_DEPENDENT_LOCALITY	null	
DEPENDENT_LOCALITY	null	
POST_TOWN	'HORSHAM'	HORSHAM
POSTCODE	'RH12 1EQ'	RH12 1EQ

8.3.1 Building a single line Delivery Point Address

Building a single line, formatted address for a delivery point is relatively straightforward. All the fields should be checked in the order described above in Table 1, and those that have values should be concatenated together into a single line. Generally, address components should be separated by a comma followed by a single space (", "), although sometimes only a space is used between a building number and a thoroughfare name; this is down to personal preference.

Example SQL logic (single line address label):

```
SELECT
uprn,
(
 CASE WHEN department_name IS NOT NULL THEN department_name || ', ' ELSE '' END
 || CASE WHEN organisation_name IS NOT NULL THEN organisation_name || ', ' ELSE '' END
 || CASE WHEN sub_building_name IS NOT NULL THEN sub_building_name || ', ' ELSE '' END
 || CASE WHEN building_name IS NOT NULL THEN building_name || ', ' ELSE '' END
 || CASE WHEN building_number IS NOT NULL THEN building_number || ' ' ELSE '' END
 || CASE WHEN po_box_number IS NOT NULL THEN 'PO BOX ' || po_box_number || ', ' ELSE '' END
 || CASE WHEN dep_thoroughfare_name IS NOT NULL THEN dep_thoroughfare_name || ', ' ELSE '' END
 || CASE WHEN thoroughfare_name IS NOT NULL THEN thoroughfare_name || ', ' ELSE '' END
 || CASE WHEN dou_dep_locality IS NOT NULL THEN dou_dep_locality || ', ' ELSE '' END
 || CASE WHEN dep_locality IS NOT NULL THEN dep_locality || ', ' ELSE '' END
 || CASE WHEN post_town IS NOT NULL THEN post_town || ', ' ELSE '' END
 || postcode
) AS dpa_single_address_label
FROM addressbase_premium.delivery_point;
```

Notes:

1. The SQL operator for concatenating text is a double pipe ("||").
2. CASE blocks have been used to test each of the fields for null values before concatenating its contents (along with a suitable separator – either ", " or " ").
3. This example formats an English version of an address. In order to obtain a Welsh address, the thoroughfare_name, locality and post town fields can be substituted (for example, "welsh_thoroughfare_name" instead of "thoroughfare_name", and so on…). SQL "Coalesce" statements could be added in this case to substitute the English values when a Welsh language field value is null.
4. The field names and table names used are illustrative and may vary between databases.
5. Depending on the database schema and data loading method used, it may be necessary to test some fields for empty strings ('') or zero values (for integer fields) instead of, or as well as testing for NULLs.

8.3.2 Building a multi-line Delivery Point Address

Splitting a Delivery Point Address into multiple lines is more complicated. There are several rules to consider in order to avoid having very short lines (for example, just a building number) or very long lines within the formatted address. A summary of these rules is as follows:

- Generally, if there is a building number, it should appear on the same line as the thoroughfare (or dependent thoroughfare) name. If there is no thoroughfare name information, it should appear on the same line as the first locality name.
- In cases where building numbers have been placed in the building name field due to the presence of a letter suffix (for example, "11A") or a number range separator (for example, "3-5"), these should be detected and placed on the same line as the thoroughfare name in the same way as a building number (or on the first locality line, if no thoroughfare name is present). In most other cases, the building name, if present, should appear on a separate line above the thoroughfare_name (or dependent thoroughfare) name (or locality line if no thoroughfare name is present).

- Similar tests should be applied to the sub building name field: if this field contains a number, a number with a suffix, or a numeric range, it should precede the building name on the same line. In most other cases, it should appear on a separate line above the building name.

For much more detailed information on how to build a multi-line Delivery Point Address please consult pages 26–37 of the **PAF Programmers Guide**.

8.4 Geographic Address (Local Authority Address)

Description:

The structure of a geographic address is based on the British Standard BS7666 and is split into a number of components. This means that in order to construct a complete address label, for example, on an envelope, database form or GIS display, the components need to be constructed according to a set of rules.

This chapter contains principles that should be used when rendering geographic addresses from the AddressBase products in order to generate unambiguous and fit for purpose address labels.

Within the AddressBase products the core property level address information is stored within the Land and Property Identifier (LPI), in particular the Primary Addressable Object (PAO) and Secondary Addressable Object (SAO) fields. The additional attribution required to build a full address label is maintained in the BLPU (postcode_locator), ORGANISATION (organisation) and STREET_DESCRIPTOR (street_description, locality_name, town_name, administrative_area) tables.

For a full description of PAOs and SAOs, and the complete set of AddressBase fields, please refer to the relevant **AddressBase products technical specification** document.

Constructing a single address label from the Geographic Address Fields

To construct a single address label based purely on the BS7666 address fields, the following attributes should be used to build a geographic address label:

Table 2 – Geographic Address Components

Table	Geographic Address Component
Organisation	ORGANISATION
LPI	SAO_TEXT
LPI	SAO_START_NUMBER
LPI	SAO_START_SUFFIX
LPI	SAO_END_NUMBER
LPI	SAO_END_SUFFIX
LPI	PAO_TEXT
LPI	PAO_START_NUMBER
LPI	PAO_START_SUFFIX
LPI	PAO_END_NUMBER
LPI	PAO_END_SUFFIX
Street Descriptor	STREET_DESCRIPTION
Street Descriptor	LOCALITY_NAME
Street Descriptor	TOWN_NAME
Street Descriptor	ADMINISTRATIVE_AREA*
BLPU	POSTCODE_LOCATOR

** ADMINISTRATIVE_AREA is optional because it is common for this field to be the same as the TOWN_NAME. Sometimes however, this field will help users construct a more complete address.*

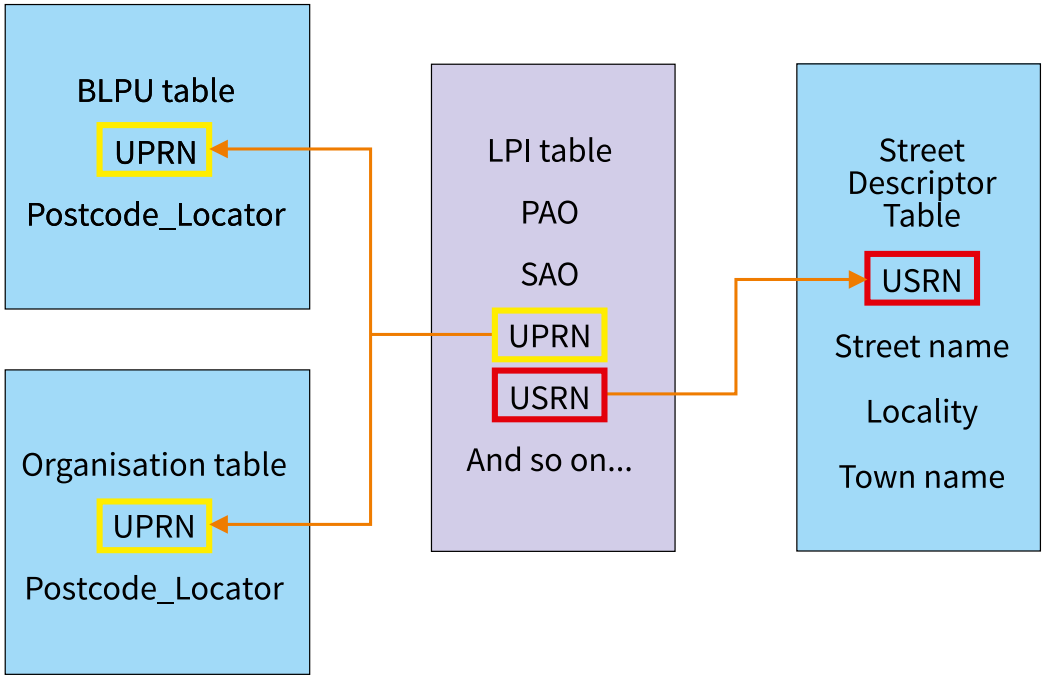
These address components are listed in the correct order in which they should appear on an address label.

8.4.1 Linking Address Components

The Land and Property Identifier (LPI) table includes the primary and secondary addressable object fields; however, in order to obtain the rest of the address, it is necessary to join this to the Street Descriptor table to pick up the street name, locality and town information (using the Unique Street Reference Number – USRN as the key), and also to the Organisation and BLPU tables (using the UPRN as the key) to pick up the organisation names and postcodes respectively.

The following diagram shows the links that need to be made in order to build a full geographic address from the different BS7666 components in AddressBase Premium:

Figure 6: Table join logic for geographic address label



Using the LPI table as a starting point, the remaining address components can be picked up using table joins to the other tables on UPRNs and USRNs. Note that there can be more than one LPI for each UPRN, so if only one address is required per BLPU, the LPI with logical_status = 1 (approved) should be selected (there can be only one approved LPI per BLPU).

8.4.2 Rendering SAOs and PAOs

When building a single address label it may be necessary to concatenate the various SAO fields and PAO fields together (respectively). These fields contain any property names, numbers, number ranges or suffixes that apply to an address.

A PAO number/range string should be constructed from the *pao_start_number, pao_start_suffix, pao_end_number and pao_end_suffix* fields as illustrated in the following table:

Attribute	Example 1	Example 2	Example 3	Example 4
PAO_START_NUMBER	1	1	1	1
PAO_START_SUFFIX		A		A
PAO_END_NUMBER			5	5
PAO_END_SUFFIX				C
Rendered PAO range	1	1A	1-5	1A-5C

Similarly, a SAO number/range string should be constructed from the *sao_start_number, sao_start_suffix, sao_end_number and sao_end_suffix* fields as illustrated in the following table:

Attribute	Example 1	Example 2	Example 3	Example 4
SAO_START_NUMBER	1	1	1	1
SAO_START_SUFFIX		A		A
SAO_END_NUMBER			5	5
SAO_END_SUFFIX				C
Rendered SAO range	1	1A	1-5	1A-5C

In addition to the numeric range fields described above, there are also pao_text and sao_text fields. These fields may be populated instead of, or as well as the numeric range fields. In both cases, if both text and a numeric range string are present, the text should appear before the numeric range in any formatted address:

Attribute	Example 1	Example 2	Example 3	Example 4
PAO (number string)	1	1A	1A	
PAO (text)			Rose Cottage	Rose Cottage
Rendered PAO (showing street name location)	1 <street>	1A <street>	Rose Cottage, 1A <street>	Rose Cottage, <street>

Note: for primary addressable objects (PAOs), there will always be either a text entry, or a numeric/range entry, or both. This is not the case for secondary addressable objects (SAOs), which may be entirely absent for a given address.

8.4.3 Street description, town, locality and administrative area names

These fields are all found within the "Street Descriptor" table in AddressBase Premium. The street description, town name (where street type is 1 or 2) and administrative area names are always present, while the locality name may be empty.

The ADMINISTRATIVE_AREA field always contains a value; however, this value will not always enhance an address, but in some cases it will. In particular, a check should be made to ensure that it is not the same as the value in the TOWN_NAME field, as is often the case.

For example, in the following case, the administrative area adds value to the address:

Administrative area not included

GEO_SINGLE_ADDRESS_LABEL
34, CROW LANE, RAMSBOTTOM, BL0 9BR

Vs.

Administrative area included (BURY)

GEO_SINGLE_ADDRESS_LABEL
34, CROW LANE, RAMSBOTTOM, BURY, BL0 9BR

In other cases, the administrative area name will simply contain the local authority name, which would not traditionally form part of a single or multi-line address, but can be included to add additional information to an address label; its inclusion is largely down to business requirements and/or personal preference however it may also be useful to 'de-duplicate' some geographic addresses.

The following (entirely fictional) example shows all of the BS7666 geographic address fields filled in, and how they should be ordered in a single address label:

Table 3 – Geographic Address Formatting

Geographic Address Component	Example
ORGANISATION	JW SIMPSON LTD
SAO_TEXT	THE ANNEXE
SAO (number/range string)*	1A
PAO_TEXT	THE OLD MILL
PAO (number/range string)*	7–9
STREET_DESCRIPTION	MAIN STREET
LOCALITY_NAME	HOOK
TOWN_NAME	WARSASH
ADMINISTRATIVE_AREA	SOUTHAMPTON
POSTCODE_LOCATOR	SO99 9ZZ

* The number/range strings are built from the relevant PAO/SAO start_number, start_suffix, end_number and end_suffix fields, as described above, and formatted as character strings.

Where an administrative area matches the town name, it should always be omitted:

Delivery Point Address Component	Data content	Formatted output
PAO_TEXT	"HIGHBURY HOUSE"	HIGHBURY HOUSE
STREET_DESCRIPTION	"HIGH STREET"	HIGH STREET
TOWN_NAME	"SOUTHAMPTON"	SOUTHAMPTON
ADMINISTRATIVE_AREA	"SOUTHAMPTON"	
POSTCODE_LOCATOR	"SO77 0SF"	SO77 0SF

Where null or empty string values exist (for character fields) or zeros or nulls (for integer fields), those fields should be entirely omitted from the output; however, the order in which the fields should be concatenated always remains the same:

Delivery Point Address Component	Data content	Formatted output
ORGANISATION	"TM MOTORS"	TM MOTORS
SAO_TEXT	null	
SAO (number/range string)*	null	
PAO_TEXT	"THE OLD BARN"	THE OLD BARN
PAO (number/range string)*	"1"	1
STREET_DESCRIPTION	"HORSHAM LANE"	HORSHAM LANE
LOCALITY_NAME	null	
TOWN_NAME	"HORSHAM"	HORSHAM
ADMINISTRATIVE_AREA	"HORSHAM"	* Duplicate name omitted
POSTCODE_LOCATOR	"RH12 1EQ"	"RH12 1EQ"

Building a single line geographic address

Building a single line, formatted address for a delivery point is slightly more complicated than for a Delivery Point Address due to the need to pre-format the SAO and PAO number/range strings, and the need to join tables together (in AddressBase Premium). However, once this is done, the process is largely the same as before: the calculated fields should be checked in the order described above in Table 3, and those that have values should be concatenated together into a single line. Generally, address components should be separated by a comma followed by a single space (", "), although sometimes only a space is used between a PAO number/range string and a street description; this is down to personal preference.

Example SQL logic (single line address label):

```
SELECT
uprn,
(
 CASE WHEN o.organisation IS NOT NULL THEN o.organisation || ', ' ELSE '' END
 || CASE WHEN l.sao_text IS NOT NULL THEN l.sao_text || ', ' ELSE '' END
 || CASE WHEN GetAONumberString(l.sao_start_number, l.sao_start_suffix, l.sao_end_number, l.sao_end_suffix) <> '' THEN
        GetAONumberString(l.sao_start_number, l.sao_start_suffix, l.sao_end_number, l.sao_end_suffix) || ' '
 ELSE '' END
 || CASE WHEN l.pao_text IS NOT NULL THEN l.pao_text || ', ' ELSE '' END
 || CASE WHEN GetAONumberString(l.pao_start_number, l.pao_start_suffix, l.pao_end_number, l.pao_end_suffix) <> ''
THEN
        GetAONumberString(l.pao_start_number, l.pao_start_suffix, l.pao_end_number, l.pao_end_suffix) || ' '
 ELSE '' END
 || CASE WHEN s.street_description IS NOT NULL THEN s.street_description || ', ' ELSE '' END
 || CASE WHEN s.locality_name IS NOT NULL THEN s.locality_name || ', ' ELSE '' END
 || CASE WHEN s.town_name IS NOT NULL THEN s.town_name ELSE '' END
 || CASE WHEN s.administrativearea <> s.town_name THEN ', ' || s.administrativearea ELSE '' END
 || CASE WHEN b.postcode_locator IS NOT NULL THEN ', ' || b.postcode_locator ELSE '' END
) AS bs7666_single_address_label
FROM addressbase_premium.lpi l INNER JOIN addressbase_premium.blpu b ON b.uprn = l.uprn
INNER JOIN addressbase_premium.street_descriptor s ON s.usrn = l.usrn
LEFT JOIN addressbase_premium.organisation o ON o.uprn = l.uprn
WHERE l.logical_status_code = 1 -- Condition to include approved addresses only (optional)
and l.language = s.language; --to ensure that the database forms an address based on the appropriate language if an
alternative language is available.
```

Notes:

1. The SQL operator for concatenating text is a double pipe ("||").
2. CASE blocks have been used to test each of the fields for null values before concatenating its contents (along with a suitable separator – either ", " or " ").
3. For clarity and brevity in the above example, a user-defined function has been assumed to exist, called GetAONumberString (highlighted in blue above), which accepts four arguments (start_number, start_suffix, end_number, end_suffix) and returns a string value formatted as described in the section "Rendering SAOs and PAOs". In a real-world implementation, this function would need to be explicitly written and created within the database, or the additional logic would need to be added and embedded in-line within the above SQL.
4. The field names and table names used are illustrative and may vary between databases.
5. Depending on the database schema and data loading method used, it may be necessary to test some fields for empty strings ('') or zero values (for integer fields) instead of, or as well as testing for NULLs.

8.5.5 Building a multi-line geographic address

Splitting a geographic address into multiple lines is more complex. As with Delivery Point Addresses, there are several rules to consider in order to avoid having very short lines (for example, just a building number) or very long lines within the formatted address. A summary of these rules is as follows:

- Generally, if there is a PAO number/range string, it should appear on the same line as the street description.

Example:	11A MAIN STREET

- If there is a pao_text value, it should always appear on the line above the street name (or on the line above the <pao number string> + <street name> where there is a pao number/range).

Example (pao_text only):

ROSE COTTAGE,
MAIN STREET

Example (pao_text and pao number or range):

ROSE COTTAGE,
11A MAIN STREET

- If there is a sao_text value, it should appear on a separate line above the pao_text line (or the pao number/range + street line where there is no pao_text value).

Example (sao_text value only, with pao_text value only):

THE ANNEXE,
ROSE COURT,
MAIN STREET

Example (sao_text value only, with pao number/range only):

THE ANNEXE,
11A MAIN STREET

- If there is a sao number/range value, it should be inserted either on the same line as the pao_text (if there is a pao_text value), or on the same line as the pao number/range + street name (if there is only a pao number/range value and no pao_text value). If there are both pao_text and a pao number/range, then the sao number/range should appear on the same line as the pao_text, and the pao number/range should appear on the street line.

Example (sao number/range value only, and pao_text value only):

1A ROSE COURT,
MAIN STREET

Example (sao number/range value only, and pao number/range value only):

1-3, 11A MAIN STREET

Example (sao number/range value only, and both pao_text and pao number/range values):

1A ROSE COURT,
11A MAIN STREET

- If there is a sao_text value, it should always appear on its own line.

Example (sao_text value only with pao_text only):

THE ANNEXE,
ROSE COTTAGE,
MAIN STREET

Example (sao_text and sao number/range and pao_text and pao number/range):

WARDEN'S FLAT,
1A ROSE COURT,
11A MAIN STREET

- If there is an organisation name, it should always appear alone as the top line of the address.

Example (organisation name along with all pao + sao fields):

COTTAGE INDUSTRY LTD,
THE ANNEXE,
1A ROSE COURT,
11A MAIN STREET

- The locality name (if present) should appear on a separate line beneath the street description, followed by the town name on the line below it. If there is no locality name, the town name should appear alone on the line beneath the street description.

Example (locality and town name present):

…(first part of address, formatted as described above) …
MAIN STREET,
HIGHFIELD,
SOUTHAMPTON

Example (town name only):

…(first part of address, formatted as described above) …
HIGH STREET,
SOUTHAMPTON

- If the administrative area name is desired, and if it is not a duplicate of the town name, if may optionally be included on a separate line beneath the town name.

Example (administrative area name included):

…(first part of address, formatted as described above) …
MAIN STREET,
WINDSOR,
ROYAL BOROUGH OF WINDSOR AND MAIDENHEAD

- Finally, the postcode locator, if present, should be inserted on the final line of the address

Example (with postcode_locator on final line):

…(first part of address, formatted as described above) …
HIGH STREET,
MILTON,
ML99 0WW

8.5 Creating mailing lists

Given that AddressBase Premium and Plus contain two different types of address, a decision needs to be made, based on business requirements, whether to use geographic or Delivery Point Addresses, or a mixture of the two.

The following two options should be considered:
a) Use Delivery Point Addresses whenever they are available, and when they are not, use a geographic address.
b) Use geographic addresses in all cases.

Depending on business requirements, in some user interfaces, it may be worth considering displaying both forms of an address, since this will provide the maximum information available about a given BLPU. "Mixing and matching" components from the two different forms of address into a single address label is not recommended as this is likely to cause confusion in some instances.

8.5.1 Address status

When building your query to extract a mailing list, it is important that you consider filtering your results based on the address status and type. The status of an address is often something that needs to be considered when working with address data. Questions like "is the addressable object in planning, being constructed, current, demolished or accurately positioned?" need to be answered before AddressBase Premium can be used effectively.

The table below offers guidance on what status filters should be considered. Please see the **technical specification** for more information about each of these attributes.

Note: the logical status attributes are only present in the AddressBase Premium product. AddressBase Plus only contains records that are "Approved" (logical status code 1).

Status attributes	Table	Use	Values
LOGICAL_STATUS	BLPU	Describes where a land or property unit is in its lifecycle.	1 – Approved
			6 – Provisional
			8 – Historical
LOGICAL_STATUS	LPI	Describes where an address is in its lifecycle.	1 – Approved
			3 – Alternative
			6 – Provisional
			8 – Historical
BLPU_STATE_CODE (optional)	BLPU	Informs the user what physical state the land or property is in (for example, "under construction","in use","demolished").	1 – Under construction
			2 – In use
			3 – Unoccupied
			4 – No longer existing
			6 – Planning permission granted null – Unknown or N/A
RPC_CODE	BLPU	To ascertain how accurate the coordinate is. Use in conjunction with the postcode_locator field to understand the accuracy of the address' position.	1 – Visual centre
			2 – General internal point
			3 – SW corner of 100m grid ref
			4 – Start of referenced street
			5 – Postcode unit point
			9 – Centre of Local Authority area
POSTAL_ADDRESS_CODE	BLPU	This field can be used to limit your records based on whether they are capable of receiving mail or not.	S – A single postal address
			N – Not a postal address
			C – A child address
			M – A parent address
LANGUAGE	LPI STREET_ DESCRIPTOR	This information can be used to limit your records based on the language.	ENG – English
			CYM – Welsh
			GAE – Gaelic

8.5.2 Other filters

AddressBase Premium is a rich addressing dataset that contains a wealth of other attributes that could be used in conjunction with address labels. For example, classification can be used to target certain types of property; or OS MasterMap® Topography TOID cross references can be used to link address labels to Topographic objects and viewed in a GIS.

9 Searching for addresses using AddressBase Premium

A common requirement for customers using the AddressBase products is to search for properties using full or partial addresses. Address searches may return a large number of addresses, a short list of possibilities, a single match, or no results, depending on the search criteria.

There are many potential ways of implementing an address search, from free text queries through to structured, address component searches. This guide will illustrate two such approaches that may be used when working with AddressBase Premium.

These methods are not intended as recommendations – merely as examples to illustrate how to get maximum value out of the product when implementing an address search function.

One type of search implementation involves a single "search engine" style text box, into which a user can type some or all of an address:

Find Address:

Results:

1, Clover Avenue, Fieldtown, Addressville, SW99 9ZZ
2, Clover Avenue, Fieldtown, Addressville, SW99 9ZZ
3, Clover Avenue, Fieldtown, Addressville, SW99 9ZZ
4, Clover Avenue, Fieldtown, Addressville, SW99 9ZZ
5, Clover Avenue, Fieldtown, Addressville, SW99 9ZZ
6, Clover Avenue, Fieldtown, Addressville, SW99 9ZZ
7, Clover Avenue, Fieldtown, Addressville, SW99 9ZZ

In this scenario, the user can choose to type anything in the box, which may be just one component of an address (for example, a postcode, street name or building name), several parts of an address (for example, street name + town name, house name + postcode, and so on), or even (rarely) a complete address. There may or may not be commas between search items, the user may or may not have capitalised some letters, and so on. In short, with this search method, there is no structure to the user input, and the search methodology must be designed with this in mind.

The other common type of implementation for address searches involves guiding the user to enter search criteria in a structured way (for example, with a different text box for each major address component):

Find Address:

House name: Rose Cottage

House number:

Street name: Main Street

Locality:

Town:

Postcode:

Search

Results:

Rose Cottage, Main Street, Fieldtown, Addressville, SW99 9ZZ
Rose Cottage, Main Street, Ashford, AS45 9PP
Rose Cottage, Main Street, Buxtew, Monley, MO88 4TY
And so on…

This method guides the user to enter known components of an address, and also creates a predictable user input structure around which to build a search function. While generally simpler to use and implement, in can be less user-friendly – particularly in cases where it is not obvious which box to type an address component into (for example, "Richmond Terrace" – is it a building name or a street? And so on…).

This document outlines possible solutions for implementing both of the search methods described above using AddressBase Premium. It should be used alongside the document on formatting single address labels using AddressBase products. Note that the methods described here may be adapted to work with both AddressBase Plus and AddressBase; however, in the case of AddressBase Plus, it is not possible to search alternative or historicalal addresses, and AddressBase provides postal (delivery point) addresses only, so the geographic guidance does not apply to this product.

9.1 Understanding the different addresses available in AddressBase Premium

As described in chapter 8; at a high level AddressBase Premium provides two different types of address – the Delivery Point Address and the geographic address.

However, for some geographic addresses an alternative, provisional, historicalal or language variant* of the approved record may also be provided as well as the approved address (all sharing the same UPRN).

*The Delivery Point Address also optionally contains a Welsh variant of the Royal Mail record.

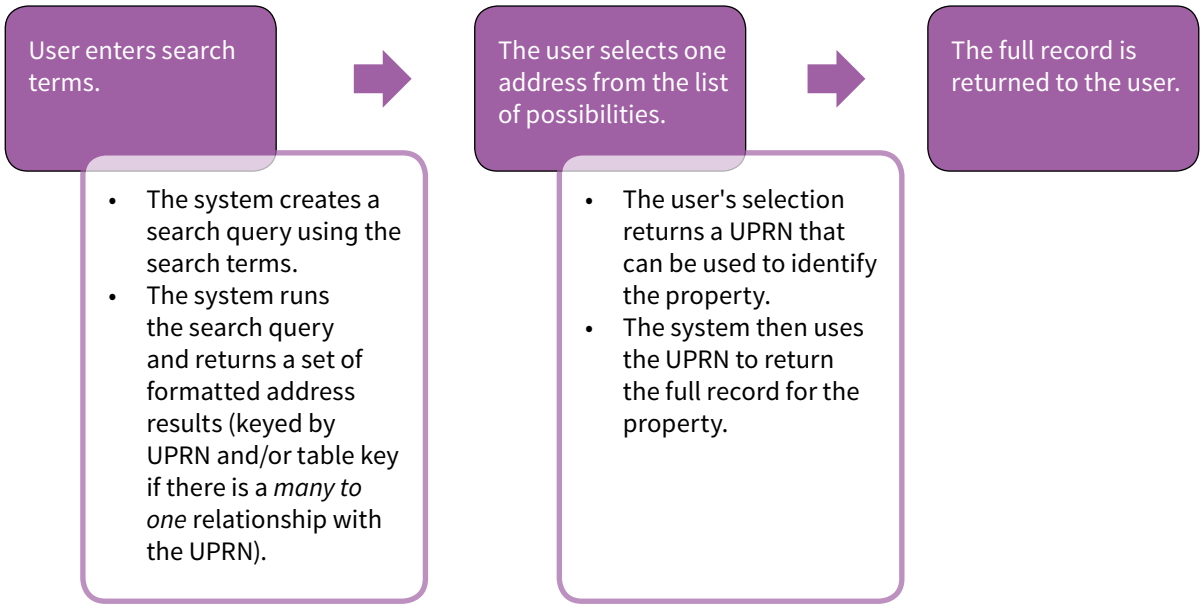
The following table outlines what these addresses are and how to access them in the product.

Address type	What is it?	Where is it?
Delivery Point Address	The postal address, as assigned to the property by Royal Mail (and widely used by the public).	Delivery Point Address Table.
Approved geographic address	The legal/approved address as assigned by the local naming and numbering authority.	LPI Table with Logical Status = 1, joined to street descriptor, organisation and BLPU tables.
Provisional geographic address	Provisional addresses may exist for a property from the moment that an address has been granted planning permission to be built to the time when construction has been completed.	LPI Table with Logical Status = 6, joined to street descriptor, organisation and BLPU tables.
Alternative geographic address	Any alternative addresses that may exist for this property (for example, alternative names). [There may be more than one alternative address per property]	LPI Table with Logical Status = 3, joined to street descriptor, organisation and BLPU tables.
Historicalal geographic address	Any historicalal addresses (recorded since data collection began) that may have existed in the past for this property (for example, previous house names or business names, and so on…). [There may be more than one historicalal address per property]	LPI Table with Logical Status = 8, joined to street descriptor, organisation and BLPU tables.
Alternative Language	Where there is more than one official language in use – that is English and Welsh in Wales, and English and Scottish Gaelic in Scotland – there may be two versions of a Geographic address, one for each language.	LPI table with Language Code = CYM or GAE and Street_ Descriptor table with Language Code = CYM or GAE.

To provide the most comprehensive address search functionality, any implementation of a search function should make full use of all these different address types.

9.2 The search process

An address search operation typically requires two stages of interaction from a user, and several processing steps from the underlying IT system. These steps can be summarised in the following diagram:



The second user interaction can be omitted if there is only one result returned from the query. In almost all cases, there should be an option to "search again" at the second and third stages in case no results are returned, or if none of the options shown is the required address.

Of course, different applications require different approaches; however, the general principles of the above process apply in all cases where an address is searched for based on user-entered criteria.

9.3 Generating a search query from structured user input

Within an interface that accepts structured user input for an address search, it is necessary to "map" the fields presented to the user with those found within AddressBase Premium. In particular, any query will need to test multiple fields for a given input, and will need to combine result sets from two different address formats (Delivery Point Address and geographic address) in order to produce the most complete result set.

Generally a search form will describe a "simplified" view of an address in order to keep the user interface tidy and intuitive. Users may be given a set of text boxes to fill in, generally including building name, building number, street name, locality name, town name, and postcode. The relationships between some common search fields and the fields found in AddressBase Premium are as follows:

Search Box	Mapped Delivery Point fields	Mapped geographic fields
Business Name	Organisation_Name	Organisation PAO_Text SAO_Text
Flat/Subdivision Name	Sub_Building_Name Department_Name	SAO_Text
Flat/Subdivision Number	Sub_Building_Name	SAO_StartNumber SAO_StartSuffix SAO_EndNumber SAO_EndSuffix
Building Name	Building_Name	PAO_Text
Building Number	Building_Number Building_Name (in cases where a suffix or range is present)	PAO_StartNumber PAO_StartSuffix PAO_EndNumber PAO_EndSuffix
Street	Thoroughfare Dependent_Thoroughfare	Street PAO_Text
Locality	Dependent_Locality Double_Dependent_Locality	Locality Town Street
Town	Dependent_Locality Post_Town	Town Locality
Postcode	Postcode	Postcode_Locator

The above mapping is an example only, and it is possible to break the search fields down differently, in which case a different mapping would be required. The important thing is to consider all possibilities for how data might be recorded. For example, a business name can sometimes appear as an organisation name or a building/pao name depending on circumstances, so both must be checked when creating a search query.

Numbers need to be handled especially carefully due to the presence of suffixes and ranges. There are two options for structuring the search input in these cases: a single "number" box can be used (as shown above) which will then require some string manipulation to split the input into the appropriate numeric range and suffix components in order to search the geographic addresses; or four boxes can be provided for each number (start number, start suffix, end number, end suffix) which would then need to be combined into an appropriate string to search the Delivery Point Addresses.

9.4 Structuring the query for a structured address search

The basic rules to follow when generating a search query from structured input are as follows:

1. Ignore any search boxes that are not filled in with values.
2. Where a value is entered, assume that a match on at least one of the mapped fields is essential.

In SQL query terms, this means that each search term should generate a sub-query that searches each of the mapped fields (using OR), and that these sub-queries should then be combined together (using AND) into a single search query. The following SQL code illustrates this (for the Delivery Point Address search only) where a street, locality and town name have been entered by the user:

```
SELECT dp.UPRN, GetFormattedAddress(dp.*) FROM addressbase_premium.dpa dp WHERE
(dp.thoroughfare = streetsearchtext OR dp.dependent_thoroughfare = streetsearchtext) AND
(dp.dependent_locality = localitysearchtext OR dp.double_dependent_locality = localitysearchtext) AND
(dp.dependent_locality = townsearchtext OR dp.post_town = townsearchtext)
```

In the above example, "streetsearchtext", "localitysearchtext", and "townsearchtext" (shown in blue) represent user-entered search terms (which could be parameters within an SQL function) and the GetFormattedAddress(*) function is a hypothetical user-defined function that returns the formatted address as a single string (suitable for display in the user interface). For more information on formatting addresses, please read chapter 8.

On top of this, for a complete query, the two different types of addresses should be queried separately (geographic and Delivery Point Addresses), and the two result sets should be amalgamated into a single set using a UNION. The following example builds upon the previous example to include geographic addresses as well as Delivery Point Addresses:

```
SELECT dp.UPRN, GetFormattedAddress(dp.*) FROM addressbase_premium.dpa dp WHERE
(dp.thoroughfare = streetsearchtext OR dp.dependent_thoroughfare = streetsearchtext) AND
(dp.dependent_locality = localitysearchtext OR dp.double_dependent_locality = localitysearchtext) AND
(dp.dependent_locality = townsearchtext OR dp.post_town = townsearchtext)
UNION
SELECT b.uprn, GetFormattedAddress(b.*, l.*, s.*, o.*) FROM addressbase_premium.blpu b
INNER JOIN addressbase_premium.lpi l ON l.uprn = b.uprn
INNER JOIN addressbase_premium.street_descriptor s ON s.usrn = b.usrn
LEFT JOIN addressbase_premium.organisation o ON o.uprn = b.uprn WHERE
(s.street_name = streetsearchtext OR l.pao_text = streetsearchtext) AND
(s.locality = localitysearchtext OR s.town = localitysearchtext OR s.street_name = localitysearchtext) AND
(s.town = townsearchtext OR s.locality = townsearchtext)
```

The geographic query requires four joins between the BLPU, LPI, Street_Descriptor and Organisation tables in order to access all the fields required to build an address.

The SQL UNION operator will combine the two result sets, discarding any exact duplicates (retaining the exact duplicates requires the use of UNION ALL, but that is not desirable in this example).

The resulting output from this query will be a set of search results: formatted addresses along with their UPRN. Exact duplicates will be omitted, but all "variations" of the same address will be outputted (one row for each variation, with the same UPRN repeated more than once potentially). It may be wise to also return the "logical status" and/or "postal address flag" values against each to enable further filtering (that is, to include or exclude historicalal addresses, for example or to restrict the results to postal addresses only).

9.5 Supporting case-insensitive queries and partial matches

A flaw in the above examples is the use of equality operators. In practice, because people do not tend to be consistent with capitalisation of letters, the SQL "LIKE" operator might work better, and depending on the nature of the application, a "%" wildcard could be appended to the end of each search term to allow only the first few letters of an address component to be entered, for example.

```
dp.post_town LIKE townsearchtext                -- Case insensitive search in some databases
dp.post_town LIKE (townsearchtext || '%')       -- Matches post towns that start with the search text
dp.post_town LIKE ('%' || townsearchtext || '%') -- Matches post towns that contain the search text
```

Alternatively, if exact matches are required but case sensitivity is not, then the UPPER() or LOWER() SQL functions can be used on each side of the equals sign in comparisons (a solution that should work in all databases):

```
UPPER(dp.post_town) = UPPER(townsearchtext)     -- Case insensitive equality
```

Finally, to combine all of the approaches, the following would work for maximum flexibility:

```
UPPER(dp.post_town) LIKE ('%' || UPPER(townsearchtext) || '%')
```

9.6 Generating a search query from unstructured user input

When offering a "search engine" style search feature, with just a single text box to enter search terms, a wholly different approach is required. No assumptions can be made about the order, format or style of the user input, and the data will need to be "indexed" in a way that facilitates searches of this type.

9.7 Creating a "Search Index" for addresses

Search engine style searches are likely to require the creation of an additional index/lookup table for addresses. Such a table is likely to consist of just two main columns: a key value (UPRN) and a formatted address string. Additional columns may be required to allow filtering of results (such as the "logical status" values, which would allow the results to be filtered on "approved", "provisional" and "historicalal" statuses, for example).

The following table shows a possible "address index" table structure:

UPRN	Address Text	Statuses (multiple fields)
123456789012	4 THE MEADOWS HIGH STREET WALTHAMSDALE BURRIDGE BU27 9UB	Approved
123456789012	FLAT 4 THE MEADOWS HIGH STREET WALTHAMSDALE BURRIDGE BU27 9UB	Alternative + PAF
123456789012	4 HIGH STREET WALTHAMSDALE CLOSE BURRIDGE BU27 9UB	Historicalal
947364758903	ROSE COTTAGE MAIN STREET HAVERSHAM SUDBURY SU45 9TY	Approved + PAF
947364758903	ROSE FARMHOUSE MAIN STREET HAVERSHAM SUDBURY	Historicalal

Note how the addresses have been formatted as a single text string with a single space between each word (although leaving commas in would do no harm). All forms of each address (both PAF and geographic, current and historicalal, approved and alternative) have been added to the index, so there can be several rows with the same UPRN. To speed up complex searching, an appropriate index could be added to the Address Text field, such as a full text search index.

9.8 Structuring the query for an unstructured address search

Once a suitable search index is in place, the query itself can be put together. The basic idea is to split the user input into search terms by removing commas, double spaces and other unnecessary whitespace and then splitting it at each single space, as follows:

User Input: 4, High Street, westville, wv17

Capitalised, with commas and double spaces removed:

4 HIGH STREET WESTVILLE WV17

Split into separate search terms:

- 4
- HIGH
- STREET
- WESTVILLE
- WV17

Once the user input has been pre-processed into separate search terms, a query can be generated. The key assumption in this example will be that ALL search terms must be matched against the index table to be considered as a result; this implies a query where each value is matched using an "AND" operator. In order to search the whole index, the "LIKE" operator will need to be used along with a "%" wildcard on either side of the search text. A suitable search query for the above example would be as follows:

```
SELECT UPRN, AddressText FROM AddressSearchIndex
WHERE
AddressText LIKE '%4%' AND
AddressText LIKE '%HIGH%' AND
AddressText LIKE '%STREET%' AND
AddressText LIKE '%WESTVILLE%' AND
AddressText LIKE '%WV17%';
```

This query would return all rows from the index table that contain all of the search terms, along with the appropriate UPRNs. The following table shows how the index table would be used in the above example to return relevant results:

UPRN	Address Text	Statuses (multiple fields)
894756389092	4 HIGH STREET WESTVILLE SUNNYTOWN WV17 7HL	Approved + PAF
894756389092	ROSE COTTAGE 4 HIGH STREET WESTVILLE SUNNYTOWN WV17 7HL	Alternative
894756389092	ROSE COTTAGE HIGH STREET WESTVILLE SUNNYTOWN WV17 7HL	Alternative
274859037849	FLAT 4 HIGHBURY COURT HIGH STREET WESTVILLE SUNNYTOWN WV17 7HL	Approved + PAF
482974769830	MAPS4U LTD HIGH STREET WESTVILLE SUNNYTOWN WV17 7HL	Approved

This result set can then be presented to the user, who can then select the most appropriate record (which can then be retrieved in full using the UPRN).

Of course, in a practical implementation, the above query would need to be dynamically generated, with a separate condition added for each search term. This example is quite a strict search query that requires all search terms to be present; many layers of complexity could be added to allow partial and "fuzzy" matches, and to return confidence scores, for example, but such enhancements are beyond the scope of this guide.

9.9 Summary

This guide was intended as an introduction to implementing address search functionality using AddressBase Premium. The following is a summary of the main points:

- A user front-end for an address search may contain a single, search-engine style textbox, or multiple textboxes representing different parts of an address.
- A typical address search function takes place in three stages: 1) a user enters search text 2) a query is run, returning a set of possible matches; and 3) the user selects the address of interest and the full record is then returned.
- With a structured search interface, the addresses can be queried directly by mapping the various address fields to the textboxes supplied.
- For an unstructured (single text box) interface, it is necessary to create an index table with fully formatted address strings against each UPRN. Queries can then be run against this index table by splitting the user input into individual search terms and requiring them all to be present.
- It is possible to filter results by status ("approved", "alternative" and "historicalal", as well as "postal" or "non-postal", and so on…).
- Any search function should search all forms of an address (both geographic and Delivery Point Addresses, including provisional and historicalal variations).
- Careful consideration should be given to the use of "fuzzy" search algorithms used (for example using wildcard or sound-alike searches).

10 Proposed AddressBase Premium COU Implementation

10.1 Introduction

In July 2013, Ordnance Survey made AddressBase products available to licenced users as a Change Only Update (COU) supply. This guidance documents presents the logic for applying the COU to AddressBase Premium which is significantly more complex to manage when stored in a database than AddressBase and AddressBase Plus. This has been supplied as technical code where possible.

10.2 Background

AddressBase Premium is a complex relational dataset that is being used by a variety of customers who are using a variety of methods and software to manage the data. Some of the software solutions take a considerable length of time to load and manage the data and therefore COU is being seen as a simple and effective way to keep data holdings up-to-date without spending considerable time loading and managing a full supply every time the data is refreshed.

10.3 Types of change

At a high level there are three types of change found within COU:

1 Deletes (CHANGE_TYPE 'D') are objects that have ceased to exist in your area of interest since the last product refresh.
2 Inserts (CHANGE_TYPE 'I') are objects that have been newly inserted into your area of interest since the last product refresh.
3 Updates (CHANGE_TYPE 'U') are objects that have been updated in your area of interest since the last product refresh.

10.4 High level COU implementation model

The table below illustrates how to implement the AddressBase Premium COU within a database.

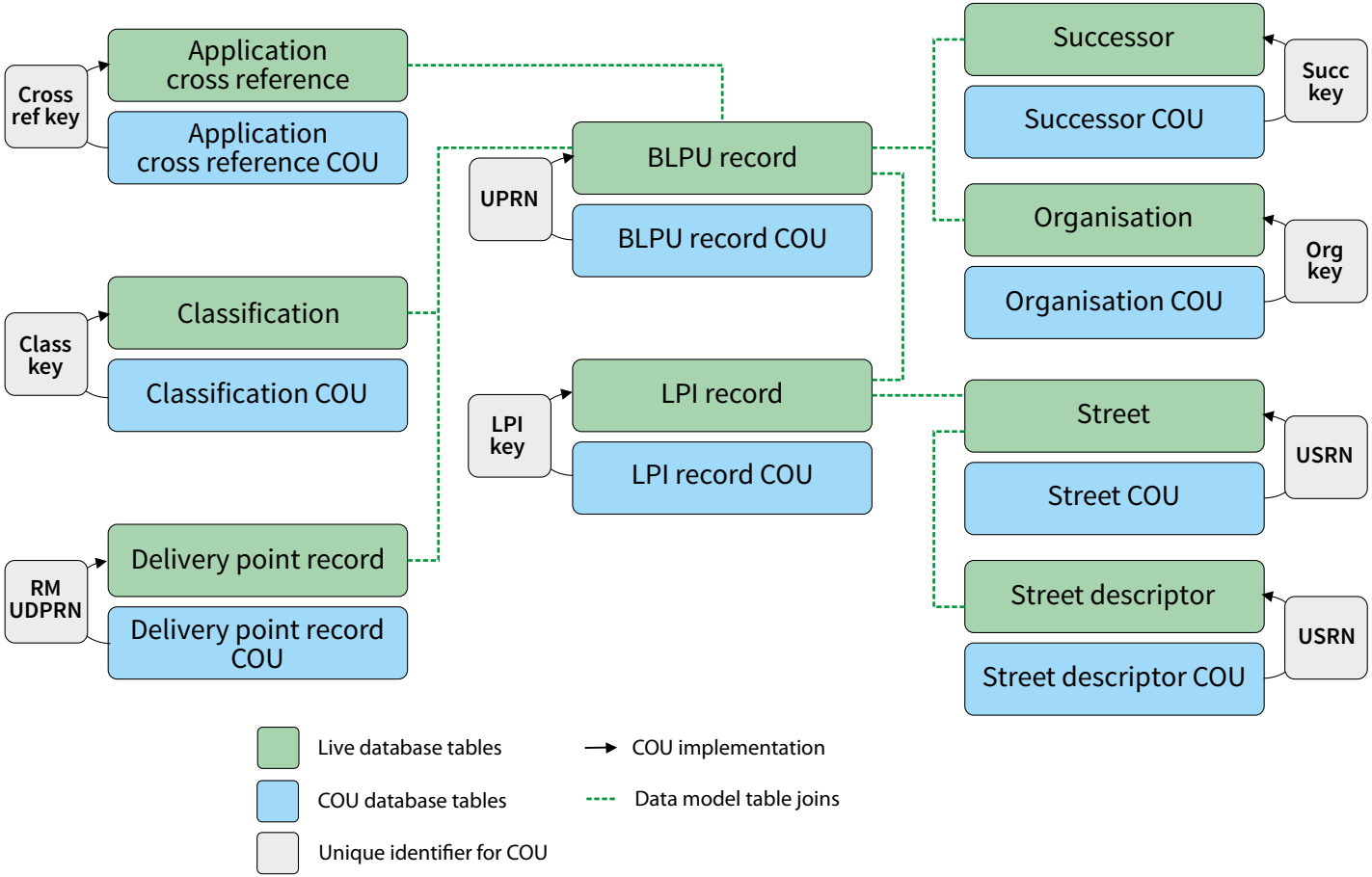


Figure 7: High level COU implementation model

10.5 High level COU implementation model – with archiving

Before COU is applied there may be a business requirement to archive existing address records. The table below illustrates how to implement the AddressBase Premium COU within a database.

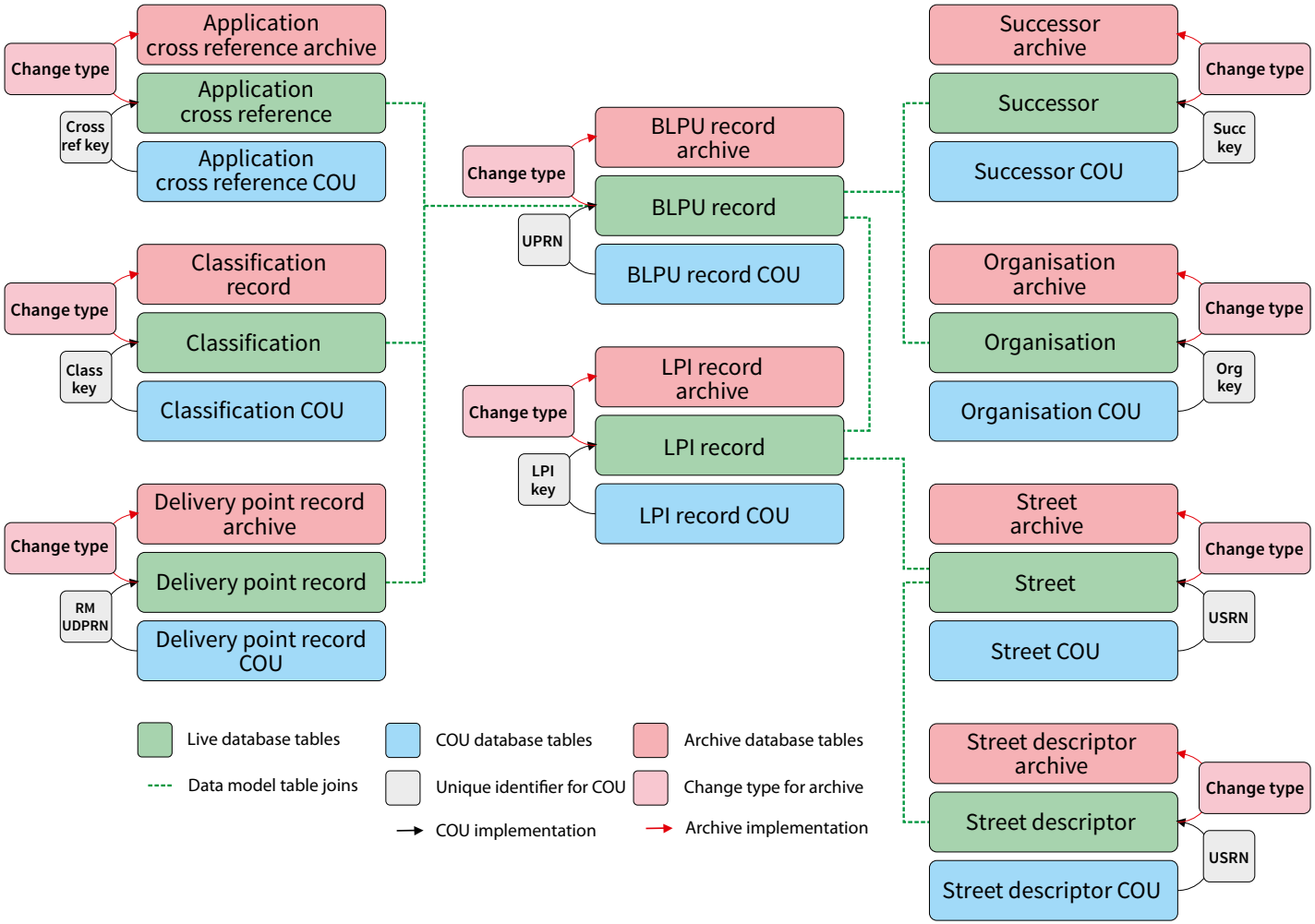


Figure 8: High level COU implementation model on how to create Archive tables and apply COU

10.6 Applying COU to tables

10.6.1 Changes to BLPU record

Within the BLPU table there will not be any records with the same UPRN. This can be tested by checking the number of records that have the same UPRN. The following SQL code would notify you of any duplicates.

```
SELECT uprn, COUNT(uprn) AS NumOccurrences
FROM addressbasepremium.abp_blpu_record
GROUP BY uprn
HAVING ( COUNT(uprn) > 1 );
```

This query should return 0 rows and this confirms there are no duplicates.

As there are no duplicate records we can therefore use the UPRN to apply the COU.

Once confirmed the following steps can be taken to apply the COU (without archiving):

#Initially delete the existing records that will be Updated and Deleted

```
DELETE FROM addressbasepremium.abp_blpu_record WHERE uprn IN (SELECT distinct uprn FROM addressbasepremium_cou.abp_blpu_record WHERE change_type != 'I');
```

#Insert the new updated records and the new inserted records

```
INSERT INTO addressbasepremium.abp_blpu_record SELECT * FROM addressbasepremium_cou.abp_blpu_record WHERE change_type != 'D';
```

Some of the COU records that are Change Type 'U' (Updates) may change the Logical Status Code from '1' to '8' meaning that this address has become 'Historical'. This means that the BLPU table intrinsically archives historical record.

Where there is a business requirement to keep the records that are being Updated and Deleted in a separate archive table the following SQL will create an Archive Table and populate with records that are being Updated and Deleted from the live BLPU table.

#The following command creates an archive table of the records that are being updated and deleted from the existing BLPU table

#If this table already exists you can simple use INSERT INTO rather than CREATE TABLE

```
CREATE TABLE addressbasepremium.abp_blpu_record_archive AS SELECT * FROM addressbasepremium.abp_blpu_record WHERE uprn IN (SELECT distinct uprn FROM addressbasepremium_cou.abp_blpu_record WHERE change_type != 'I');
```

#The following command then deletes the records from the existing table which are either Updates or Deletions

```
DELETE FROM addressbasepremium.abp_blpu_record WHERE uprn IN (SELECT distinct uprn FROM addressbasepremium_cou.abp_blpu_record WHERE change_type != 'I');
```

#The following command then inserts the new Insert records and the new Updated records into the live BLPU table.

```
INSERT INTO addressbasepremium.abp_blpu_record SELECT * FROM addressbasepremium_cou.abp_blpu_record WHERE change_type != 'D';
```

10.6.2 Changes to Classification table

Because there is a one-to-many relationship between the BLPU table and the Classification table there can be records in the Classification table that share a UPRN. To apply COU to the correct record users should use the Class_Key to ensure that the correct classification record is updated.

Figures 9: This example illustrates a scenario when a user would need to choose between two Classification records that have the same UPRN. In this case the Class_Key has been used to apply the COU to 'Record 2'

Classification	Record 1	Record 2
Record identifier	32	32
Change type	I	I
Pro order	706838	706839
UPRN	116000665	116000665
Class key	9055C000081107	9055C000001834
Classification code	U	R
Class scheme	AddressBase Premium classification scheme	AddressBase Premium classification scheme
Scheme version	1.0.0	1.0.0
Start date	2011-12-01	2010-03-16
End date	N/A	N/A
Last update date	2011-12-01	2010-08-12
Entry date	2011-12-01	2010-03-16

Classification	COU record
Record identifier	32
Change type	U
Pro order	331481
UPRN	116000665
Class key	9055C000081107
Classification code	CL10RE
Class scheme	AddressBase Premium classification scheme
Scheme version	1.0.0
Start date	2011-12-01
End date	N/A
Last update date	2013-05-04
Entry date	2011-12-01

Classification	Updated record	Record 2
Record identifier	32	32
Change type	U	I
Pro order	331481	706839
UPRN	116000665	116000665
Class key	9055C000081107	9055C000001834
Classification code	CL10RE	R
Class scheme	AddressBase Premium classification scheme	AddressBase Premium classification scheme
Scheme version	1.0.0	1.0.0
Start date	2011-12-01	2010-03-16
End date	N/A	N/A
Last update date	2013-05-04	2010-08-12
Entry date	2011-12-01	2010-03-16

To achieve this outcome (without archiving the 'old' record 2), we can use the following SQL commands to apply the COU

#Initially delete the existing records that are being Updated and Deleted

```
DELETE FROM addressbasepremium.abp_classification WHERE class_key IN (SELECT distinct class_key FROM
addressbasepremium_cou.abp_classification WHERE change_type != 'I');
```

#Insert the new Update records and the new Insert records

```
INSERT INTO addressbasepremium.abp_classification SELECT * FROM addressbasepremium_cou.abp_classification
WHERE change_type != 'D';
```

One thing you may want to consider is keeping an archive of the updated and deleted Classification records. For example, this might be useful to understand when an address has changed use from residential to commercial.

To achieve this outcome for change types 'U' or 'D' (with archiving) we can use the following SQL commands to apply the COU

#The following command creates an archive table of the records that are being updated and deleted from the existing Classification table. If this table already exists you can simple use INSERT INTO rather than CREATE TABLE

```
CREATE TABLE addressbasepremium.abp_classification_archive AS SELECT * FROM addressbasepremium.abp_
classification WHERE class_key IN (SELECT distinct class_key FROM addressbasepremium_cou.abp_classification WHERE
change_type != 'I');
```

#The following command then deletes the records from the existing table which are either Updates or Deletions

```
DELETE FROM addressbasepremium.abp_classification WHERE class_key IN (SELECT distinct class_key FROM
addressbasepremium_cou.abp_classification WHERE change_type != 'I');
```

#The following command then inserts the new Insert records and the new Updated records into the Classification table.

```
INSERT INTO addressbasepremium.abp_classification SELECT * FROM addressbasepremium_cou.abp_classification
WHERE change_type != 'D';
```

Figure 10

Classification	Archive record
Record identifier	32
Change type	I
Pro order	706838
UPRN	116000665
Class key	9055C000081107
Classification code	CL10RE
Class scheme	AddressBase Premium classification scheme
Scheme version	1.0.0
Start date	2011-12-01
End date	N/A
Last update date	2011-12-01
Entry date	2011-12-01

One thing to note, when the Update or Deleted records are moved into an Archive table, the End Date may not always be populated as seen in Figure 4. If this is the case, users may wish to consider adding an end_date (which could be based on the epoch date that it was archived) as shown in Figure 5. This will enable querying for a particular timeframe.

Figure 11

Classification	Archive record
Record identifier	32
Change type	I
Pro order	706838
UPRN	116000665
Class key	9055C000081107
Classification code	CL10RE
Class scheme	AddressBase Premium classification scheme
Scheme version	1.0.0
Start date	2011-12-01
End date	2013-05-04
Last update date	2011-12-01
Entry date	2011-12-01

10.6.3 Changes to the Organisation table

Because there is a one-to-many relationship between the BLPU table and the Organisation table there can be records in the Organisation table that share a UPRN. To apply COU to the correct record users should use the Org_Key to ensure that the correct classification record is updated.

To apply COU to the Organisation table (without archiving) the following code can be used:

#Initially delete the existing records that will be Updated and Deleted

```
DELETE FROM addressbasepremium.abp_organisation WHERE org_key IN (SELECT distinct org_key FROM
addressbasepremium_cou.abp_organisation WHERE change_type != 'I');
```

#Insert the new updated records and the new inserted records

```
INSERT INTO addressbasepremium.abp_organisation SELECT * FROM addressbasepremium_cou.abp_organisation WHERE
change_type != 'D';
```

As with the Classification table, the changes in Organisation name may be useful to keep as archives allowing a business to find previous organisations and understand when those may have changed.

To apply COU to the Organisation table (with archiving) the following code can be used:

#The following command creates an archive table of the records that are being updated and deleted from the existing Organisation table. If this table already exists you can simple use INSERT INTO rather than CREATE TABLE

```
CREATE TABLE addressbasepremium.abp_organisation_archive AS SELECT * FROM addressbasepremium.abp_
organisation WHERE org_key IN (SELECT distinct org_key FROM addressbasepremium_cou.abp_organisation WHERE
change_type != 'I');
```

#The following command then deletes the records from the existing table which are either Updates or Deletions

```
DELETE FROM addressbasepremium.abp_organisation WHERE org_key IN (SELECT distinct org_key FROM
addressbasepremium_cou.abp_organisation WHERE change_type != 'I');
```

#The following command then inserts the new Insert records and the new Updated records into the Organisation table.

```
INSERT INTO addressbasepremium.abp_organisation SELECT * FROM addressbasepremium_cou.abp_organisation WHERE
change_type != 'D';
```

10.6.4 Changes to the Delivery Point Address table

Within the Delivery Point Address table there will not be any records with the same RM_UDPRN. This can be tested by checking the number of records that have the same RM_UDPRN. The following SQL code would notify you of any duplicates.

```
SELECT rm_udprn, COUNT(rm_udprn) AS NumOccurrences
FROM addressbasepremium.abp_delivery_point
GROUP BY rm_udprn
HAVING ( COUNT(rm_udprn) > 1 );
```

This query should return 0 rows and this confirms there are no duplicates.

As there are no duplicate records we can therefore use the RM_UDPRN to apply the COU.

To apply COU to the Delivery Point Address table (without archiving) the following code can be used:

#Initially delete the existing records that will be Updated and Deleted

```
DELETE FROM addressbasepremium.abp_delivery_point WHERE rm_udprn IN (SELECT distinct rm_udprn FROM
addressbasepremium_cou.abp_delivery_point WHERE change_type != 'I');
```

#Insert the new updated records and the new inserted records

```
INSERT INTO addressbasepremium.abp_delivery_point SELECT * FROM addressbasepremium_cou.abp_deliverypoint
WHERE change_type != 'D';
```

The Delivery Point Address table does not have the ability to hold historical records as it is the current view of the Royal Mail Delivery Point Address File (PAF). Therefore, in order to capture the historical records you will need to create an archive table that is populated when records are either deleted or updated. The following code will create the archive records.

To apply COU to the Delivery Point Address table (with archiving) the following code can be used:

#The following command creates an archive table of the records that are being updated and deleted from the existing Delivery Point Address table. If this table already exists you can simple use INSERT INTO rather than CREATE TABLE

```
CREATE TABLE addressbasepremium.abp_delivery_point_archive AS SELECT * FROM addressbasepremium.abp_delivery_
point WHERE rm_udprn IN (SELECT distinct rm_udprn FROM addressbasepremium_cou.abp_delivery_point WHERE
change_type != 'I');
```

#The following command then deletes the records from the existing table which are either Updates or Deletions

```
DELETE FROM addressbasepremium.abp_delivery_point WHERE uprn IN (SELECT distinct uprn FROM
addressbasepremium_cou.abp_delivery_point WHERE change_type != 'I');
```

#The following command then inserts the new Insert records and the new Updated records into the Delivery Point Address table.

```
INSERT INTO addressbasepremium.abp_delivery_point SELECT * FROM addressbasepremium_cou.abp_delivery_point
WHERE change_type != 'D';
```

10.6.5 Changes to LPI table

Because there is a one-to-many relationship between the BLPU table and the LPI table there can be records in the LPI table that share a UPRN. To apply COU to the correct record users should use the LPI_Key to ensure that the correct classification record is updated.

To apply COU to the LPI table (without archiving) the following code can be used:

#Initially delete the existing records that will be Updated and Deleted.

```
DELETE FROM addressbasepremium.abp_lpi WHERE lpi_key IN (SELECT distinct lpi_key FROM addressbasepremium_cou.
abp_lpi WHERE change_type != 'I');
```

#Insert the new updated records and the new inserted records

```
INSERT INTO addressbasepremium.abp_lpi SELECT * FROM addressbasepremium_cou.abp_lpi WHERE change_type != 'D';
```

As with the BLPU table, some of the COU records that are Change Type 'U' (Updates) may change the Logical Status Code from '1' to '8' meaning that this address has become 'Historical'. This means that the LPI table intrinsically archives historical record.

Where there is a business requirement to keep the records that are being Updated and Deleted in a separate archive table the following SQL will create an Archive Table and populate with records that are being Updated and Deleted from the live LPI table.

To apply COU to the LPI table (with archiving) the following code can be used:

#The following command creates an archive table of the records that are being updated and deleted from the existing LPI table. If this table already exists you can simple use INSERT INTO rather than CREATE TABLE

```
CREATE TABLE addressbasepremium.abp_lpi_archive AS SELECT * FROM addressbasepremium.abp_lpi WHERE lpi_key IN
(SELECT distinct lpi_key FROM addressbasepremium_cou.abp_lpi WHERE change_type != 'I');
```

#The following command then deletes the records from the existing table which are either Updates or Deletions

```
DELETE FROM addressbasepremium.abp_lpi WHERE lpi_key IN (SELECT distinct lpi_key FROM addressbasepremium_cou.
abp_lpi WHERE change_type != 'I');
```

#The following command then inserts the new Insert records and the new Updated records into the LPI table.

```
INSERT INTO addressbasepremium.abp_lpi SELECT * FROM addressbasepremium_cou.abp_lpi WHERE change_type != 'D';
```

Figure 12: Shows an original LPI Record next to a COU Record. In this example the record is being made historical (Logical Status Code: 8) and therefore has a populated End Date attribute.

LPI	Record	COU Record
Record identifier	24	24
Change type	I	U
Pro order	478857	478857
UPRN	100000527208	100000527208
LPI key	4520L000005174	4520L000005174
Language	ENG	ENG
Logical status	1	8
Start date	"2001-03-23"	"2001-03-23"
End date		"2013-04-24"
Last update date	"2010-05-21"	"2013-04-24"
Entry date	"2001-03-23"	"2001-03-23"
SAO start number		
SAO start suffix		
SAO end number		
SAO end suffix		
SAO text		
PAO start number		
PAO start suffix		
PAO end number		
PAO end suffix		
PAO text	"SITE OF FORMER MISER NETHAULERS"	"FORMER SITE OF MISER NETHAULERS"
USRN	36815950	36815950
USRN match indicator	1	1
Area name		
Level		
Official flag		

10.6.6 Changes to Street table

Within the Street table there will not be any records with the same USRN. This can be tested by checking the number of records that have the same USRN. The following SQL code would notify you of any duplicates.

```
SELECT usrn, COUNT(usrn) AS NumOccurrences
FROM addressbasepremium_rsa.abp_street
GROUP BY usrn
HAVING ( COUNT(usrn) > 1 );
```

This query should return 0 rows and this confirms there are no duplicates.

As there are no duplicate records we can therefore use the USRN to apply the COU.

To apply COU to the Street table (without archiving) the following code can be used:

#Initially delete the existing records that will be Updated and Deleted

```
DELETE FROM addressbasepremium.abp_street WHERE usrn IN (SELECT distinct usrn FROM addressbasepremium_cou.
abp_street WHERE change_type != 'I');
```

#Insert the new updated records and the new inserted records

```
INSERT INTO addressbasepremium.abp_street SELECT * FROM addressbasepremium_cou.abp_street WHERE change_type
!= 'D';
```

The Street table does not have the ability to hold historical records as it does not have a Logical Status Code attribute. Therefore, in order to capture the historical records you will need to create an archive table that is populated when records are either deleted or updated.

To apply COU to the LPI table (with archiving) the following code can be used:

#The following command creates an archive table of the records that are being updated and deleted from the existing Street table. If this table already exists you can simple use INSERT INTO rather than CREATE TABLE

```
CREATE TABLE addressbasepremium.abp_street_archive AS SELECT * FROM addressbasepremium.abp_street WHERE usrn
IN (SELECT distinct usrn FROM addressbasepremium_cou.abp_street WHERE change_type != 'I');
```

#The following command then deletes the records from the existing table which are either Updates or Deletions

```
DELETE FROM addressbasepremium.abp_street WHERE usrn IN (SELECT distinct usrn FROM addressbasepremium_cou.
abp_street WHERE change_type != 'I');
```

#The following command then inserts the new Insert records and the new Updated records into the Street table.

```
INSERT INTO addressbasepremium.abp_street SELECT * FROM addressbasepremium_cou.abp_street WHERE change_type
!= 'D';
```

10.6.7 Changes to Street Descriptor table

Within the Street Descriptor table there will not be any records with the same USRN and the same language. This can be tested by checking the number of records that have the same USRN. The following SQL code would notify you of any duplicates.

```
SELECT usrn, language, COUNT(usrn) AS NumOccurrences
FROM addressbasepremium.abp_street_descriptor
GROUP BY usrn, language
HAVING ( COUNT(usrn) > 1 );
```

This query should return 0 rows and this confirms there are no duplicates.

As there are no duplicate records we can therefore use the USRN to apply the COU.

To apply COU to the LPI table (without archiving) the following code can be used:

```
#Initially delete the existing records that will be Updated and Deleted

DELETE FROM addressbasepremium.abp_street_descriptor WHERE EXISTS (SELECT 1 FROM addressbasepremium_cou.
abp_street_descriptor

WHERE addressbasepremium_cou.abp_street_descriptor.usrn = addressbasepremium.abp_street_descriptor.usrn

AND addressbasepremium_cou.abp_street_descriptor.language = addressbasepremium.abp_street_descriptor.language

AND addressbasepremium_cou.abp_street_descriptor.change_type != 'I' )

#Insert the new updated records and the new inserted records

INSERT INTO addressbasepremium.abp_street_descriptor SELECT * FROM addressbasepremium_cou.abp_street_
descriptor WHERE change_type != 'D';
```

The Street Descriptor table does not have the ability to hold historical records as it does not have a Logical Status Code attribute. Therefore, in order to capture the historical records you will need to create an archive table that is populated when records are either deleted or updated.

To apply COU to the LPI table (with archiving) the following code can be used:

#The following command creates an archive table of the records that are being updated and deleted from the existing Street table. If this table already exists you can simple use INSERT INTO rather than CREATE TABLE

```
CREATE TABLE addressbasepremium.abp_street_descriptor_archive AS SELECT * FROM addressbasepremium.abp_street_descriptor WHERE EXISTS (SELECT 1 FROM addressbasepremium_cou.abp_street_descriptor

WHERE addressbasepremium_cou.abp_street_descriptor.usrn = addressbasepremium.abp_street_descriptor.usrn

AND addressbasepremium_cou.abp_street_descriptor.language = addressbasepremium.abp_street_descriptor.language
AND addressbasepremium_cou.abp_street_descriptor.change_type != 'I' )
```

#The following command then deletes the records from the existing table which are either Updates or Deletions

```
DELETE FROM addressbasepremium.abp_street_descriptor WHERE usrn IN (SELECT distinct usrn FROM addressbasepremium_cou.abp_street_descriptor WHERE change_type != 'I');
```

#The following command then inserts the new Insert records and the new Updated records into the Street table.

```
INSERT INTO addressbasepremium.abp_street_descriptor SELECT * FROM addressbasepremium_cou.abp_street_descriptor WHERE change_type != 'D';
```

11 Further information

AddressBase Products pages:

http://www.ordnancesurvey.co.uk/oswebsite/products/addressbase-premium/index.html
http://www.ordnancesurvey.co.uk/oswebsite/products/addressbase-plus/index.html
http://www.ordnancesurvey.co.uk/oswebsite/products/addressbase/index.html

General information
www.ordnancesurvey.co.uk/contactus

General enquiries
+44 (0)8456 05 05 05

Textphone
+44 (0)23 8005 6146

This document has been screened in accordance with the requirements set out in Ordnance Survey's Equality Scheme. If you have difficulty reading this information in its current format and would like to find out how to access it in a different format (Braille, large print, computer disk or in another language), please contact us on: +44 (0)8456 05 05 05.

D08911 0713