QAS Batch

API Guide

Version 7, Revision 1, 2019

Contacts and Support

For resolutions to common issues, answers to frequently asked questions and hints and tips for using our products, contact our regional Support teams:

www.edq.com/documentation/contact-support

For more information about us and to get in touch:

www.edq.com

Contents

Introduction

What Is QAS Batch API?

QAS Batch API cleans the address records in your database by verifying them against the official postally-correct address files for the relevant country. Cleaned records are assigned a match result, based on the accuracy of the original address.

QAS Batch API does not directly access your file of address records. You must extract the records that you wish to check, pass them through QAS Batch API one at a time and, where appropriate, subsequently update your database.

QAS Batch API can make use of several datasets, each of which contains full address information for one country. You can choose to search on one or more of these datasets, depending on the addresses that you have. Some datasets are enhanced further with Additional Datasets, which allow you to access a variety of additional data, such as Names, Business or Utility information. For more information about available Additional Datasets for your data, see the relevant Data Guide.

In order to maximise the ease of integrating this product, it is supplied with the following interfaces and test harnesses:

- C API;
- C#.NET;
- Java.

To facilitate easy integration, Experian Data Quality also supplies detailed sample code in those languages.

These interfaces, test harnesses and sample code are for use on Windows systems only.

What Does This Guide Contain?

In this manual you can find information on:

1. **Installing QAS Batch API and Updating Data**

   The section beginning on page 5 describes how to install QAS Batch API, update data, and test your installation. It also provides information on licence keys.

   Running the test harness supplied with the API should verify that you have installed the API correctly. It will also give you an idea of what QAS Batch API can do, and the type of results it can produce. See page 18 for more information.

2. **How QAS Batch API Works**

   The section of the manual starting on page 21 describes how QAS Batch API searches on your addresses and the match codes it produces. Reading this should clarify the values that are returned by some of the API functions.

3. **Pseudocode Example**

   The pseudocode example on page 49 demonstrates a possible implementation of the API functions.

4. **Using the API Functions**

   The listing of QAS Batch API functions starts on page 54. You do not have to use all of the functions.

   It is recommended that you integrate the API in stages, beginning with the startup and shutdown functions, then adding the open and close functions, followed by address search and retrieval facilities. Any other functions can be added in the appropriate places. You should also make use of the system functions, especially **QAErrorMessage**. This function enables you to see the description of any errors that occur, and as such should be called after any function returns an error. A full list of error codes starts on page 179.

5. **Configuring your API**

   Before running your integrated API, you need to give QAS Batch API:

   - the name and location of your dataset(s);
   - the level of address cleaning you want to undertake;
   - the format of your output address;
   - the name of your log file, if there is one.

   This information should be specified in the configuration file. Details of how to do this start on page 147.

6. **Using QAS Batch API With Suppression Data**

   This section describes the benefits of using QAS Batch API with AUS or GBR Suppression data, and how to pay for its use.

Accompanying Documentation

This section provides a list of the documentation supplied with QAS Batch API and where it can be located.

Data Guide

A Data Guide is supplied with each dataset you purchase. This guide provides data installation instructions, dataset-specific information and search tips for each dataset, and should be used in conjunction with the other documentation supplied with QAS Batch API. Any functionality unavailable in QAS Batch API is also covered.

Under Windows, you have the option to install the Data Guide during data installation. The guide is installed to C:\Program Files\Experian\Data Guides by default. If you choose not to install the guide, it can be accessed from the Docs folder on the Data installation CD.

Under UNIX, you should copy across the Docs folder to a location of your choice.

API Help

An API help system is provided in the QAS Batch API program group in the **Start** menu. This help system provides similar information to that included in this manual.

Technical Support

Experian Data Quality provides three forms of Technical Support.

Web

If you encounter problems using QAS Batch API, that are not answered in the documentation, please visit the Experian Data Quality support website http://support.qas.com. Answers to questions about all aspects of Experian Data Quality products are contained within a FAQ section and a searchable knowledge base.

E-mail / Telephone

If you cannot locate the required information on http://support.qas.com, Experian Data Quality Technical Support can be contacted via e-mail or telephone. The Technical Support contact details for each local Experian Data Quality office can be found at http://support.qas.com/contact.

In order that your request can be dealt with as efficiently as possible, it would be helpful if you could have your Experian Data Quality account reference number and the version number of the software you are using to hand.

QAS Batch API Installation

Overview

Follow these steps to install QAS Batch API:

1. Install the product. See "Installing QAS Batch API" on page 9.
2. Once the product installation is complete, you can install Experian Data Quality data. See "Installing And Updating Data" on page 10.

System Requirements

To run QAS Batch API you will need the following:

Operating System - Windows	The following Windows operating systems are supported: • Server machines: • Windows Server 2016 64-bit • Windows Server 2012 R2 64-bit • Desktop machines: • Windows 7 Professional SP1 64-bit • Windows 10 Pro 64-bit For more information about downloading Service Packs, see http://support.microsoft.com.

Operating System - UNIX	The following UNIX operating systems are supported: • Sun Solaris 64bit (Sparc) - 10, 11 • IBM AIX 64 bit (Power 5) - 6.1 • Linux (x86-64) - kernel: 2.6.18; GCC: 4.1.2 • Linux (x86-64) - kernel: 2.6.32; GCC: 4.4.7
Available memory	1GB (minimum), 4GB (recommended)
Disk Space	At least 100MB of drive space is required to install the QAS Batch API program files. Further space is required to install the dataset files. Refer to the Data Guide supplied with your data for details about how much space you will need for each dataset. You also need to ensure that there is sufficient disk space for QAS Batch API to process your addresses. The space used depends on the size and type of input database, but as a general rule you should ensure that space equivalent to at least the size of your database is available.
Other	A CD/DVD drive (this is required for installation only, and only if you are installing your product/data via installation disks). An internet connection (if using Electronic Updates, or for DPV unlocking in the event of encountering a seed address when using the USA dataset); this does not have to be on the machine running QAS Batch API.

Licences

What Is A Licence Key?

A licence key is required for each combination of data and product that you purchase. Failure to enter a valid licence key means that the product will be unable to use the data. You should have already received licenses from Experian Data Quality for any products and datasets you have purchased. If you have not received them, please contact Technical Support. See "Technical Support" on page 4 for more information.

Adding A Licence Key

Windows Installer Users

If you have chosen to install/update your data using installation disks rather than Electronic Updates, the data installer prompts you to enter a licence key for each product / data combination for which you have purchased data. For example, if you are using both QAS Pro and QAS Batch API and have purchased GBR data for both products then you will be asked to enter a separate GBR licence key for each product.

See "Installing And Updating Data" on page 10 for more information.

Non-Windows Installer Users

If you are running QAS Batch API on UNIX, or you do not install data using the Windows installer utility, then you will need to manually edit qalicn.ini to insert the licence keys supplied with your data. Each key should appear on its own line, starting on the first character of the line.

> Ensure that you enter licence keys only for the product associated with the licence file you are editing.

Expiry Warnings

If a licence has expired, it is not possible to open an instance of the API (**QABatchWV_Open**).

You can use **QABatchWV_DataSetInfo** to view how long is left before the data expires.

Evaluations

Evaluation licence keys set time limits on the usage of data. To continue using the product and data after these time limits have been reached, you must purchase a full licence.

To upgrade from an evaluation licence to a full licence, contact Experian Data Quality Sales using the contact details provided at the beginning of this guide, or complete the Purchase Licences form on http://support.qas.com.When you have received your new licence key from Experian Data Quality, you should enter it in qalicn.ini.

Installing QAS Batch API

Windows

QAS Batch API has been supplied to you on CD-ROM, and comes with an installation program called setup.exe. When you run setup.exe, the QAS Batch API libraries and associated files are installed to the location of your choice (the default is C:\Program Files\Experian\QAS Batch API).

To run the installation program:

1. Insert the QAS Batch API CD into your CD-ROM drive.
2. Select **Run...** from the Start menu.
3. In the dialog box that appears, type **d:\setup**, where d is the drive letter for your CD-ROM drive, and press **Enter**. Once the installation program starts, follow the on-screen instructions to install the API.

UNIX

The QAS Batch API image contains both shared object and static library compiles of QAS Batch API. Copy the contents of the relevant folder from the image to the location of your choice, for example /opt/qas batch/.

You can choose to install QAS Batch API as a 32-bit or 64-bit version. You should select the version that is correct for your integration.

When you wish to run your integrated application, you should ensure that the following files are in the same directory as the application executable:

- country.ini
- qaworld.ini
- qawserve.ini
- qalicn.ini
- qalcl.dat
- static library or shared object of libqabwvcd
- 32-bit or 64-bit shared object of libCorrectA (USA or CAN dataset only)

Installing And Updating Data

Once you have installed the program, you can install Experian Data Quality data to use with QAS Batch API.

To ensure that your data is compatible, all data for a country must be the same version, and should be installed at the same time.

Each dataset has an expiry date and must be updated periodically. Experian Data Quality provides regular updates of the datasets as and when updated data is available. Each update should be applied promptly, otherwise the data may expire and the product will become unusable.

> Before you begin the data installation process, you should ensure that you have received all of the licence keys for each dataset you have purchased. During the installation you are prompted to insert the licence key for each dataset that you want to install.
>
> See "Licences" on page 7 for more information about licences.

Windows

To install or update data on Windows, we recommend that you use Electronic Updates, which automatically ensures that your Experian Data Quality products are using the latest data available. For more information about Electronic Updates, visit support.qas.com. If you have chosen to use Electronic Updates, see "Electronic Updates" on page 11.

Alternatively, to install or update data using any data installation disks that you may have received, follow these steps:

1. Insert the first Experian Data Quality Data CD or DVD into the relevant drive, click the **Start** menu and select **Run**.
2. In the dialog box that appears, type d:\setup, where d is the drive letter for your CD or DVD drive, and press **Enter**.
3. Once the installation program starts, follow the on-screen instructions to install or update your data.

4. If you have purchased additional datasets (such as Names or Suppression additional data) or DataPlus sets to enhance your address data, you should select the relevant data in the Data Setup dialog. Once the address data has been installed, you will be prompted to install all the additional data that you have selected.

This information is only relevant when using GBR data.

NCOA data

According to the terms of your third-party licence agreements, once you have performed an initial clean of your data using QAS Batch API, NCOA Update and NCOA Suppress data are tied to the machine that QAS Batch API is installed on.

If you do need to move these datasets to a different machine after you have performed an initial clean, you should contact Experian Data Quality Technical Support who will guide you through the process. See page 4 for Technical Support contact details for your region.

Telephone Preference Service (TPS) and Monthly Pointer data

If you wish to use Telephone Preference Service or Monthly Pointer data, you must install and use Electronic Updates (EU) to do so, due to the fortnightly data update cycle.

Electronic Updates

If you have chosen to install or update your data using Electronic Updates, follow these steps:

1. Install Electronic Updates (if you have not yet done so), following the on-screen instructions to install and run the EU client.
2. Use the EU client to install and/or update any datasets (including additional datasets) that you have purchased to use with QAS Batch API.
3. Once you have configured and run the EU client, all future data updates will automatically be downloaded to your computer.

For more information about how to use Electronic Updates and the EU client, refer to the Electronic Updates documentation.

UNIX

Installing New Data

The first time you install a dataset you must do the following:

1. Copy the data files from the data CD / DVD to a suitable location. If your dataset is supplied on more than one disk, then repeat for each disk, including any additional datasets associated with your dataset. See "About Data Files" on page 15 for more information about the data files you require.

   > If you transfer the data files using FTP you must transfer them as binary files, otherwise the data may be corrupted.

2. Navigate to /opt/qasBatch/ (or wherever you installed QAS Batch API) and open the qawserve.ini configuration file in a text editor such as vi.
3. Under the [QADefault] section add a line to the `InstalledData` setting, specifying the location of the data you just copied. The setting is in the format:

   InstalledData={dataset identifier},{path}

   For example, if you had just copied the GBR dataset to /opt/qasData/gbr/ the line would read:

   ```
   InstalledData=GBR,/opt/qasData/gbr
   ```

   For more information about this setting see page 151.
4. In the same section add at least one line to the `DataMappings` setting, to specify the combinations of additional datasets you wish to use. The setting is in the format:

   DataMappings={data mapping identifier},{dataset/group name},{dataset+additional datasets}

   For example, if you had the United Kingdom With Names additional dataset to enhance your GBR dataset you might add two lines, one without the additional dataset and one including it:

   ```
   DataMappings=GBR,United Kingdom,GBR
   +GBN,United Kingdom With Names,GBR+GBRNAM
   ```

   For more information about this setting see page 152.

5. This information is only relevant when using USA data.

   If you are installing the USA dataset you must also specify the location of the supplementary USA QAS Batch data and libraries which are supplied on separate disks to the USA data. In the same section of qawserve.ini, add the `CorrectADataLocUSA` setting and the `CorrectAAPILoc` setting. For example:

   ```
   CorrectADataLocUSA=/opt/qasData/USA
   CorrectAApiLoc=/opt/qasData/USA
   ```

   You do not need to use the `CorrectAApiLoc` setting if you installed the supplementary libraries to the same location as your core QAS Batch API libraries, or to a location specified in your environment variable.

   For more information about these settings, see "CorrectAApiLoc" on page 154 and "CorrectADataLocUSA" on page 154.

6. This information is only relevant when using CAN data.

   If you are installing the CAN dataset you must also specify the location of the supplementary CAN QAS Batch data and library which are supplied on separate disks to the CAN data. In the same section of qawserve.ini, add the `CorrectADataLocCAN` setting and the `CorrectAAPILoc` setting. For example:

   ```
   CorrectADataLocCAN=/opt/qasData/CAN
   CorrectAApiLoc=/opt/qasData/CAN
   ```

   You do not need to use the `CorrectAApiLoc` setting if you installed the supplementary library to the same location as your core QAS Batch API libraries, or to a location specified in your environment variable.

   For more information about these settings, see "CorrectAApiLoc" on page 154 and "CorrectADataLocCAN" on page 155.

7. Open the qalicn.ini configuration file in a text editor such as vi. Add all of your supplied licence keys for your purchased Experian Data Quality datasets to the end of this file. For more information see "Licences" on page 7.

Updating Existing Data

To update a previously installed dataset, overwrite your existing files with those provided on the data update CD / DVD. To ensure that your datasets are compatible, all data for a country must be the same version, and should be installed at the same time. The supplementary data and libraries for the USA and CAN datasets must also be copied from the QAS Batch data disks. The following section details the files which comprise each dataset.

If you transfer the data files using FTP you must transfer them as binary files, otherwise the data may be corrupted.

About Data Files

You have been supplied with at least one dataset. Each dataset that you have purchased comprises at least three files. The main dataset file has the .dts extension, and is accompanied by index files.

The files you receive for each dataset are:

```
<dataset identifier>.dts
<dataset identifier>.tpx
<dataset identifier>.zlx
<dataset identifier>.kfx  (datasets compatible with the Keyfinder engine only)
<dataset identifier>.zlb  (certain datasets only)
```

The dataset identifiers are three characters long and for most datasets are derived from the country's ISO code. For example, the UK dataset identifier is 'GBR', and the Australia dataset identifier is 'AUS'. For datasets that do not represent a particular country's address data, and therefore are not associated with an ISO code, the dataset identifier is a unique three character identifier. For example, the Gazetteer dataset identifier is 'LPG'.

Dataset identifiers are used throughout the API and its documentation as unique three character identifiers for datasets.

Additional datasets are available for some datasets. The additional datasets have the .ads extension and are supplied with index files. The filenames are derived from the dataset identifier with which they are associated. For example, the following files comprise GBR Business data:

```
gbrbus.ads
gbrbus.tpx
gbrbus.zlx
```

DataPlus sets are also available for some datasets and additional datasets. They have the .dap extension, and their filenames are derived from the dataset identifier. For example, the filename for the GBR Government DataPlus set is gbrgov.dap, and the filename for the AUS Mosaic Code DataPlus set is ausmos.dap.

This information is only relevant when using USA or Canada data.

The USA and CAN datasets require supplementary data and libraries for use in QAS Batch API. For Windows users, these files are configured automatically by the Windows installer. UNIX users must copy the data files manually from the ‘Data’ directory of the USA/CAN Batch data disks (along with the UNIX libraries that are supplied separately), and update their configuration files as described in "Installing New Data" on page 12.

Activating Metered Datasets

A click is the unit of measurement for a metered licence. Depending on how such a licence is set up, a click may be decremented from a meter when an address is matched in the dataset, or when data is appended to your matched records. For Suppression meters, a click may be decremented when an address record is suppressed or when a Suppression DataPlus flag is returned.

If you purchased a metered dataset (such as a Suppression additional dataset), you will need to activate these clicks before you can output any cleaning results from QAS Batch API.

For more information about metered datasets and how to use them, see "Appendix D: Analysing Costs of Suppression Data" on page 191.

Testing Your API Installation

QAS Batch API is supplied with a simple text-based application written in C, and can be used to verify that you have installed QAS Batch API correctly, and also demonstrate some of the API functionality. It is not intended to be used as a commercial application. For Windows 32-bit API installations it is known as batwv.exe, for Windows 64-bit installations it is known as batwv64.exe, and for UNIX installations it is known as batwv.

On UNIX, the main library must be accessible as a shared object. To ensure this is the case, you can register the library with the following steps (you will need to be logged in as **root**):

1. Navigate to /opt.
2. Copy libqapwccd.so (the main library) from the /lib directory in your installation location to /usr/lib.
3. Run **ldconfig** (or your system's equivalent).

QAS Batch API for Windows is also supplied with additional test harnesses written in other languages:

	32-bit	**64-bit**
Java	BatchTestHarness_Java.jar	BatchTestHarness64_Java.jar
C#.NET	BatchTestHarness_CS.exe	BatchTestHarness64_CS.exe

Although these versions are not described in detail in this manual, they can also be used to verify that you have installed QAS Batch API correctly.

> If you want to run either the 32-bit or 64-bit Java test harness on a 64-bit machine, the loaded Java virtual machine must be the same version as the test harness. If the virtual machine and the test harness are not the same version, the test harness will not run successfully.

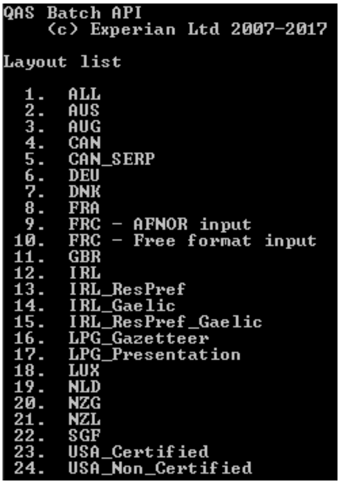
Running The Test Harness

The C test harness enables you to obtain a matching address and match code from an input address that you type in on the command line.

If you are using Windows, run the test harness from the QAS Batch API installation directory.

If you are using UNIX, go to the "apps" folder in the directory where the program files were installed and type **./batwv** at the command prompt.

The test harness appears, looking similar to this:



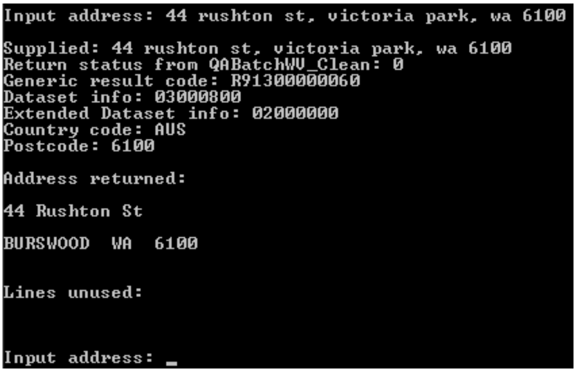
```
QAS Batch API
    (c) Experian Ltd 2007-2017

Layout list

  1.  ALL
  2.  AUS
  3.  AUG
  4.  CAN
  5.  CAN_SERP
  6.  DEU
  7.  DNK
  8.  FRA
  9.  FRC - AFNOR input
 10.  FRC - Free format input
 11.  GBR
 12.  IRL
 13.  IRL_ResPref
 14.  IRL_Gaelic
 15.  IRL_ResPref_Gaelic
 16.  LPG_Gazetteer
 17.  LPG_Presentation
 18.  LUX
 19.  NLD
 20.  NZG
 21.  NZL
 22.  SGF
 23.  USA_Certified
 24.  USA_Non_Certified
```

Once you have selected a layout from the list, try typing an address, separating each part from the next with a comma, and press **Enter**. For example:

    44 Rushton St, Victoria Park, WA 6100

The test harness returns various details, concluding with the matching address (if there is one).

For example, entering '44 Rushton St, Victoria Park, WA 6100' (with the Australia dataset) might return this:



```
Input address: 44 rushton st, victoria park, wa 6100

Supplied: 44 rushton st, victoria park, wa 6100
Return status from QABatchWV_Clean: 0
Generic result code: R91300000060
Dataset info: 03000800
Extended Dataset info: 02000000
Country code: AUS
Postcode: 6100

Address returned:

44 Rushton St

BURSWOOD  WA  6100


Lines unused:



Input address: _
```

The details shown are:

- the address you supplied;
- the return value from the function **QABatchWV_Clean** (see page 63);
- the generic match code assigned to the address (see page 29);
- the dataset-specific information, which comprises the dataset-specific information bits and the extended dataset-specific information bits (see the Data Guide supplied with your data for more information about these information bits);
- the dataset identifier of the dataset used;
- the postal code;
- the returned address;
- any unused components from the input address.

The supplied address is a duplication of the elements you typed in, while the returned address is a matched address as it is stored in the dataset.

QABatchWV_Clean returns 0 in this case, indicating that the function was successful.

A match code beginning R913 indicates that:

- the match success letter is R;
- the match confidence level is 9;
- the postal code action indicator is 1;
- and the address action indicator is 3.

Match success letter R means that a full address and postal code has been found, the 9 means that QAS Batch API is confident that it has found the correct address, the 1 shows that the postal code has not been changed and the 3 shows that either a partial or a full address has been returned. For more information see "Address Match Codes" on page 29.

The output postal code is 6100. The dataset identifier, which is a three-letter code that identifies the dataset, is AUS.

There are no unused components from the address you typed in.

The QAS Batch API may behave differently, depending on your current configuration settings.

The QAS Batch API Process

Before QAS Batch API can search on an address, it needs to know what level of searching to undertake, and how to return any matches that it finds. You specify these options in the configuration file.

The configuration file contains many settings which govern the basic processing that QAS Batch API performs, and allow you to define options such as the default dataset, cleaning options, and how the output address should look. The main configuration file used within the API for these settings is called qaworld.ini. For more information about configuration see page 147.

How QAS Batch API Matches Addresses

QAS Batch API goes through a complex process when it attempts to match your address against the dataset(s). Understanding the process helps you to get the most out of the QAS Batch API.

Matching Keys

This information is only relevant when using APR data.

If you are using QAS Batch API with the AddressBase® Premium dataset, QAS Batch API can perform key matching against your input data before carrying out the address matching process. This can potentially improve the confidence level of any address matches obtained. Key matching can be carried out against the following two types of data:

- Unique Property Reference Numbers (UPRNs);
- Unique Delivery Point Reference Numbers (UDPRNs).

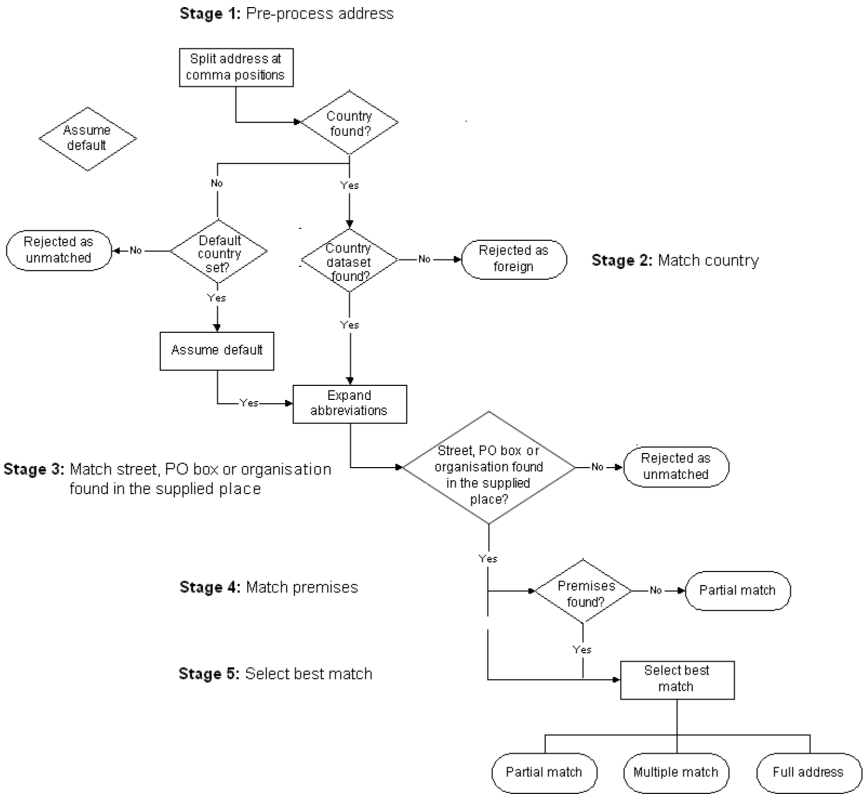
In order to match this information against AddressBase Premium data, you must first specify which fields in your input data contain UPRNs or UDPRNs. See "Setting The Input Address Format" on page 170 for information on how to do this. Once key matching has been completed, QAS Batch API begins the normal address matching process.

Address Matching

The QAS Batch API process consists of five stages:

- Stage 1: Pre-process address
- Stage 2: Match country
- Stage 3: Match Street, Organisation, PO Box and Place
- Stage 4: Match Premises
- Stage 5: Select Best Match

The diagram below summarises the QAS Batch API process.



QAS Batch API may return an address as unmatched if the place and street are matched, but the premises is not matched.

Stage 1: Pre-process Address

The first thing QAS Batch API does is attempt to put the input address into a standard format. The input address has been submitted as a single line, with address elements separated by commas. For example:

3 Mornington Mews, County Grove, London,SE5

QAS Batch API splits this address at the position of each comma so that the address looks like this:

3 Mornington Mews
County Grove
London
SE5

Stage 2: Match Country

Once it has completed its formatting, QAS Batch API tries to identify the country that the input address relates to. QAS Batch API does this by matching the contents of the last two lines against a list of countries, ignoring non-alphabetic characters.

If a country is identified in the input address, QAS Batch API goes on to verify that the relevant dataset is installed. If it is not, QAS Batch API will mark the address as 'Country not available' (see "Match Success" on page 30) and stop the search.

If no country is found in the address, then QAS Batch API tries to move on to the next stage using the default dataset. If no default dataset is set, QAS Batch API will reject the address as 'Unidentified country' and stop the search.

Stage 3: Match Street, Organisation, PO Box And Place

Depending on the country, QAS Batch API may expand street abbreviations, which means that all street descriptors such as 'Rd' or 'Ave' are expanded to 'Road' and 'Avenue', so that they match the descriptors in the dataset.

If you have specified that address fields occur on particular input lines, then QAS Batch API will use these to help decide which elements it can match with.

If you have not made any specifications of this type, QAS Batch API will make some assumptions, in particular that a place or a postal code will not occur in the first address field and that street elements will always occur before place elements. QAS Batch API will make one or more attempts to locate a valid sequence of street and place combinations in the address. At the same time, if QAS Batch API can locate PO box or organisation names along with a valid place, it will take these to be potential valid matches.

QAS Batch API will also break single words out of address lines in order to locate the best combination of elements for matching.

Stage 4: Match Premises

By this stage, QAS Batch API has matched the input address as far as the street. To find the full verified address it also needs to match the property information.

After matching a place and a street, QAS Batch API compares the property information in the input address against all the premises in the dataset for that street. If no match is achieved, the input address is marked as 'partial address found' (see "Match Success" on page 30).

> If your addresses contain property information, but the dataset you are cleaning against does not, QAS Batch API does not alter the property information; it retains it and returns it. Ensure that the addresses are formatted to your requirements before you write them back to your database.

Stage 5: Select Best Match

QAS Batch API now retrieves the full verified address and assigns it a 'quality' score by comparing it with the original input address.

During this comparison, QAS Batch API evaluates a number of matching rules and assigns the match a score. If there is more than one match, this process is repeated for all the matches. If there are two or more matches that have the highest score, QAS Batch API marks the input address as either 'Partial address' or 'Multiple match', depending on the matching rules that were passed.

The Returned Address

QAS Batch API returns matched addresses in the format that you specify. You can decide which address elements appear on which line, and which elements are capitalised or abbreviated. QAS Batch API always returns address elements using the string data type, even when there are fields which may only contain numeric data.

You define the output address format in the configuration file.

Some countries have several versions of an address element (for example, the Netherlands has NEN, TPG and Official street names). QAS Batch API uses the version supplied in the input to secure a match and returns the version configured in the output. For this reason, an address can be verified as correct even if there appear to be differences between the input and output addresses.

Retrieving DataPlus Information

DataPlus can provide a wide range of information relating to an address, as a supplement to the QAS Batch API. Currently, DataPlus information is only supplied with certain datasets; if you do not have it, skip the rest of this section.

DataPlus information is contained in datasets. Each piece of information relates to a locality, a postal code, or, when the data requires higher resolution, to the delivery point (letter box).

DataPlus handles the information in terms of a code and its related description (if there is one). For example, a dataset containing MOSAIC information might include one or more demographic details.

DataPlus details can only be viewed once you have selected and displayed a full address from QAS Batch API. For example, if you have the Australia dataset with the associated latitude and longitude DataPlus set, and you have configured QAS Batch API to return DataPlus information, a search on '314 miller st, north sydney, nsw 2060' might return the following address:

```
314 Miller Street,
NORTH SYDNEY NSW 2060
-33.8313 151.208
```

In this example, the latitude and longitude appear beneath the address. If you want to retrieve DataPlus information with your addresses, you should configure your address layout so that it contains lines specifically for DataPlus.

See the Data Guide supplied with your dataset for further information on DataPlus.

USA DataPlus and DPV

This information is only relevant when using USA data.

One of the requirements of CASS accreditation is that DPV functionality is active (see "Appendix F: Delivery Point Validation" on page 205). USPS requires all CASS-certified software to return a +4 code only when the address has been DPV-confirmed. If an address is not DPV confirmed, a +4 code will not be returned, and by extension, any DataPlus items you have configured as part of the address output format may not be returned either.

More DataPlus elements will be returned when using QAS Compatibility Formatting mode because an additional matching routine is attempted for addresses that do not DPV-confirm. For more information see the USA Data Guide.

Using QAS Batch With Suppression Additional Datasets

In order to achieve the most effective results when using QAS Batch API with Suppression data, we recommend following these guidelines:

- Before running your input data against Suppression data, carry out a standard address cleaning session on your data first. If you have purchased any other additional datasets (for example Names, or Utilities data) these can also be configured in this initial session.

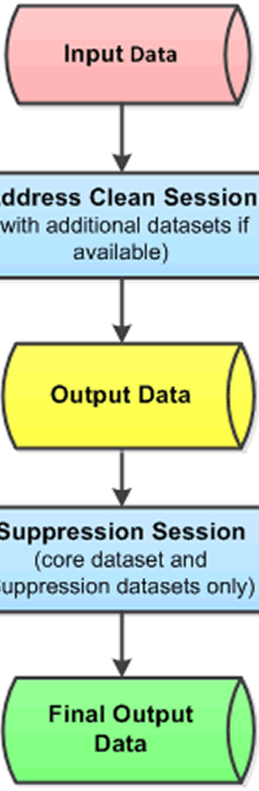
  Running a Suppression session against addresses that have already been cleaned and verified means that QAS Batch API is more likely to find good suppression matches.

- Do not run your input data against both Suppression data and other additional datasets during the same session. A Suppression session should use only the core country dataset, and your selected Suppression datasets.

  Running a Suppression session at the same time as other additional datasets may prevent QAS Batch API from finding good suppression matches in some cases.

The recommended workflow for using QAS Batch API with Suppression data is shown in the diagram below:

Suppression Workflow


Input Data
Address Clean Session
(with additional datasets if available)
Output Data
Suppression Session
(core dataset and Suppression datasets only)
Final Output Data

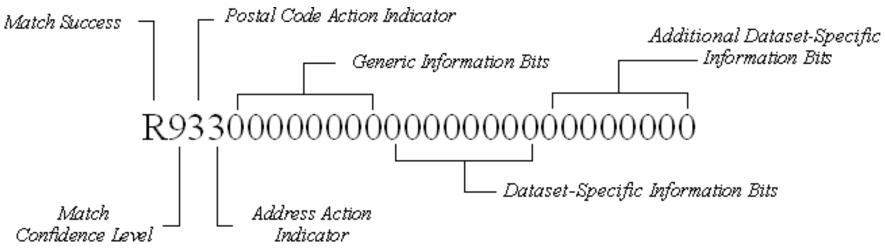
More information about configuring QAS Batch API, including setting up additional datasets can be found in "QAS Batch API Configuration" on page 147.

Address Match Codes

During a QAS Batch API search, the nature of processing and any changes made to the address are recorded in a match code. The match code is returned as a result of a call to the function **QABatchWV_Clean** (see page 63). The first four characters of the match code provide the following information:

Match Success	1 upper case letter. This specifies how well QAS Batch API matched the address.
Match Confidence Level	1 single digit. This tells you how accurate QAS Batch API thinks the match is.
Postal Code Action	1 single digit. This indicates any action that QAS Batch API has performed on the postcode.
Address Action	1 single digit. This describes what action has been performed on the address.
Information Bits	1 eight digit number provides general match information, including details of the matching process and reasons for the confidence level. These are the Generic Information Bits. 1 sixteen digit number provides dataset-specific match information. These are the Extended Information Bits.

This is what a full match code looks like:


Match Success
Postal Code Action Indicator
Generic Information Bits
Additional Dataset-Specific Information Bits
R9330000000000000000000000000
Dataset-Specific Information Bits
Match Confidence Level
Address Action Indicator

For more information about the generic information bits, see "Appendix G: Generic Information Bits" on page 209. For more information about the dataset-specific and additional dataset-specific information bits, see the relevant Data Guide.

Match Success

The letter at the beginning of the match code indicates how successfully QAS Batch API was able to match your input address to an address in an Experian Data Quality dataset.

The values of the match success letter are split into two ranges which indicate specific types of information:

A-D	The input address was not processed. The reason for this is specified by the letter returned.
K-S	The input address was processed, and the match quality is indicated by the letter returned.

The match success letter only indicates what type of matching address has been found in the data, it does **not** indicate whether this address is a good match for your input data. This information is indicated by the "Match Confidence Level" (see page 34)

If QAS Batch API returns a **Q** or **R** match, along with a match confidence level of **9**, you can be confident that it has found the right match.

Due to restrictions matching against Suppression datasets, all successfully suppressed records can be considered a confident match, regardless of the returned match code. The match code refers only to matches against the main address data or other, non-suppression datasets.

Match Success Letters A-D

A Unprocessed	Results could not be returned for the input address. This reflects an internal processing issue. For example, if DPV processing has been locked because you encountered a seed address, then all US addresses will return an **A** match. See page 205 for more information about DPV.

B Blank	This means that QAS Batch API could either find no data in the input address or too insignificant an amount of data to return an address.
C Country not available	This match letter is returned when your input address contains a country name and the appropriate dataset is not installed. For example, if you do not have the Australia dataset, this address would return a **C** match: 7 Speed Avenue, Five Dock NSW 2046, AUSTRALIA If you want to restrict QAS Batch API's ability to match against countries, use the `CountryRevert` configuration setting (see page 158).
D Unidentified country	A match letter of **D** is assigned to an address record when QAS Batch API is unable to ascertain the record's country of origin and no default country has been configured (see the `CountryBase` configuration setting on page 156).

Match Success Letters K-S

K No address or postcode could be derived	This match letter is used when QAS Batch API cannot find any data which matches your input address. This might occur if the input address does not contain a country name and does not match anything in the default dataset. For example if you processed "42 Durlston Square" against the GBR dataset, QAS Batch API would return a **K** match. This is because QAS Batch API cannot find any matching street names and has no other information (such as a locality or postcode) to search on.
L Postcode found, but no address could be derived	This match letter is returned if QAS Batch API derives a valid postal code from your input address, but no address information.

M Multiple addresses found, but no postcode	QAS Batch API returns this match letter if the input address matches more than one address in the dataset. For example, the following address finds four matches in the GBR dataset: 146 High Street, Cambridge Because the address exists in the localities of Sawston, Cottenham, Chesterton and Landbeach, QAS Batch API cannot determine which is the desired match. As all four potential matches have different postal codes and no single postal code can be returned, QAS Batch API marks the address as an **M** match.
N Multiple addresses found with postcode	This type of match is returned when QAS Batch API finds more than one matching address within a postal code. This is most likely to occur where a country's postal codes cover large areas, such as in Australia. For example, this Australian address has two possible matches, because it exists in the localities of Kingsholme and Ormeau: 25 Cliff Barrons Rd, QLD, 4208
O Partial address found, but no postcode	In this case QAS Batch API has found a partial address which matches your input. However, it cannot return a full postal code with it, because the partial address is covered by more than one postal code. This might occur if your input address has a missing or invalid property number. QAS Batch API cannot determine the correct property number, and returns as much of the address as it can. For example, in the street of this UK address, number 70 does not exist: 70 Glebe Road, Long Ashton, Bristol As no postal code is included in the input address, QAS Batch API does not know which of the street's two possible postal codes to return, and produces this output: Glebe Road, Long Ashton, Bristol

P Partial address found with postcode	QAS Batch API has found a partial address which matches your input. In addition, either the input postal code was valid, or QAS Batch API has managed to find a single postal code for the partial address. For example, if this Australian address is searched on: Robertson St, Sherwood QAS Batch API is able to add a postal code and state code, but the missing property number prevents it from returning a full address.
Q Full address found, but no postcode	This occurs when QAS Batch API finds a full address which matches your input data, but cannot find a full postal code to go with it. This is most likely to happen if a dataset does not include postal codes for every address.
R Full address and postcode found	In this case, QAS Batch API has made a full match, either by simply verifying a correct input address, or by locating a full address from partial input data. These examples all return R matches: 14 Carnaby St, London Grimmstr 5, 79848 Bonndorf Sintelweg 10, 9364 Nuis 19 Meyer Place, Melbourne, Victoria 3000 However, an R match only signifies that a full address and postal code have been returned; it does not necessarily mean that the address is the one you want. You can gauge the likelihood of a correct match from the match confidence level.

S Address matched to one or more Suppression datasets	This match code is only applicable when QAS Batch API has matched an address against GBR or AUS Suppression data. When this match code is returned, all returned address information and information bits will be cleared other than the suppression information bit. If one or more DataPlus elements are configured in the output layout, this match code is no longer returned, and instead all address and suppression infobits are returned, including the suppression information bit. Due to restrictions matching against Suppression datasets, all successfully suppressed records can be considered a confident match, regardless of the returned match code. The match code refers only to matches against the PAF or other, non-suppression datasets.

Match Confidence Level

The first digit in the match code indicates how confident QAS Batch API is about the match it has returned.

There are three levels of confidence: low, intermediate and high. As the completeness of the returned address is determined by the match success letter, QAS Batch API could return an **R** match with low confidence, indicating that although it has found a complete and correct address, it is not sure that it is the same address as the input.

Confidence is determined by the matching rules. Low confidence indicates that the essential matching rules were not satisfied. Intermediate confidence shows that the less important rules were not satisfied, or another check failed (for example, the input address is not in the expected order).

0: Low Confidence

QAS Batch API sets the confidence level to 0 if it finds a match which differs considerably from the input address. For example, take this UK address:

Rich & Carr, LE1 9GX

QAS Batch API returns its nearest guess (Rich & Carr, PO Box 15, Leicester, LE1 9GX). As this is a full address, it is given an **R** match success letter. However, as the input address did not specify PO Box details, QAS Batch API is not confident that this is the right match.

5: Intermediate Confidence

Confidence level 5 is returned when QAS Batch API is reasonably sure that it has found the right match. This might occur if the input address is slightly inaccurate. Consider this UK input address:

> Churchill Green House, Churchill Green, Churchill, Winscombe,
> Avon, BS25 5QH

In this example, the building name is incorrect (it should be Churchill House). However, QAS Batch API is able to find the correct address. Only the variance in building name prevents a high confidence match.

> If a record has the confidence reduced to Intermediate for more than one reason, this does not reduce it further, and it remains as Intermediate confidence.

9: High Confidence

QAS Batch API returns a 9 when it is sure that the output address matches the input data. This happens when an input address is fully accurate, or when partial address data is detailed enough (for example, exact property number, street and locality) to have the remaining address details appended. Consider this UK input address:

> Castle Gayer Cottage,Leys Lane,Marazion,Cornwall,TR17 0AQ

In this example, the address is spelt correctly and is found in the data exactly as typed. A High Confidence match is returned.

Postal Code Action Indicator

The second digit in the match code signifies the action that QAS Batch API has performed on the postal code.

There are four possible values for this digit:

Value	Description
3	The existing postal code has been corrected.
2	A new postal code has been added.
1	The existing postal code was already correct.
0	No action was taken.

Address Action Indicator

The third digit in the match code tells you what QAS Batch API has done to the address.

There are three possible values for this digit:

Value	Description
3	Part of or a whole address was returned. The amount of address is signified by the match success letter.
2	The existing address was enhanced. No significant information has been removed, but some information has been added.
0	No action was taken as the supplied address was not matched.

Information Bits

The hexadecimal information bits provide details of each match made by QAS Batch API. They consist of:

- an eight digit number which provides generic information;
- a sixteen digit number which provides extended information. This information is dataset-specific.

Generic Information Bits

The generic information bits provide detailed information on how well an address match conforms to the QAS Batch API matching rules. See "Matching Rules" on page 38 for more information.

For a full list see "Appendix G: Generic Information Bits" on page 209.

Extended Information Bits

Extended information bits are dataset-specific. They are added together in the same way as generic information bits. Refer to the Data Guide supplied with your data for further information about dataset-specific information bits.

Matching Rules

This section provides a guide to the considerations that take place when QAS Batch API is matching addresses. There are a number of predefined rules governing the results that the QAS Batch API produces. These are listed below:

- Generic matching rules
- Essential matching rules
- Preferred matching rules
- Close matching rules
- Acceptance matching rules
- Further rules
- Dataset-specific matching rules

Information on which matching rules have been applied to an address is supplied using information bits.

Generic Matching Rules

This core set of rules applies to the matching process for all countries:

- Elements must roughly occur in the expected predefined order (the order is defined in each country's data files).
- Any numbers appearing before the place elements in an input address should be matched.
- Numbers should associate correctly with accompanying elements. For example, in a UK address the building number is expected before the street name.
- At least one place element should be found in the input address.

If these rules are not satisfied, match confidence will be reduced to intermediate unless the rule is specifically suppressed in the particular country's dataset. For example, if a place is not supplied for a UK address, the match confidence will not be reduced if a match can be made by using the supplied postcode. Normally, postal codes represent larger areas, and confidence would be reduced in those circumstances. Regardless of any change in match confidence, the information bits set within the generic match code for a returned address indicate the rules that have failed.

Essential Matching Rules

These rules state the criteria which must be satisfied. If one of these rules is not met, a match of low confidence will be returned. These rules include conditions such as:

- There must be an input element defining the location of the address (a town, a postal code, etc).
- There must be an element defining the property information.

Preferred Matching Rules

These rules are criteria which should ideally be satisfied. These will be slightly stricter than the Essential rules, for example:

- A PO Box number must be matched if one exists in the dataset.
- There must be some form of match with a street name if one exists in the dataset.
- The postal code must match if none of the place elements in the dataset has matched.

If any one of these rules is not satisfied, confidence in the address matched will be intermediate at best.

Close Matching Rules

This is an additional set of rules which affect the confidence of a match to the address under consideration. Strictness is roughly equivalent to the Preferred rules, for example:

- Both the PO Box number and the corresponding place must match.
- At least one of the street descriptor and the locality elements must match.
- The sub-premises address element must make an exact match.

If one of these rules is not satisfied, confidence in the address matched will be intermediate at best.

Acceptance Matching Rules

This is a set of rules applied by QAS Batch API at its address acceptance stage, just before a final address is returned. These rules are only defined for some datasets, and specify the strictest final criteria that an address match must satisfy. These rules are only used when confidence has not previously been reduced for any other reason. Example rules are:

- It was not necessary to change more than one character in the supplied street name to obtain a match.
- All of the supplied multiple place and postcode-level elements matched.

If any one of these rules fails, confidence will be reduced to intermediate.

Further Rules

There is a further set of rules which are inherent to the code for all countries. These rules are dataset independent, and all will cause the confidence to be reduced to *Intermediate* if not satisfied.

- Elements must occur in a predefined expected order (the order is defined in each dataset)
- All numbers appearing before the place elements in the input address must be matched
- Numbers must associate correctly with the accompanying elements
- At least one place element must be found in the input address.

The information bits returned with an address indicate the rules that have been enforced.

Dataset-Specific Matching Rules

Unmatched Input Text

This information is only relevant when using GBR or APR data.

When using one of the datasets listed above, QAS Batch API will downgrade match success scores and confidence levels if it is unable to match some text in the input address. This is designed to prevent misleading high confidence matches being made to address records that share only a certain amount of text in common with the input address.

Consider using QAS Batch API with United Kingdom or AddressBase® Premium data with the following input address:

> Quick and Speedy Dry Cleaning Ltd, 2-3 Clapham Common North Side, London, SW4 0QL

Part of that input address would match to part of the following, postally correct address:

> Experian Ltd, George West House
> 2-3 Clapham Common North Side
> London
> SW4 0QL

When using other datasets, the equivalent result would be a full address match with intermediate confidence. But when using one of the datasets listed above, the match is not used due to the amount of unmatched input text. Instead we receive a result similar to this:

> George West House
> 2-3 Clapham Common North Side
> London
> SW4 0QL

The result is a **partial address match** which only includes elements from the input address that can be matched with **high confidence**.

Additional Dataset-Specific Matching Rules

Since QAS Batch API is a multi-country address matching product, additional rules are tailored for each dataset and are embedded in the data.

API Function Reference

This section introduces the QAS Batch API functions. It consists of the following sections:

- Data Types

  This section explains the QAS-specific data types that define the parameters that the functions take.

- Multithreading Example

  This section gives information about running several 'threads' of the application simultaneously.

- Pseudocode Example

  This section gives a programming language independent example of a QAS Batch API implementation.

- API Functions

  This section details the API functions in alphabetical order and lists them in groups according to the type of function. It also provides a list of replaced functions.

Data Types

There are a few QAS-specific data types which appear in the API and need some explanation. These types define the parameters that the functions take and values they return. They can be split into three categories:

- the values that are returned by the functions;
- the parameters that go into the functions;
- the parameters you get out of the functions.

Equivalent C data types are shown, but additional type modifiers are applied for certain environments, such as Windows.

Function Return Values

QAS Batch API data type	Description	Equivalent C data type
INTRET	integer	int
LONGRET	long integer	long
VOIDRET	no return value	void

Parameters (Input)

QAS Batch API data type	Description	Equivalent C data type
STRVAL	string	char *
INTVAL	integer	int
LONGVAL	long integer	long
VOIDARG	no arguments	void

Parameters (Output)

QAS Batch API data type	Description	Equivalent C data type
STRREF	string	char *
INTREF	integer	int *
LONGREF	long integer	long *

Calling Functions From Languages Other Than C

While C is the language most commonly used when working with these API functions, it is perfectly possible to work in other programming environments. However, there are a few points which you should note. These are:

- Passing by Value or by Reference;
- NULL Termination;
- Padding.

Passing By Value Or By Reference

In normal C programming, function parameters can be passed either by value or by reference.

You must pass a parameter in the way the function expects you to pass it. If you pass a parameter by value when the function is expecting it to be passed by reference then this might crash your program and will certainly produce incorrect results.

- Strings are always passed by reference, whether they are input or output parameters.
- Numbers are passed by value when they are inputs to the function. They are passed by reference when they are outputs from the function.

NULL Termination

QAS Batch API is written in the C programming language. In C, all strings are expected to be terminated with a NULL. The NULL character is the absolute character 0 (zero), not ASCII '0'.

For the QAS Batch API functions, all parameters of type STRVAL must be NULL terminated. Furthermore, all return parameters of type STRREF will be NULL terminated.

In BASIC, for instance, string termination can be achieved by appending the NULL characters to all the strings, as in this example:

```
MyString$ = "Hello" + Chr$(0)
```

Visual BASIC for Windows automatically ends all strings with a NULL character, so performing this termination might not be necessary.

After an API function has returned a string, it might be necessary to strip off the NULL character if it cannot be handled by the calling language.

In BASIC this could be achieved as follows:

```
nullOffset=INSTR(retBuffer$, CHR$(0))
IF nullOffset>0 THEN retBuffer$=LEFT$(retBuffer$, nullOffset -
1)
```

Padding

When an API function returns a result, it writes a NULL-terminated result string into a buffer. You are responsible for creating this buffer. You must ensure that it is big enough to hold any string which the function is likely to return.

Consider, for example, the programming language BASIC. In BASIC, strings are not normally stored in fixed length memory blocks. Rather, they occupy only the minimum amount of memory needed to store their value, which changes as the string changes.

Therefore, before you pass a string into a function you must first 'pad' it out. This means adding extra characters to the string in order to force the system into allocating enough memory to hold any possible return string.

In BASIC you might pad a string with two hundred '+' characters, like this:

```
retBuffer$ = STRING$(200, "+")
```

Alternatively, you could create a string with two hundred space characters like this:

```
retBuffer$ = SPACE$(200)
```

Example Of Data Types

The example below uses the function **QABatchWV_DataSetInfo**.

This is how the function prototype looks in the documentation:

```
INTRET QABatchWV_DataSetInfo
  (INTVAL viHandle,
  STRVAL vsIsoCode,
  INTREF riDaysLeft,
  INTREF riDataDaysLeft
  INTREF riLicenceDaysLeft
  STRREF rsVersion,
  INTVAL viVerLength,
  STRREF rsCopyright,
  INTVAL viCopyrightLength);
```

The parameters *viHandle* and *viVerLength* are inputs to the function (in the form of integers) and thus are passed by value. The parameter *rsVersion* is an output parameter (in the form of a string), and consequently it is passed by reference. Similarly, the parameter *riDaysLeft* is also passed by reference as it is output by the function in the form of an integer.

In addition, **QABatchWV_DataSetInfo** returns a status value which indicates either the successful execution of the function, or else failure - via an error code.

This function can be written in native C as:

```
int QABatchWV_DataSetInfo
  (int viHandle,
  char *vsIsoCode,
  int *riDaysLeft,
  int *riDataDaysLeft,
  int *riLicenceDaysLeft,
  char *rsVersion,
  int viVerLength,
  char *rsCopyright,
  int viCopyrightLength);
```

On 32-bit system, Visual BASIC for Windows would declare this function as:

```
Declare Function QABatchWV_DataSetInfo Lib "QABWVED.DLL"
  (ByRef viHandle As Long,
  ByRef vsIsoCode As String,
  ByRef riDaysLeft As Long,
  ByRef riDataDaysLeft As Long,
  ByRef riLicenceDaysLeft As Long,
  ByRef rsVersion As String,
  ByRef viVerLength As Long,
  ByRef rsCopyright As String,
  ByRef viCopyrightLength As Long)
  As Long
```

On 64-bit system, Visual BASIC for Windows would declare this function as:

```
Declare Function QABatchWV_DataSetInfo Lib "QABWVGD.DLL"
  (ByRef viHandle As Long,
  ByRef vsIsoCode As String,
  ByRef riDaysLeft As Long,
  ByRef riDataDaysLeft As Long,
  ByRef riLicenceDaysLeft As Long,
  ByRef rsVersion As String,
  ByRef viVerLength As Long,
  ByRef rsCopyright As String,
  ByRef viCopyrightLength As Long)
  As Long
```

Multithreaded Integrations

It is recommended that multithreading is not implemented for QAS Batch API integrations intended for USA address cleaning only. For more information, see Multithreading Considerations.

Multithreading is the ability of an application to maintain several execution 'threads' of the same program in memory. This is a highly efficient method that may be employed by an application to perform several concurrent tasks with the minimum duplication of system resources.

QAS Batch API supports up to 32 separate API instance handles (see **QABatchWV_Open**), each with up to 8 related search handles (see **QABatchWV_Clean**). Attempting to use any more threads or instances will cause QAS Batch API to return an error.

QAS Batch API uses additional RAM for each API instance and search handle used. The amount required varies depending on which countries and datasets you are using. You will need to ensure you have enough available memory to use your required number of API instance handles and related search handles. If there is insufficient memory available, QAS Batch API will return an ‘out of memory’ error. To remedy this, you will need to reduce the number of instance handles or search handles until this error is no longer returned. Note that if you are using the 32-bit version of QAS Batch API, the operating system will limit the amount of RAM to a maximum of 4GB per application.

When using QAS Batch API with more than one execution thread, there are limitations in using the handles issued by the API. It is currently the integrator's responsibility to enforce these rules, and instability or incorrect results may occur if this advice is ignored.

1. API Instance Handles

   You may use **QABatchWV_Clean** simultaneously with multiple threads and the same API instance handle. Any other QAS Batch API function that accepts an API instance handle may only be used by one thread for each API instance handle at any one time.

2. Search Handles

   Any search handle may only be used by one thread at any one time.

Pseudocode Example Of QAS Batch API

This section provides a conceptual overview of how a program using the QAS Batch API works. The pseudocode used is programming language independent.

The example below uses many of the QAS Batch API functions, so that you can see how they work together. In practice, however, you will not need to use every function.

The pseudocode does not include all of the system functions. When using the API within your application, you will probably want to use the function **QAErrorMessage** after every function call in case an error occurs.

The conventions within the pseudocode are as follows:

Convention	Meaning
/ Comment */*	Italic text between asterisks and forward slashes denotes a comment.
`[`**`QABatchWV_DataSetInfo`**`]`	The functions which relate to each part of the pseudocode appear in bold type on the right hand side of the page.
`[`**`QABatchWV_Open`**` (Close)]`	Some API functions are 'paired', i.e. when a function is called, its pair must also be called at some point. When a paired function is used in the pseudocode, its pair appears in brackets directly after the function name.

Pseudocode Listing

```
/* Before calling any function in the QAS Batch API , you must initialise it with
QABatchWV_Startup. If this function is called successfully, you can move on
to other functions. */
Initialise API                             [QABatchWV_Startup
                                           (Shutdown)]
If initialisation failed
  get textual description of error         [QAErrorMessage]
  display error
  shut down API                            [QABatchWV_Shutdown]
  exit procedure
end if
/* Once the API is initialised, you can either call the non-search related
functions, or open a search session. In this case, the program gets a list of
available layouts and asks the user to select one. */
Get number of available layouts            [QABatchWV_LayoutCount]
for each layout
  retrieve layout name                     [QABatchWV_GetLayout]
  display layout name
end for
Ask user to select one of displayed layouts
/* A layout contains information such as the available datasets, the default
dataset, and the output address format. Now that the user has chosen a
layout, the program moves on to open an instance of the API, in order to
perform searches. */
```

```
Open an instance of the API              [QABatchWV_Open
                                         (Close)]
/* The open might fail for various reasons. The most common reason is that a
specified dataset is not installed. */
If open failed
  get textual description of error       [QAErrorMessage]
  display error
  close API instance                     [QABatchWV_Close]
  shut down API                          [QABatchWV_ShutDown]
  exit procedure
end if
/* You can open multiple QAS Batch API instances as required, but you must
ensure that each instance is closed with QABatchWV_Close before calling
QABatchWV_ShutDown. */

Repeat while there are more searches to perform
/* The following section is only required where USA specific Delivery
Point Validation functionality is active: */
  If USA DPV component is locked         [QABatchWV_DPVState]
    get DPV lock code length             [QABatchWV_
                                         DPVGetCodeLength]
    get DPV lock code                    [QABatchWV_DPVGetCode]
    display lock code
    Ask user for DPV unlock key
    submit DPV unlock key                [QABatchWV_DPVSetKey]
  end if
/* End of USA-specific pseudocode*/
/* The program is now ready to perform address searches. The address must
be supplied to the search function, preferably with each address element
separated by a comma. Where the address comes from is up to the user: it
could be typed in manually, read from a database, or supplied by another
program. */
  get input address
  perform a search                       [QABatchWV_Clean
                                         (EndSearch)]
```

/ On completion of a search, the following information is returned: a 28-character match code, a postal code (if successfully matched), and a dataset identifier indicating which dataset the address was matched against. What happens next depends on how you want to use this information. You could accept just the postal code, the complete matched postal address or nothing. The decision you make will almost certainly be based on the match code. For this example, the program assumes you wish to have the full matched postal addresses returned for 'R' and 'Q' match types with a confidence level of 9. In addition for 'P' and 'N' match types that have a confidence level of 5 or 9, you wish to accept just the postal code. */*

```
  if match success letter = 'R' or 'Q' and confidence level
  = 9
```

/ With an R or Q match code, you are happy to take the full address. Therefore, you find out how many lines are in this address and read each line one at a time. */*

```
    count lines in matched address   [QABatchWV_
                                     FormattedLineCount]
    for each line in the matched
    address
      retrieve address line          [QABatchWV_
                                     GetFormattedLine]
    end for
```

/ Now you do something with this matched address. This could be storing it in a database or just displaying it on screen. This program does the latter. */*

```
    Display retrieved address
  Else if match success letter = 'P' or 'N' and confidence
  level = 9 or 5
```

/ In this case, you are only happy to accept the postal code returned by the search function. */*

```
    display returned postcode
  Else
    display message "No acceptable match"
  end if
```

/ You have done the search and processed the results. You now need to clear the search results, ready for the next search. */*

```
  free resources used by search     [QABatchWV_EndSearch]
```

/ The following section is only required where USA-specific Delivery Point Validation functionality is active: */*

```
  If USA additional dataset-specific information bits
  0x00000002 (DPV Disabled) or 0x00000004 (DPV seed
  address) set for match
```

```
    get DPV lock code length         [QABatchWV_
                                     DPVGetCodeLength]
    get DPV lock code                [QABatchWV_DPVGetCode]
    display lock code
    Ask user for DPV unlock key
    submit DPV unlock key            [QABatchWV_DPVSetKey]
  end if
```
/ End of USA-specific pseudocode /**

/ When you require no more searches, you need to close this instance of the API and shut down in order to exit cleanly. */*

```
end repeat
Close instance of API                [QABatchWV_Close]
Shut down API                        [QABatchWV_Shutdown]
```

QAS Batch API Functions

This section contains an alphabetical listing of the API functions. For each function there is a brief explanation of what it does, followed by its prototype, parameters, return value and any comments on its functionality.

The functions can be loosely split into the following groups:

- general functions, which open and close the API and provide program information
- system functions, covering the whole library system to handle errors and usage details
- search functions, which send the input addresses to the API and optionally provide feedback
- retrieval functions, which return the number and contents of formatted and unused address lines
- USA-specific DPV functions (see "Appendix F: Delivery Point Validation" on page 205 for more information).
- suppression-specific functions (refer to the Australia With Suppression Additional Data Guide or the United Kingdom With Suppression Additional Data Guide for more information about Suppression).

There is also a list of replaced functions, for reference by users of previous versions of QAS Batch API.

General Functions

These functions open and close the API and provide program information.

QABatchWV_Startup	page 135	Initialises the API.
QABatchWV_Shutdown	page 134	Closes down the API.
QABatchWV_LayoutCount	page 120	Retrieves the number of available configuration layouts.
QABatchWV_GetLayout	page 111	Retrieves the name of a configuration layout.
QABatchWV_ChangeLayout	page 61	Changes the layout to a custom one.
QABatchWV_Open	page 129	Opens an instance of the API.
QABatchWV_Close	page 66	Closes an instance of the API.

QABatchWV_CountryCount	page 78	Returns the number of available datasets.
QABatchWV_GetCountry	page 98	Returns the name of a dataset.
QABatchWV_DataSetInfo	page 82	Returns details of a dataset.
QABatchWV_DataSetCount	page 80	Retrieves the number of DataPlus sets, additional datasets and keyfinder sets that are associated with a dataset.
QABatchWV_GetDataSet	page 100	Retrieves the DataPlus sets, additional datasets and keyfinder sets for a dataset.
QABatchWV_LicenceInfoCount	page 127	Returns the number of lines of licensing information available for a specified dataset.
QABatchWV_GetLicenceInfo	page 113	Returns a specified line of licensing information.
QABatchWV_GetDPFieldCount	page 102	Retrieves the number of DataPlus fields that are associated with a given DataPlus or Additional dataset.
QABatchWV_GetDPFieldName	page 107	Retrieves the code and name for a specific DataPlus field for a dataset.
QABatchWV_GetDPFieldInfo	page 104	Retrieves information for a specific DataPlus field for a dataset.

System Functions

The QAS Batch API also includes four low-level system functions, which cover the whole library system and are common across all QAS product APIs.

QAErrorMessage	page 142	Expands a numeric error code into a simple text explanation of that error.
QAErrorHistory	page 139	Provides more detailed information about an error.
QAErrorLevel	page 141	Indicates the severity of an error and whether you should take action on it.

QASystemInfo	page 143	Lists system usage details, such as what resources the API has taken from your operating system.

Search Functions

These functions send the input addresses to the API and return results, and free resources used in the search.

QABatchWV_Clean	page 63	Performs a search on an input address.
QABatchWV_EndSearch	page 92	Frees resources used by a search.

Retrieval Functions

These functions return the number and contents of formatted and unused address lines.

QABatchWV_LayoutLineCount	page 122	Returns the number of address lines available in the current address layout.
QABatchWV_LayoutLineElements	page 124	Returns a description of the elements fixed to a particular line of the address layout.
QABatchWV_FormattedLineCount	page 94	Returns the number of formatted lines that a search has resulted in.
QABatchWV_GetFormattedLine	page 109	Retrieves a single address line.
QABatchWV_UnusedLineCount	page 137	Returns the number of unused lines from the input address.
QABatchWV_GetUnusedInput	page 117	Retrieves an unused address line.
QABatchWV_GetMatchInfo	page 115	Retrieves detailed match information.

USA-Specific DPV Functions

This information is only relevant when using USA data.

The QAS Batch API includes various functions that are specific to the DPV system. You only need to integrate these functions if you do not intend to use the DPV Unlock Utility supplied with QAS Batch API:

QABatchWV_DPVGetCode	page 84	Returns a DPV lock code when a 'seed' address has been encountered.
QABatchWV_DPVGetCodeLength	page 86	Returns the length of the DPV lock code when a 'seed' address has been encountered.
QABatchWV_DPVGetInfo	page 87	Returns the information about the DPV 'seed' address which is required by the USPS to issue an unlock key.
QABatchWV_DPVSetKey	page 89	Allows an unlock key to be set where DPV functionality is disabled.
QABatchWV_DPVState	page 90	Determines the state of the DPV system.

Suppression-Specific Functions

This information is only relevant when using GBR or AUS data with additional Suppression data.

The QAS Batch API includes eight functions that are specific to the use of Suppression data. If you are using QAS Batch API with Suppression data for the first time, Experian Data Quality recommends that you integrate these functions:

QABatchWV_GetAuditCode	page 96	Extracts a text-based audit code from the counters file on the disk where QAS Batch API With Suppression is installed.
QABatchWV_CompareAuditCode	page 68	Compares the amount of clicks used between two audit codes.
QABatchWV_CounterOpen	page 72	Opens a counter tied to the specified instance of QAS Batch.
QABatchWV_CounterReport	page 74	Creates a report on click usage and address cleaning since the counter was opened.

QABatchWV_ CounterReportLength	page 76	Returns, in bytes, the size that the XML report would be if created.
QABatchWV_CounterClose	page 70	Closes an open counter by handle.
QABatchWV_ApplyUpdateCode	page 59	Populates a counters file with post-pay meters for each supported Suppression set.
QABatchWV_RunMode	page 132	Switches QAS Batch in or out of Estimate mode.

Suppression data is only compatible with QAS Batch API version 🇬🇧 6.10 and later or 🇦🇺 7.05 and later.

Replaced Functions

These functions should no longer be used by your application, although the QAS Batch API retains them for backwards compatibility.

QABatchWV_Search	See **QABatchWV_Clean** on page 63 for the replacement function.
QABatchWV_GetUnusedLine	See **QABatchWV_GetUnusedInput** on page 117 for the replacement function.
QABatchWV_DataInfo	See **QABatchWV_DataSetInfo** on page 82 for the replacement function.
QABatchWV_GetLayoutLine	See **QABatchWV_LayoutLineElements** on page 124 for the replacement function.

QABatchWV_ApplyUpdateCode

This information is only relevant when using GBR or AUS data with additional Suppression data.

Populates a counters file with post-pay meters for each supported Suppression set. May also be used if the existing counters file has become corrupted or to add extra Suppression DataPlus sets.

Pre-call Conditions

The API must be initialised. No specific instances of the API need to be running.

Prototype

```
INTRET QABatchWV_ApplyUpdateCode
   (STRVAL vsUpdateCode);
```

Parameters

Argument	Explanation
vsUpdateCode	String containing counter update code.

Return Values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.

Invalid copy control code:	The update code that has been entered is incorrect. Check that the code you entered is identical to the one that you were supplied with by Experian Data Quality, and re-enter if necessary. If the error persists, contact **Experian Data Quality Technical Support**.

Related Functions:

QABatchWV_GetAuditCode

QABatchWV_ChangeLayout

Changes the layout to a custom one, provided as an argument. Also changes the ADS list. The new layout and ADS list are provided by using strings rather than sections in the .ini file. The country of the new layout cannot be changed by this function.

Pre-call conditions

The API must be initialized and a specific instance should have been started with **QABatchWV_Open**.

Prototype

```
INTRET QABatchWV_ChangeLayout
  (INTVAL viHandle,
  STRVAL vsLayout,
  STRVAL vsADSList);
```

Parameters

Argument	Explanation
viHandle	Handle for this instance of the API.
vsLayout	String containing new layout (with '\n' at the end of each line). "!" means leave the layout unchanged.
vsADSList	List of ADS sets (comma-separated string). "!" means leave the list of ADS sets unchanged.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_ Startup** must be successfully called prior to this function.

Bad handle:	The handle passed to the parameter *viHandle* is not valid. **QABatchWV_Open** must be successfully called prior to this function, and the handle returned in *viHandle* should be used.
Format error:	The layout or ADS list was in incorrect format.
Different country:	The country of the layout was different than the current one.

Related Functions:

QABatchWV_GetLayout

QABatchWV_LayoutCount

QABatchWV_Clean

Performs a search on the specified input address.

Pre-call conditions

The API must be initialised and a specific instance should have been started with **QABatchWV_Open**.

Prototype

```
INTRET QABatchWV_Clean
  (INTVAL viHandle,
  STRVAL vsSearch,
  INTREF riSearchHandle,
  STRREF rsPostcode,
  INTVAL viPostcodeLength,
  STRREF rsIsoCode,
  STRREF rsReturnCode,
  INTVAL viReturnLength);
```

Parameters

Argument	Explanation
viHandle	Handle for this instance of the API.
vsSearch	Search string.
riSearchHandle	Handle returned by search.
rsPostcode	Postal code returned from search.
viPostcodeLength	Maximum length of buffer to receive returned postal code.
rsIsoCode	Dataset identifier.
rsReturnCode	Match code brought back for the input address.
viReturnLength	Maximum length of buffer to receive match code.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_ Startup** must be successfully called prior to this function.
Bad handle:	The handle passed to the parameter *viHandle* is not valid. **QABatchWV_Open** must be successfully called prior to this function, and the handle returned in *riHandle* should be used.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.
Call pending:	**QABatchWV_Clean** has been called too many times with the specified handle. Ensure that **QABatchWV_EndSearch** is called at the end of each search once the results have been returned.
Invalid input item:	There is a mistake in the input specification passed to **QABatchWV_Open** (see `InputLineCount` and `InputLineN`).

Comments

When you call **QABatchWV_Clean** with an input address, the search string should be comma-separated so that QAS Batch API can distinguish between address elements. For example:

> 15 Stoke Way, Edgbaston, Birmingham
> 25 Tattersalls Lane, Melbourne 3000

This function call passes back four values:

- Search handle. This is used to retrieve results pertaining to this search, with the functions **QABatchWV_GetFormattedLine**, and **QABatchWV_ GetUnusedInput**. This parameter should be set to NULL if you do not want to use search handles.
- The Match code. This indicates how successfully QAS Batch API has matched your input address.
- Postal code. If QAS Batch API has found a valid postal code in your input address, or has added a postal code to the address, it is returned in this field.

- Dataset identifier. This is the three-character alpha-numeric code that identifies a particular dataset. The code will be that of your default dataset, unless QAS Batch API has detected a different country name in the input address. QAS Batch API will return the dataset identifier for that country name even if the dataset is unavailable. QAS Batch API will only return a country name for unmatched addresses if you set a default dataset.

A maximum of eight searches can be submitted at a time to each handle of the API. If any more than eight are submitted simultaneously, an error will be returned.

Related Functions:

QABatchWV_GetFormattedLine

QABatchWV_GetUnusedInput

QABatchWV_EndSearch

QABatchWV_Close

Closes down this instance of the API.

Pre-call conditions

An instance of the API has been initialised and been opened with **QABatchWV_ Open**, and no searches are in progress.

Prototype

```
INTRET QABatchWV_Close
  (INTVAL viHandle);
```

Parameters

Argument	Explanation
viHandle	Handle for this instance of the API.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.
Bad handle:	The handle passed to the parameter *viHandle* is not valid. **QABatchWV_Open** must be successfully called prior to this function, and the handle returned in *riHandle* should be used.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.

Comments

The error qaerr_CALLPENDING will be returned if this instance of the API has a search in progress.

Related Functions:

QABatchWV_Open

QABatchWV_CompareAuditCode

This information is only relevant when using GBR or AUS data with additional Suppression data.

Compares two previously returned audit codes generated using **QABatchWV_ GetAuditCode**. An XML report is returned into the specified buffer.

Pre-call Conditions

The API must be initialised and as specific instance should have been started with **QABatchWV_Open**.

Prototype

```
INTRET QABatchWV_CompareAuditCode
  (INTVAL viHandle,
  STRVAL vsAudit1,
  STRVAL vsAudit2,
  STRREF rsXmlReport,
  INTVAL viXmlReportLength);
```

Parameters

Argument	Explanation
viHandle	Handle for this instance of the API.
vsAudit1	An audit code created by **QABatchWV_GetAuditCode**
vsAudit2	An audit code created by **QABatchWV_GetAuditCode**
rsXmlReport	Returns the XML report into the specified buffer.
viXmlReportLength	Maximum length of buffer to receive returned XML report.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_ Startup** must be successfully called prior to this function.
Bad handle:	The handle passed to the parameter *viHandle* is not valid. **QABatchWV_Open** must be successfully called prior to this function, and the handle returned in *riHandle* should be used.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.
More Clicks Needed:	If less than 10 clicks have been deducted since the first audit code was generated, the error `qaerr_ MORECLICKSNEEDED` will be returned. See "Appendix A: Error Code Listing" on page 179 for more information.

Comments

The two audit codes being compared must have both been created using the QAS Batch installation that this function is being called from.

Related Functions:

QABatchWV_GetAuditCode

QABatchWV_CounterClose

This information is only relevant when using GBR or AUS data with additional Suppression data.

Closes an active counter handle, removing any stored statistics.

Pre-call Conditions

The API must be initialised and a specific instance should have been started with **QABatchWV_Open**. A counter must have been initialised using **QABatchWV_ CounterOpen**.

Prototype

```
INTRET QABatchWV_CounterClose
  (INTVAL viCounterHandle);
```

Parameters

Argument	Explanation
viCounterHandle	Handle for this instance of the counter.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_ Startup** must be successfully called prior to this function.
Bad handle:	The handle passed to the parameter *viCounterHandle* is not valid. **QABatchWV_CounterOpen** must be successfully called prior to this function, and the handle returned in *riCounterHandle* should be used.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.

Comments

Any counter report that is required must be retrieved before the counter is closed as all recorded statistics are deleted upon closure.

Related Functions:

QABatchWV_CounterOpen

QABatchWV_CounterReport

QABatchWV_CounterReportLength

QABatchWV_CounterOpen

This information is only relevant when using GBR or AUS data with additional Suppression data.

Opens a new counter instance tied to the QAS Batch API instance supplied in *viHandle*.

Pre-call Conditions

The API must be initialised and a specific instance should have been started with **QABatchWV_Open**.

Prototype

```
INTRET QABatchWV_CounterOpen
  (INTVAL viHandle,
  INTREF riCounterHandle);
```

Parameters

Argument	Explanation
viHandle	Handle for this instance of the API.
riCounterHandle	Handle for this instance of the counter.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_ Startup** must be successfully called prior to this function.
Bad handle:	The handle passed to the parameter *viHandle* is not valid. **QABatchWV_Open** must be successfully called prior to this function, and the handle returned in *riHandle* should be used.

Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.
Invalid input item:	There is a mistake in the input specification passed to **QABatchWV_Open** (see `InputLineCount` and `InputLineN`).

Comments

If viHandle is set to -1, the counter will count transactions from all open instances of the QAS Batch API.

A counter tied to an instance of the QAS Batch API will be closed automatically if the instance is closed.

Related Functions:

QABatchWV_CounterClose

QABatchWV_CounterReport

QABatchWV_CounterReportLength

QABatchWV_CounterReport

This information is only relevant when using GBR or AUS data with additional Suppression data.

Returns an XML report of the records cleaned and clicks used since the counter was opened (see **QABatchWV_CounterOpen**)

Pre-call Conditions

The API must be initialised and a specific instance should have been started with **QABatchWV_Open**. A counter should also have been opened using **QABatchWV_CounterOpen**.

Prototype

```
INTRET QABatchWV_CounterReport
  (INTVAL viCounterHandle,
  STRREF rsXmlReport,
  INTVAL viXmlReportLength);
```

Parameters

Argument	Explanation
viCounterHandle	Handle for this instance of the counter.
rsXmlReport	Returns the XML report into a specified buffer.
viXmlReportLength	Maximum length of buffer to receive returned XML report.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_ Startup** must be successfully called prior to this function.

Bad handle:	The handle passed to the parameter *viHandle* is not valid. **QABatchWV_Open** must be successfully called prior to this function, and the handle returned in *riHandle* should be used.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.

Comments

Though click information will not be returned if less than ten clicks have been used since the counter was created (or since the last counter report was run), match information will always be returned.

The XML report will be returned into the specified buffer.

Related Functions:

QABatchWV_CounterOpen

QABatchWV_CounterReportLength

QABatchWV_CounterClose

QABatchWV_CounterReportLength

This information is only relevant when using GBR or AUS data with additional Suppression data.

Returns the length of the XML counter report that would be returned. This length is given in bytes.

Pre-call Conditions

The API must be initialised and a specific instance should have been started with **QABatchWV_Open**. A counter should also have been opened using **QABatchWV_CounterOpen**.

Prototype

```
INTRET QABatchWV_CounterReportLength
  (INTVAL viCounterHandle,
  INTREF riXmlReportLength);
```

Parameters

Argument	Explanation
viCounterHandle	Handle for this instance of the counter.
riXmlReportLength	Returns an integer of the report length in bytes.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_ Startup** must be successfully called prior to this function.

Bad handle:	The handle passed to the parameter *viHandle* is not valid. **QABatchWV_Open** must be successfully called prior to this function, and the handle returned in *riHandle* should be used.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.

Related Functions:

QABatchWV_CounterOpen

QABatchWV_CounterReport

QABatchWV_CounterClose

QABatchWV_CountryCount

Retrieves the number of dataset identifiers available to this instance of the API.

Pre-call conditions

The API is initialised, and a specific instance has been started with **QABatchWV_ Open**.

Prototype

```
INTRET QABatchWV_CountryCount
  (INTVAL viHandle,
  INTREF riCount);
```

Parameters

Argument	Explanation
viHandle	Handle for this instance of the API. If the handle that is passed to *viHandle* is 0, all datasets are used. If the handle is passed, all datasets within the section are used.
riCount	Number of datasets available.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.
Bad handle:	The handle passed to the parameter *viHandle* is not valid. **QABatchWV_Open** must be successfully called prior to this function, and the handle returned in *riHandle* should be used.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.

Comments

This function tells you how many datasets are available for this particular thread of the API. The availability of datasets is determined by the configuration file specified in **QABatchWV_Open**.

Once you have the number of datasets, you can call **QABatchWV_GetCountry** as many times as is necessary to retrieve a description of each dataset.

Related Functions:

QABatchWV_GetCountry

QABatchWV_DataSetCount

Retrieves the number of DataPlus sets, additional datasets and keyfinder sets that are associated with a dataset.

Pre-call conditions

The API is initialised. No specific instances of the API need to be running.

Prototype

```
INTRET QABatchWV_DataSetCount
  (INTVAL viHandle,
  STRVAL vsIsoCode,
  INTREF riCount);
```

Parameters

Argument	Explanation
viHandle	Handle for this instance of the API.
vsIsoCode	The identifier of the set for which information will be returned.
riCount	Number of sets associated with a dataset.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.
File error:	The file specified in *vsIniFile* could not be opened.

Related Functions:

QABatchWV_GetDataSet

QABatchWV_DataSetInfo

QABatchWV_GetCountry

QABatchWV_CountryCount

QABatchWV_DataSetInfo

Returns information about a particular dataset.

Pre-call conditions

The API must be initialised. A specific instance must have been started with **QABatchWV_Open**.

Prototype

```
INTRET QABatchWV_DataSetInfo
  (INTVAL viHandle,
  STRVAL vsIsoCode,
  INTREF riDaysLeft,
  INTREF riDataDaysLeft,
  INTREF riLicenceDaysLeft,
  STRREF rsVersion,
  INTVAL viVerLength,
  STRREF rsCopyright,
  INTVAL viCopyrightLength);
```

Parameters

Argument	Explanation
viHandle	Handle for this instance of the API.
vsIsoCode	The dataset or additional dataset identifier for which information will be returned.
riDaysLeft	The lower of *riDataDaysLeft* and *riLicenceDaysLeft.*
riDataDaysLeft	Number of days left until the first dataset expires.
riLicenceDaysLeft	Number of days left until the first licence in the dataset expires.
rsVersion	Buffer to receive version information for the dataset.
viVerLength	Maximum length of *rsVersion.*
rsCopyright	Buffer to receive copyright information.
viCopyrightLength	Maximum length of *rsCopyright.*

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_ Startup** must be successfully called prior to this function.
Bad handle:	The handle passed to the parameter *viHandle* is not valid. **QABatchWV_Open** must be successfully called prior to this function, and the handle returned in *riHandle* should be used.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.
Invalid country:	The string passed to the *vsIsoCode* parameter is not valid. The dataset identifier of the dataset has to be specified in the `InstalledData` configuration setting.

Comments

When you pass a dataset identifier (for example, 'AUS' for Australia) into this function, it returns information about the dataset to which the dataset identifier relates.

The parameter *riDaysLeft* contains the lower of *riDataDaysLeft* and *riLicenceDaysLeft*. For example, if *riDataDaysLeft* is 60 and *riLicenceDaysLeft* is 90, the *riDaysLeft* parameter will contain 60.

The parameter *rsCopyright* contains copyright information for the dataset in question. For example, the Australia dataset returns a copyright for Australia Post.

If you only want to see some of the information that can be returned from this function, you can set return parameters to NULL. For example, if you only want to see when the dataset is going to expire, you can set *rsVersion* and *rsCopyright* to NULL.

Related Functions:

QABatchWV_CountryCount, **QABatchWV_GetCountry**

QABatchWV_DPVGetCode

This information is only relevant when using USA data.

Used to query the lock code generated by the product when DPV is disabled in the event of a seed address being searched upon. It is this code that must be reported back to Experian Data Quality, in order for Experian Data Quality to generate a corresponding unlock key. Since the lock code varies in length, **QABatchWV_DPVGetCodeLength** should be called in order to make sure the buffer provided is large enough for the lock code.

Prototype

```
INTRET QABatchWV_DPVGetCode
  (STRREF rsLockCode,
  INTVAL viLockCodeLength);
```

Parameters

Argument	Explanation
rsLockCode	Buffer to receive lock code.
viLockCodeLength	Length of provided buffer *rsLockCode*.

Return values

Either: 0 if successful (lock code has been stored in the supplied buffer).

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_ Startup** must be successfully called prior to this function.

Comments

When DPV is disabled by a seed address, QAS Batch API generates a variable-length alphanumeric code which is required to unlock DPV. If you do not call **QABatchWV_DPVGetCodeLength** to determine the length of the lock code then the supplied buffer may not be large enough. If truncation occurs while populating the buffer *rsLockCode*, this will be signalled in the error log. If an error occurs and it is possible to populate the buffer *rsLockCode*, then this will be zero terminated.

QABatchWV_DPVGetCodeLength

This information is only relevant when using USA data.

Returns the length of the lock code generated by the product where DPV is disabled in the event of a seed address being searched upon. This function should be called before **QABatchWV_DPVGetCode** in order to ensure an adequate buffer is supplied to that function to obtain the lock code.

Prototype

```
INTRET QABatchWV_DPVGetCodeLength
  (INTREF riLockCodeLength);
```

Parameters

Argument	Explanation
riLockCodeLength	Length of the lock code.

Return values

Either: 0 if successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.

Comments

QAS Batch API generates an alphanumeric lock code when DPV is disabled. The code can vary in length, so this function should be called before **QABatchWV_DPVGetCode** in order to ensure an adequate buffer is supplied to that function when obtaining the lock code.

QABatchWV_DPVGetInfo

🇺🇸 This information is only relevant when using USA data.

Returns information about the DPV seed address which caused DPV to be disabled. The USPS require this information to be submitted before an unlock key can be issued.

Prototype

```
INTRET QABatchWV_DPVGetInfo
  (INTVAL viDPVInfoType,
  STRREF rsDPVInfo,
  INTVAL viLength);
```

Parameters

Argument	Explanation
viDPVInfoType	Type of lock information to be returned.
rsDPVInfo	Buffer to receive lock information.
viLength	Length of provided buffer *rsDPVInfo*.

Return values

Either: 0 if successful (lock information has been stored in the supplied buffer).

Or: negative error code

Possible values of *viDPVInfoType* are:

Value	Description
dpvlockinfo_DATE	Returns the date the seed address was encountered.
dpvlockinfo_TIME	Returns the time the seed address was encountered.
dpvlockinfo_SEED	Returns the seed address that was searched upon.
dpvlockinfo_SESSION	Returns the name of the session in use when the seed address was encountered.

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_ Startup** must be successfully called prior to this function.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.

Comments

If the DPV has not been disabled by a seed address, or if the status of the DPV system cannot be determined, this function will return blank strings.

If truncation occurs while populating the buffer *rsDPVInfo*, this will be signalled in the error log. If an error occurs and it is possible to populate the buffer *rsDPVInfo*, then this will be zero terminated.

QABatchWV_DPVSetKey

This information is only relevant when using USA data.

Sets an unlock key to re-enable DPV functionality where the DPV functionality is disabled (i.e. key supplied by Experian Data Quality following the reporting of the corresponding lock code).

Prototype

```
INTRET QABatchWV_DPVSetKey
   (STRVAL vsUnlockKey);
```

Parameters

Argument	Explanation
vsUnlockKey	Buffer containing the unlock key.

Return values

Either: 0 if successful (the unlock code has re-enabled the DPV system)

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.
Invalid key:	The key specified in parameter *vsUnlockKey* is not the valid key required to unlock the DPV system. The DPV unlock key should be as provided by Experian Data Quality.

QABatchWV_DPVState

This information is only relevant when using USA data.

Determines whether DPV functionality is enabled, disabled, or not in use. This information can also be determined on a per-search basis through the use of the additional dataset-specific information component of QAS Batch API's return code.

Prototype

```
INTRET QABatchWV_DPVState
  (INTREF riDPVState);
```

Parameters

Argument	Explanation
riDPVState	Returned state of the DPV system.

Return values

Either: 0 if successful (DPV state has been determined)

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.

Comments

The argument *riDPVState* will be populated with one of the following values:

State	Macro	Explanation
-1	DPVstate_Unknown	Returned by the function in the event of an error.
0	DPVstate_NotInUse	The DPV system is not in use.

State	Macro	Explanation
1	DPVstate_Enabled	The DPV system is enabled.
2	DPVstate_Disabled	The DPV system has been disabled following a seed address search.

QABatchWV_EndSearch

Deallocates resources and the search handle used by a call to **QABatchWV_ Clean**.

Pre-call conditions

An instance of the API has been initialised and opened with **QABatchWV_Open**. **QABatchWV_Clean** has been called, and all results have been retrieved.

Prototype

```
INTRET QABatchWV_EndSearch
  (INTVAL viSearchHandle);
```

Parameters

Argument	Explanation
viSearchHandle	Handle for this search. If a NULL search handle was passed to **QABatchWV_Clean** this should be set to 0.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.
Bad handle:	The handle passed to the parameter *viSearchHandle* is not valid. **QABatchWV_Clean** must be successfully called prior to this function, and the handle returned from *riSearchHandle* should be used.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.

Comments

This function should be called after results have been retrieved from **QABatchWV_Clean**, in order to free the search handle for use in further searches.

Related Functions:

QABatchWV_Clean

QABatchWV_FormattedLineCount

Returns the number of formatted lines that a search has resulted in.

Pre-call conditions

The API must be initialised, and started with **QABatchWV_Open**, and a call to **QABatchWV_Clean** should have been completed successfully.

Prototype

```
INTRET QABatchWV_FormattedLineCount
  (INTVAL viSearchHandle,
  INTREF riCount);
```

Parameters

Argument	Explanation
viSearchHandle	Handle for this search. If a NULL search handle was passed to **QABatchWV_Clean** this should be set to 0.
riCount	Count of formatted lines.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_ Startup** must be successfully called prior to this function.
Bad handle:	The handle passed to the parameter *viSearchHandle* is not valid. **QABatchWV_Clean** must be successfully called prior to this function, and the handle returned from *riSearchHandle* should be used.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.

Comments

This function tells you how many times **QABatchWV_GetFormattedLine** needs to be called in order to retrieve a full address.

For the standard address and enhanced cleaning modes, the value of this function is constant for a dataset within a session as its value is set in the configuration file.

For the Postal code update mode, the value is determined by the number of clean lines passed into **QABatchWV_Clean**.

Related Functions:

QABatchWV_GetFormattedLine

QABatchWV_LayoutLineElements

QABatchWV_GetAuditCode

This information is only relevant when using GBR or AUS data with additional Suppression data.

Extracts a text-based audit code from the counters file on the disk where QAS Batch API With Suppression is installed.

Pre-call Conditions

The API is initialised. No specific instances of the API need to be running.

Prototype

```
INTRET QABatchWV_GetAuditCode
  (STRREF rsAuditCode,
  INTVAL viAuditCodeLength);
```

Parameters

Argument	Explanation
rsAuditCode	Buffer to receive audit code string.
viAuditCodeLength	Maximum length of audit code.

Return Values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.

Related Functions:

QABatchWV_ApplyUpdateCode

QABatchWV_GetCountry

Returns a description of a dataset available to this instance of the API.

Pre-call conditions

The API must be initialised, with no searches in progress. A specific instance must have been started with **QABatchWV_Open**.

Prototype

```
INTRET QABatchWV_GetCountry
  (INTVAL viHandle,
  INTVAL viIndex,
  STRREF rsIsoCode,
  STRREF rsCountry,
  INTVAL viLength);
```

Parameters

Argument	Explanation
viHandle	Handle for this instance of the API. If the handle that is passed to *viHandle* is 0, all datasets are used. If the handle is passed, all datasets within the section are used.
viIndex	Number of dataset.
rsIsoCode	Buffer to receive the dataset identifier.
rsCountry	Buffer to receive name of dataset.
viLength	Maximum length of *rsCountry.*

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_ Startup** must be successfully called prior to this function.

Bad handle:	The handle passed to the parameter *viHandle* is not valid. **QABatchWV_Open** must be successfully called prior to this function, and the handle returned in *riHandle* should be used.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.
Country out of range:	The index passed to the *viIndex* parameter is too large. The value should be between 0 and the count returned from **QABatchWV_CountryCount** -1.

Comments

This function, in conjunction with **QABatchWV_CountryCount**, is useful if you want to confirm the number, names and dataset identifiers of available datasets for a particular instance of the API. You might want to call these two functions prior to the first call of **QABatchWV_Clean** so that you know which datasets are available to search on. You can also pass the dataset identifier into **QABatchWV_DataSetInfo** to get further information about the dataset.

You should call this function as many times as required to retrieve dataset details. For example, if **QABatchWV_CountryCount** returned a count of 4, you would call **QABatchWV_GetCountry** a maximum of four times to retrieve details of each dataset, setting *viIndex* to 0, 1, 2 and 3.

The parameter *viIndex* contains the number of the dataset whose details you want to retrieve. For example, inputting 0 retrieves the name of the first installed dataset, 1 returns the name of the second dataset, and so on.

The output parameters *rsIsoCode* and *rsCountry* contain the dataset identifier and country name respectively of a dataset. A dataset identifier is a three-character descriptor for a dataset, which appears on the data sheet supplied with each dataset. For example, the Australia dataset identifier is AUS.

The *rsIsoCode* buffer must be at least 4 characters long, in order to accommodate a three-character dataset identifier and a trailing NULL.

Related Functions:

QABatchWV_CountryCount

QABatchWV_DataSetInfo

QABatchWV_GetDataSet

Retrieves the DataPlus sets, additional datasets and keyfinder sets for a dataset.

Pre-call conditions

The API must be initialised with **QABatchWV_Open**, and a call to **QABatchWV_ DataSetCount** should have been completed successfully.

Prototype

```
INTRET QABatchWV_GetDataSet
  (INTVAL viHandle,
  INTVAL viIndex,
  STRVAL vsIsoCode,
  STRREF rsName,
  INTVAL viNameLength,
  STRREF rsDesc,
  INTVAL viDescLength,
  LONGREF rlType);
```

Parameters

Argument	Explanation
viHandle	Handle for this instance of the API.
viIndex	Set index (between 0 and count).
vsIsoCode	The dataset identifier of the dataset for which information will be returned.
rsName	Name of dataset.
viNameLength	Maximum length of buffer for *rsName.*
rsDesc	Description of the dataset.
viDescLength	Maximum length of buffer for *rsDesc*.
rlType	Type of data.

Return values

Either: 0 if call successful

Or: negative error code

Possible values of *rlType* are:

Value		Description
datasettype_BASE	1	Dataset type is base dataset.
datasettype_DATAPLUS	2	Dataset type is DataPlus set.
datasettype_ADDITIONAL	4	Dataset type is additional dataset.
datasettype_KEYFINDER	32	Dataset type is keyfinder (i.e. the dataset contains a logical reverse search key).

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.
Function called out of sequence:	The function has been called out of sequence. **QABatchWV_DataSetCount** must be successfully called prior to this function.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.
File error:	The file specified in *vsIniFile* could not be opened.
Dataset out of range:	The index passed to the *viIndex* parameter is too large. It should be between 0 and the count from **QABatchWV_DataSetCount** -1.

Related Functions:

QABatchWV_DataSetCount

QABatchWV_DataSetInfo

QABatchWV_GetCountry

QABatchWV_CountryCount

QABatchWV_GetDPFieldCount

Retrieves the number of DataPlus fields that are associated with a given DataPlus or Additional dataset. A list of suitable datasets can be obtained from **QABatchWV_DataSetCount** and **QABatchWV_GetDataSet**.

Datasets that can contain DataPlus fields are those of the types **datasettype_ DATAPLUS** or **datasettype_ADDITIONAL**. See **QABatchWV_GetDataSet** for more information about dataset types.

Note that not all datasets listed by **QABatchWV_DataSetCount** and **QABatchWV_GetDataSet** will be accessible due to layout and licensing restrictions.

Pre-call conditions

The API must be initialized with **QABatchWV_Open**.

Prototype

```
INTRET QABatchWV_GetDPFiledCount
  (INTVAL viHandle,
  STRVAL vsDPSet,
  LONGREF rlCount);
```

Parameters

Argument	Explanation
viHandle	Handle for this instance of the API.
vsDPSet	The identifier of the dataset for which information will be returned.
rlCount	Number of DataPlus fields associated with a dataset.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.
No DataPlus:	The requested DataPlus is not available for the current layout.

Related Functions:

QABatchWV_DataSetCount

QABatchWV_GetDataSet

QABatchWV_GetDPFieldName

QABatchWV_GetDPFieldInfo

QABatchWV_GetDPFieldInfo

Retrieves information for a specific DataPlus field for a dataset. DataPlus fields are indexed from zero to **QABatchWV_GetDPFieldCount** -1.

Pre-call conditions

The API must be initialized with **QABatchWV_Open** and a call to **QABatchWV_GetDPFieldCount** should have been completed successfully.

Prototype

```
INTRET QABatchWV_GetDPFieldInfo
  (INTVAL viHandle,
  STRVAL vsDPSet,
  INTVAL viIndex,
  INTVAL viInfoType,
  STRREF rsStringInfo,
  INTVAL viStringInfoLength,
  LONGREF rlLongInfo);
```

Parameters

Argument	Explanation
viHandle	Handle for this instance of the API.
vsDPSet	The dataset identifier of the dataset for which information will be returned.
viIndex	Field index (between 0 and count).
viInfoType	Field info type.
rsStringInfo	Receives the string information for string info types.
viStringInfoLength	Maximum length of buffer for *rsStringInfo*.
rlLongInfo	Receives the long information for long info types.

Return values

Either: 0 if call successful

Or: negative error code

Possible values of *vsInfoType* are:

Value	Type	Description
dpfieldinfotype_CODE	String	Field Code, also returned by **QABatchWV_GetDPFieldName**.
dpfieldinfotype_NAME	String	Field Name, also returned by **QABatchWV_GetDPFieldName**.
dpfieldinfotype_ FORMATSTRING	String	Format String, mainly for date fields.
dpfieldinfotype_ATTR	Long	Attributes, bitfield, see the *dpfieldatr_* values below.
dpfieldinfotype_MAXLEN	Long	Maximum possible length for the value when formatting with this field.

Possible attribute flags returned in *rlLongInfo* for *dpfieldinfotype_ ATTR* are:

Value	Description
dpfieldattr_DATE	Field is a date.
dpfieldattr_BARCODE	Field is a barcode.
dpfieldattr_SUPPRESSION	Field is part of suppression data.
dpfieldattr_DATERANGE	Field is a date range.

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.
Function called out of sequence:	The function has been called out of sequence. **QABatchWV_ GetDPFieldCount** must be successfully called prior to this function.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.
No DataPlus:	The requested DataPlus is not available for the current layout.
Field out of range:	The index passed to the *viIndex* parameter is too large. It should be between 0 and the count from **QABatchWV_ GetDPFieldCount** -1.

Invalid info type:	The info type passed to the *viInfoType* parameter is not valid. See the table above for the list of valid info types.

Related Functions:

QABatchWV_GetDPFieldCount

QABatchWV_GetDPFieldName

QABatchWV_GetDPFieldName

Retrieves the code and name for a specific DataPlus field for a dataset. DataPlus fields are indexed from zero to **QABatchWV_GetDPFieldCount** -1.

The DataPlus field code is suitable for addresses format specification. The DataPlus field name is a human-readable description of the field. Use **QABatchWV_GetDPFieldInfo** for more information about the DataPlus field.

Pre-call conditions

The API must be initialized with **QABatchWV_Open** and a call to **QABatchWV_ GetDPFieldCount** should have been completed successfully.

Prototype

```
INTRET QABatchWV_GetDPFieldName
   (INTVAL viHandle,
  STRVAL vsDPSet,
  INTVAL viIndex,
  STRREF rsCode,
  INTVAL viCodeLength,
  STRREF rsName,
  INTVAL viNameLength);
```

Parameters

Argument	Explanation
viHandle	Handle for this instance of the API.
vsDPSet	The dataset identifier of the dataset for which information will be returned.
viIndex	Field index (between 0 and count).
rsCode	DataPlus field code.
viCodeLength	Maximum length of buffer for *rsCode*.
rsName	DataPlus field name.
viNameLength	Maximum length of buffer for *rsName*.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.
Function called out of sequence:	The function has been called out of sequence. **QABatchWV_ GetDPFieldCount** must be successfully called prior to this function.
Bad parameter:	One of the API parameters has been passed as an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.
No DataPlus:	The requested DataPlus is not available for the current layout.
Field out of range:	The index passed to the *viIndex* parameter is too large. It should be between 0 and the count from **QABatchWV_ GetDPFieldCount** -1.

Related Functions:

QABatchWV_GetDPFieldCount

QABatchWV_GetDPFieldInfo

QABatchWV_GetFormattedLine

This function gets one formatted address line from the latest retrieved address.

Pre-call conditions

The API must be initialised and started with **QABatchWV_Open**, and a call to **QABatchWV_Clean** should have been completed successfully.

Prototype

```
INTRET QABatchWV_GetFormattedLine
  (INTVAL viSearchHandle,
  INTVAL viLine,
  STRREF rsBuffer,
  INTVAL viBuffLen);
```

Parameters

Argument	Explanation
viSearchHandle	Handle for this search. If a NULL search handle was passed to **QABatchWV_Clean** this should be set to 0.
viLine	Number of the line to be retrieved.
rsBuffer	Buffer to return the formatted address line.
viBuffLen	Maximum length of the address line buffer.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.

Bad handle:	The handle passed to the parameter *viSearchHandle* is not valid. **QABatchWV_Clean** must be successfully called prior to this function, and the handle returned from *riSearchHandle* should be used.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.
Out of range:	The parameter passed to *viLine* is too large. This should be a value between 0 and the count from **QABatchWV_ FormattedLineCount** -1.

Comments

This function should be called as many times as necessary to retrieve a full address. If this function is called during a Postal code only clean, it will return the same number of lines as contained in the input address.

Related Functions:

QABatchWV_FormattedLineCount

QABatchWV_GetLayout

Retrieves the name of one layout in the specified configuration file. For more information about configuration files see page 147.

Pre-call conditions

The API is initialised. No specific instances of the API need to be running.

Prototype

```
INTRET QABatchWV_GetLayout
  (STRVAL vsIniFile,
  INTVAL viIndex,
  STRREF rsName,
  INTVAL viLength);
```

Parameters

Argument	Explanation
vsIniFile	Name of a configuration file, if you have created a separate file for layouts. If *vsIniFile* is not specified, the default configuration file, qaworld.ini, will be used.
viIndex	Number of layout.
rsName	Name of layout.
viLength	Maximum length of buffer for *rsName.*

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_ Startup** must be successfully called prior to this function.

Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.
File error:	The file specified in *vsIniFile* could not be opened.
Layout out of range:	The index passed to the *viIndex* parameter is too large. It should be between 0 and the count from **QABatchWV_LayoutCount** -1.

Comments

This function, in conjunction with **QABatchWV_LayoutCount**, is useful if you want to confirm the number and names of available configuration layouts prior to calling **QABatchWV_Open**.

You should call this function as many times as required to retrieve layout names from a configuration file. For example, if **QABatchWV_LayoutCount** returned a count of 6, you would call **QABatchWV_GetLayout** a maximum of six times to retrieve each layout name.

The parameter *viIndex* contains the number of the layout whose name you want to retrieve. For example, inputting 0 retrieves the name of the first layout in the configuration file, 1 returns the name of the second layout, and so on.

Related Functions:

QABatchWV_LayoutCount

QABatchWV_ChangeLayout

QABatchWV_GetLicenceInfo

Returns a specified line of licensing information.

Pre-call conditions

The API must be initialised. A specific instance must have been started with **QABatchWV_Open**.

Prototype

```
INTRET QABatchWV_GetLicenceInfo
  (INTVAL viHandle,
  STRVAL vsIsoCode,
  INTVAL viLine,
  STRREF rsLicenceInfo,
  INTVAL viLicenceInfoLength);
```

Parameters

Argument	Explanation
viHandle	Handle for this instance of the API.
vsIsoCode	The dataset identifier of the dataset for which information will be returned.
viLine	The line number of the licensing information to get.
rsLicenceInfo	The licensing information string returned.
viLicenceInfoLength	The size of the buffer passed into *rsLicenceInfo*.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.

Bad handle:	The handle passed to the parameter *viHandle* is not valid. **QABatchWV_Open** must be successfully called prior to this function, and the handle returned in *riHandle* should be used.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.
Invalid country:	The string passed to the *vsIsoCode* parameter is not valid. The dataset identifier has to be specified in the `InstalledData` configuration setting.
Out of range:	The parameter passed to *viLine* is too large. This should be a value between 0 and the count from **QABatchWV_ LicenceInfoCount** -1.

Comments

The licensing information string returned will contain the licence information for one data file in the dataset.

QABatchWV_GetMatchInfo

This function provides access to detailed match information. Most values are returned as integers to ease processing. Each parameter provides a discrete component of the full match code.

Pre-call conditions

The API must be initialised and started with **QABatchWV_Open**, and a call to **QABatchWV_Clean** should have been completed successfully.

Prototype

```
INTRET QABatchWV_GetMatchInfo
  (INTVAL viSearchHandle,
  STRREF rsIsoCode,
  STRREF rsMatchType,
  INTREF riConfidence,
  INTREF riPostcodeAction,
  INTREF riAddressAction,
  LONGREF rlGenericInfo,
  LONGREF rlCountryInfo,
  LONGREF rlCountryInfo2)
```

Parameters

Argument	Explanation
viSearchHandle	Handle for this search thread.
rsIsoCode	Dataset identifier.
rsMatchType	Match type (zero-terminated single letter).
riConfidence	Confidence of match (0-9).
riPostcodeAction	Action performed on postal code (0-3).
riAddressAction	Action performed on address (0-3).
rlGenericInfo	Generic information (32 bit values).
rlCountryInfo	Dataset-specific information (32 bit values).
rlCountryInfo2	Additional Dataset-specific information (32 bit values).

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.
Bad handle:	The handle passed to the parameter *viSearchHandle* is not valid. **QABatchWV_Clean** must be successfully called prior to this function, and the handle returned from *riSearchHandle* should be used.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.

Related Functions:

QABatchWV_Clean

QABatchWV_Open

QABatchWV_GetUnusedInput

This function gets one unused line from the latest input address.

Pre-call conditions

The API must be initialised with **QABatchWV_Open**, and a call to **QABatchWV_Clean** should have been completed successfully.

Prototype

```
INTRET QABatchWV_GetUnusedInput
  (INTVAL viSearchHandle,
  INTVAL viLine,
  STRREF rsBuffer,
  INTVAL viLength,
  LONGREF rlLineCompleteness,
  LONGREF rlLineType,
  LONGREF rlLinePosition,
  INTREF riCareOf,
  INTREF riPremSuffix);
```

Parameters

Argument	Explanation
viSearchHandle	Handle for this search thread. If a NULL search handle was passed to **QABatchWV_Clean** this should be set to 0.
viLine	Number of the line to be retrieved.
rsBuffer	Buffer to return the unused line.
viLength	Maximum length of the unused line buffer.
rlLineCompleteness	Describes how much of the line is unused.
rlLineType	Describes what type of information is on the line.
rlLinePosition	Describes the position of the line, relative to the street.
riCareOf	Boolean denoting whether the line is a 'care of' premises prefix.
riPremSuffix	Boolean denoting whether the line is an alphabetic premises suffix.

Return values

Either: 0 if call successful

Or: negative error code

Possible values of *rlLineCompleteness* are:

Value	Description
unusedcompleteness_COMPLETE	Complete line (i.e. as supplied)
unusedcompleteness_PARTIAL	Incomplete line (i.e. part matched)

Possible values of *rlLineType* are:

Value	Description
unusedtype_ADDRESS	Unused address data
unusedtype_NAME	Unused name data

Possible values of *rlLinePosition* are:

Value	Description
unusedstreet_PRESTREET	Appeared before matched street
unusedstreet_POSTSTREET	Appeared after matched street

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.
Bad handle:	The handle passed to the parameter *viSearchHandle* is not valid. **QABatchWV_Clean** must be successfully called prior to this function, and the handle returned from *riSearchHandle* should be used.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.
Out of range:	The parameter passed to *viLine* is too large. This should be a value between 0 and the count from **QABatchWV_ UnusedLineCount** -1.

Comments

This function is optional; call it in conjunction with **QABatchWV_ UnusedLineCount** if you want to see which parts of the input address, if any, were not included in the output address.

Unused lines will include any leading non-matching elements from the input address. For example, say this is the input address:

> John Smith, Suite 1, Level 9, 60 Miller St, North Sydney, 2060

The dataset (Australia, in this case) does not contain names, so although the rest of the address is correct, QAS Batch API cannot match the leading element (i.e. 'John Smith') to anything and returns it as unused.

If the `CleaningAction` keyword is set to 'Enhanced', then 'John Smith' would be prefixed to the output address. If this is the case, it would not be returned as an unused line.

The function should be called as many times as necessary to retrieve any unused lines from the input address. If the function is called out of place (for example, before **QABatchWV_Clean**), it will fail with an error.

Related Functions:

QABatchWV_UnusedLineCount

QABatchWV_LayoutCount

Retrieves the number of available layouts in the specified configuration file. For more information about configuration files see page 147.

Pre-call conditions

The API is initialised. No specific instances of the API need to be running.

Prototype

```
INTRET QABatchWV_LayoutCount
  (STRVAL vsIniFile,
  INTREF riCount);
```

Parameters

Argument	Explanation
vsIniFile	Name of a configuration file, if you have created a separate file for layouts. If *vsIniFile* is not specified, the default configuration file, qaworld.ini, will be used.
riCount	Number of layouts in the configuration file.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.
File error:	The file specified in *vsIniFile* could not be opened.

Comments

This function tells you how many configuration layouts are available in the INI file that you specify. See Configuring the QAS Batch API for a detailed description of configuration files and layouts.

You can call this function before **QABatchWV_Open**, as it does not relate to a specific instance of the API.

Once you have the number of layouts, you can call **QABatchWV_GetLayout** as many times as is necessary to retrieve the name of each layout.

Related Functions:

QABatchWV_GetLayout

QABatchWV_ChangeLayout

QABatchWV_LayoutLineCount

This function returns the number of address lines available in the configuration layout that you specified in your call to **QABatchWV_Open**.

Pre-call conditions

The API must be initialised, and a specific instance should have been started with **QABatchWV_Open**.

Prototype

```
INTRET QABatchWV_LayoutLineCount
  (INTVAL viHandle,
  STRVAL vsCountry,
  INTREF riCount);
```

Parameters

Argument	Explanation
viHandle	Handle for this instance of the API.
vsCountry	The dataset identifier for the layout to be returned.
riCount	Number of lines in layout.

Return values

Either: 0 if call successful

Or: negative error code

Comments

This function tells you how many times the function **QABatchWV_ LayoutLineElements** needs to be called in order to retrieve a full address layout.

The value of this function is constant for a dataset within a session, as its value is set in the configuration file.

Related Functions:

QABatchWV_GetFormattedLine

QABatchWV_LayoutLineElements

QABatchWV_LayoutLineElements

Returns a description of the elements fixed to a particular line of the address layout.

Pre-call conditions

The API must be initialised, and a specific instance should have been started with **QABatchWV_Open**.

Prototype

```
INTRET QABatchWV_LayoutLineElements
  (INTVAL viHandle,
  STRVAL vsCountry,
  INTVAL viLine,
  STRREF rsBuffer,
  INTVAL viLength,
  LONGREF rlFlags);
```

Parameters

Argument	Explanation
viHandle	Handle for this instance of the API.
vsCountry	The dataset identifier for the layout to be returned.
viLine	Address line to retrieve (from 0 to **QABatchWV_LayoutLineCount** -1).
rsBuffer	Buffer to receive line elements.
viLength	Maximum length of *rsBuffer.*
rlFlags	Line description.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_ Startup** must be successfully called prior to this function.
Bad handle:	The handle passed to the parameter *viHandle* is not valid. **QABatchWV_Open** must be successfully called prior to this function, and the handle returned in *riHandle* should be used.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.
Invalid country:	The string passed to the *vsCountry* parameter is not valid. The dataset identifier has to be specified in the `InstalledData` configuration setting.
Out of range:	The parameter passed to *viLine* is too large. This should be a value between 0 and the count from **QABatchWV_ LayoutLineCount** -1.

Comments

This function tells you whether a specific address / DataPlus element, if any, has been fixed to a line of an address layout. When an element is fixed to a line, it cannot appear anywhere else in the address. See the `AddressLineN` setting for details of how to fix an element to a line.

If you want to retrieve descriptions of each address line from the current layout, you should call this function as many times as required. For example, if **QABatchWV_LayoutLineCount** returned a count of 6 address lines, you could call this function a maximum of six times to retrieve a description of each line.

The parameter *viLine* contains the number of the address line whose description you want to retrieve. For example, inputting 0 retrieves the description of the first line in the layout, 1 returns the second line, and so on.

The parameter *rsBuffer* contains the results of the function call. For example, if the town was fixed to line 4 of an address layout, you would set the value of *viLine* as 3, and *rsBuffer* would return 'Town'. If there are no elements (or more than one element) fixed to the line that you have specified, the buffer will be empty.

The *rlFlags* parameter contains the type of line that is being retrieved. The 'flags' that can be returned are as follows:

Flag name	Value
element_ADDRESS	0x00000000
element_NAME	0x00000001
element_DATAPLUS	0x00000002
element_ANCILLARY	0x00000003
format_TRUNCATED	0x00000010
format_OVERFLOW	0x00000020
format_DATAPLUSSYNTAX	0x00000040
format_DATAPLUSEXPIRED	0x00000080
format_DATAPLUSBLANK	0x00000100

A return of 0 (element_ADDRESS) essentially indicates the absence of any flags, which means that the standard line type has been received, i.e. a line containing address elements.

For example, you might have a seven-line address layout, where the first line contains name information, the second to sixth lines contain the address, and the final line is reserved for DataPlus data. In this case, the first line of the returned address would return element_NAME, the last line would return element_ DATAPLUS, and the lines in between would return element_ADDRESS.

The values assigned to each flag are symbolic constants defined by the API, and appear in the prototyped header files for each language.

Related Functions:

QABatchWV_LayoutLineCount

QABatchWV_LicenceInfoCount

Returns the number of lines of licensing information available for a specified dataset.

Pre-call conditions

The API must be initialised. A specific instance must have been started with **QABatchWV_Open**.

Prototype

```
INTRET QABatchWV_LicenceInfoCount
  (INTVAL viHandle,
  STRVAL vsIsoCode,
  INTREF riLicenceInfoCount);
```

Parameters

Argument	Explanation
viHandle	Handle for this instance of the API.
vsIsoCode	The dataset identifier of the dataset for which information will be returned.
vsLicenceInfoCount	The number of licence information lines available.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.
Bad handle:	The handle passed to the parameter *viHandle* is not valid. **QABatchWV_Open** must be successfully called prior to this function, and the handle returned in *riHandle* should be used.

Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.
Invalid country:	The string passed to the *vsIsoCode* parameter is not valid. The dataset identifier has to be specified in the `InstalledData` configuration setting.

Comments

One line of licensing information is available for each data file in the specified dataset. A single data file might be core data, an additional dataset (e.g. Names data), or a DataPlus set.

For example, if you have the GBR Mosaic DataPlus set configured in the qawserve.ini file, but there is no licence information present in the qalicn.ini file, a count of 1 is returned. In addition, **QABatchWV_GetLicenceInfo** returns the string "Data Licence not found".

QABatchWV_Open

Opens an instance of the API, specifying the name of the configuration file to be used, and the layout to use within that file.

Pre-call conditions

The API has been initialised with **QABatchWV_Startup**.

Prototype

```
INTRET QABatchWV_Open
  (STRVAL vsIniFile,
  STRVAL vsLayout,
  LONGVAL vlFlags,
  INTREF riHandle);
```

Parameters

Argument	Explanation
vsIniFile	Name of configuration file to open. If the full path is not specified, QAS Batch API will only check for the configuration file within the program directory. The default configuration file is qaworld.ini, but a separate configuration file can be used to handle layout information. For more information about configuration files, see page 147.
vsLayout	Layout section to open.
vlFlags	Included to provide for extra functionality in future versions.
riHandle	Handle returned by the API (if there is more than one user accessing the search engine).

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_ Startup** must be successfully called prior to this function.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.
No handles:	**QABatchWV_Open** has been called too many times without a call to **QABatchWV_Close**. There is a limit of 32 open handles that can be created. Ensure that **QABatchWV_Close** is called when an open instance is not required.
No installed countries:	No countries were successfully started. Check that all datasets have been installed correctly, and ensure that there are countries defined in the `InstalledData` setting.
INI file error:	The file specified in *vsIniFile* could not be opened. For more information about configuration files see page 147.
No country file:	The API could not find the file country.ini. Ensure that the product has been installed correctly, and the file country.ini is present in the program directory.
File error:	There was an error attempting to open a file. This is most likely to occur when opening a data file. Ensure the datasets have been correctly installed and that the corresponding settings are defined in `InstalledData`.
Invalid layout:	An invalid layout is specified in the configuration file passed to *vsIniFile*. Ensure that a layout is defined within the configuration setting that is passed to the *vsLayout* parameter. Layout names in the configuration file are enclosed by square brackets. However, when specifying a layout name as a parameter in a function call, the square brackets should not be included. Check that the layout has the correct syntax. See `AddressLineCount` and `AddressLineN`.

Comments

When you open an instance of the QAS Batch API, you need to specify the configuration file you are using, and the layout within that configuration file which contains your output address format.

If NULL is passed into either of the above input parameters, the API uses defaults. The default configuration file is qaworld.ini, and the default layout within that file is [QADefault] (without brackets).

You might also get an error if you have chosen (in the configuration file) to create a log file and the QAS Batch API cannot find the specified drive or directory to create it. An error will also be returned if one of the datasets has expired, or has been moved from its default location.

When the API instance has initialised, it returns a handle in the form of an integer. This handle is used to distinguish between multiple users of the QAS Batch API search engine, and should be passed into all subsequent functions. It should be set to 0 if you do not wish to multithread the QAS Batch API.

There can be 32 instances of the API running at any one time, in other words **QABatchWV_Open** can be called 32 times. If all instances are already in use when you call this function, the error qaerr_NOHANDLES is returned.

Related Functions:

QABatchWV_Close

QABatchWV_RunMode

This information is only relevant when using GBR or AUS data with additional Suppression data.

This function switches an instance of QAS Batch into or out of 'Estimate Mode'. Whilst in Estimate Mode, counter statistics can be returned, allowing a user to estimate the cost of a suppression without committing to purchasing the clicks. Upon calling this function, all open counter instances will be reset.

Pre-call Conditions

The API must be initialised and a specific instance should have been started with **QABatchWV_Open**.

Prototype

```
INTRET QABatchWV_RunMode
  (INTVAL viHandle,
  INTVAL viEstimateMode);
```

Parameters

Argument	Explanation
viHandle	Handle for this instance of the API.
viEstimateMode	To activate Estimate Mode, pass a non-zero value into this parameter. To disable Estimate Mode, pass a zero into this parameter.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_ Startup** must be successfully called prior to this function.

Bad handle:	The handle passed to the parameter *viHandle* is not valid. **QABatchWV_Open** must be successfully called prior to this function, and the handle returned in *riHandle* should be used.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.

Comments

Whilst in Estimate Mode, all items will return the matchcode A0000000000000000000 and the formatted line count will be zero. The QAS Batch statistics will however be populated with the aggregate match detail as if the run had actually taken place.

When in Estimate Mode, it is not possible to extract the match and suppression details for an address without switching out of Estimate Mode and running the address again.

After a run is complete, any counter(s) should be read and a report produced before closing the instance or changing the mode, as the information will be cleared.

For more information on using Estimate Mode, see "Estimate Mode" on page 198.

Related Functions:

QABatchWV_CounterOpen

QABatchWV_CounterReport

QABatchWV_CounterClose

QABatchWV_Shutdown

Closes down all instances of the API, and must be called as the final function.

Pre-call conditions

The API is initialised and no searches are in progress.

Prototype

```
INTRET QABatchWV_Shutdown
  (VOIDARG);
```

Return values

Either: 0 if call successful

Or: negative error code

Comments

This function will close down the API completely, and should be called even if **QABatchWV_Close** has shut down all instances of the API. An error will be returned if one or more instances of the API is in use (for example, a search is in progress).

Related Functions:

QABatchWV_Startup

QABatchWV_Close

QABatchWV_Startup

Initialises the API. This function must be called before any other functions can be used.

Pre-call conditions

None.

Prototype

```
INTRET QABatchWV_Startup
  (LONGVAL vlFlags);
```

Parameters

Argument	Explanation
vlFlags	Allows adjusting the way Batch API operates globally, affecting all API instances.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Locale file:	The API could not find the file qalcl.dat. Ensure the product has been installed correctly and the file qalcl.dat is in the program directory.

Comments

This function initialises the QAS Batch API. Once initialised, you can either call informational functions such as **QABatchWV_LayoutCount**, or you can open an instance of the API with **QABatchWV_Open**.

Attribute	Explanation
qabwvflags_NONE	This value uses the default settings for Batch API.
qabwvflags_ SESSION	This value tells QAS Batch API to work in Session Mode. This affects the way Batch API will treat custom.ini files supplied to QABatchWV_Open.

If this function is called out of turn or encounters an error while initialising, the API will shut down and an appropriate error will be flagged.

Related Functions:

QABatchWV_Shutdown

QABatchWV_Open

QABatchWV_UnusedLineCount

This function returns the number of lines that have not been used from the input address.

Pre-call conditions

The API must be initialised, a specific instance started with **QABatchWV_Open**, and **QABatchWV_Clean** called.

Prototype

```
INTRET QABatchWV_UnusedLineCount
  (INTVAL viSearchHandle,
  INTREF riCount);
```

Parameters

Argument	Explanation
viSearchHandle	Handle for this search thread.
riCount	Number of unused lines.

Return values

Either: 0 if call successful

Or: negative error code

Possible Error Scenarios

Not running:	The API has not been started properly. **QABatchWV_Startup** must be successfully called prior to this function.
Bad handle:	The handle passed to the parameter *viSearchHandle* is not valid. **QABatchWV_Clean** must be successfully called prior to this function, and the handle returned from *riSearchHandle* should be used.
Bad parameter:	One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting to determine the cause of the problem.

Comments

The value returned by this function tells you how many times you need to call **QABatchWV_GetUnusedInput**. Its value will depend on the input address, and therefore varies within a session.

A call to this function will produce a zero value if all lines from the input address also appear in the output.

Related Functions:

QABatchWV_GetUnusedInput

QAErrorHistory

This function provides detailed error information for single threaded integrations. **QAErrorHistory** should not be used as a diagnostic tool in a multithreaded integration.

Pre-call conditions

The API has been shut down with **QABatchWV_Shutdown**.

Prototype

```
INTRET QAErrorHistory
  (INTVAL viAll,
  INTVAL viLineNo,
  STRREF rsBuffer,
  INTVAL viBuffLen);
```

Parameters

Argument	Explanation
viAll	Specifies the level of errors returned. Set *viAll* to 0 to access the errors directly associated with the most recent error code, or 1 to access QAS Batch API's entire error history (which is likely to include messages corresponding to unrelated non-fatal internal errors). It is recommended that *viAll* is set to 0.
viLineNo	Line number being accessed (or negative if resetting).
rsBuffer	Buffer to receive error text message.
viBuffLen	Maximum length of the buffer (including room for NULL terminator).

Return values

Either: 0 if call successful

Or: negative error code

Comments

This function is used for accessing more detailed error information than that returned by **QAErrorMessage**. For example, **QAErrorMessage** may return 'file not found' whereas **QAErrorHistory** can highlight the specific file(s) that was not found.

Resetting frees up memory allocated by the function. For more information, see **QASystemInfo** on page 143.

QAErrorLevel

Returns the severity of an error.

Prototype

```
INTRET QAErrorLevel
  (INTVAL viStatus);
```

Parameters

Argument	Explanation
viStatus	Error code.

Return values

Either: 0 for a fatal or serious error

Or: 1 for a warning

Comments

This function indicates the severity of an error returned by the API. A fatal or serious error should be flagged to the user and the API shut down. A warning should be handled in a manner appropriate to the condition and can, if desired, be ignored.

QAErrorMessage

This function converts an error code to a text explanation.

Pre-call conditions

The API must be initialised and a specific instance should have been started with **QABatchWV_Open**.

Prototype

```
VOIDRET QAErrorMessage
  (INTVAL viStatus,
  STRREF rsBuffer,
  INTVAL viBuffLen);
```

Parameters

Argument	Explanation
viStatus	Error code.
rsBuffer	Buffer to receive error text message.
viBuffLen	Maximum length of the buffer (including room for NULL terminator).

Comments

This function is useful for converting an error code to a short text message that can be displayed to the user for informational purposes.

It is advised that **QAErrorMessage** is called after any function which returns an error code, as the text message might help you identify the cause of the error.

QASystemInfo

Returns detailed information about the system usage of the QAS Batch API library.

Prototype

```
INTRET QASystemInfo
  (INTVAL viLineNo,
  STRREF rsBuffer,
  INTVAL viBuffLen);
```

Parameters

Argument	Explanation
viLineNo	Line number being accessed (or negative if resetting).
rsBuffer	Buffer to receive text line.
viBuffLen	Buffer length (including room for NULL terminator).

Return values

Either: 0 if call successful

Or: negative for an invalid line number

Comments

You can only call this function once the API has been started.

The system information text contains detailed information about the QAS Batch API library, how it is configured, and the resources that it has grabbed from the operating system. The text is split over several lines and so has to be read one line at a time. A buffer size of 80 bytes will be sufficient to guarantee that no lines are truncated.

When the first line is read, the library generates an internal copy of the system information text. It is important that this copy is reset once all the lines have been read otherwise the allocated memory will not be freed. Below is a C example that prints the system information text.

C example that prints out QASystemInfo text:

```
void PrintSystemInfo(VOIDARG)
{
  char sBuffer[80];
  int iLineNo;
  /* read each line in turn */
  for (iLineNo = 0;
    QASystemInfo(iLineNo, sBuffer, sizeof(sBuffer)) == 0;
    iLineNo++)
  {
    puts(sBuffer);
  }
  /* reset in order to free memory */
  QASystemInfo(-1, NULL, 0);
}
```

If you run the example above, you get a result similar to this:

```
Program: QABWVED
Copyright: Experian Ltd
Release: 4.00(07)
Platform: Windows 32-bit
Libraries: QAKRNXF 2.23(213)
 QATOKXD 1.00(3)
 QAUFWXF 4.00(208)
 QACOMXD 1.01(152
 QASLGXD 1.00(3)
 QASLHXD 1.00(2)
 QASLUXD 1.00(2)
 QAHSGXD 3.00(89)
 QAGENXD 1.00(4)
 QAHSVXD 3.20(84)
 QALICXD 1.00(10)
 QAHCLXD 3.00(60)
 QACDIXD 4.00(317)
 QADC2XD 2.40(47)
 QADS2XD 2.00(14)
 QAWD2XD 2.40(66)
 QAUSGXD 4.00(109)
 QALL2XD 2.40(81)
 QAUTDXD 4.00(129)
 QASLCXD 1.00(2)
 QATIXXD 4.00(110)
 QAZLSXD 1.01(41)

Config: qaworld.ini
```

```
Section: ALL

Licensed to: <unknown>
Serial No: <none>
Days: No limit
Users: No limit
Dongle: <none>

Prog Dir: C:\Experian\QAS Batch API
Home Dir: C:\Experian\QAS Batch API
Data Dir: C:\Experian\QAS Batch API
Temp Dir: C:\DOCUME~1\chrisr\LOCALS~1\Temp
Log File: <disabled>

Memory: 69451515 (allocs=2592)
Data:  69323428 (free=760)
Blocks: 4816 (free=68)
```

QAS Batch API Configuration

Before you can perform any searches with the QAS Batch API, you need to specify how and where QAS Batch API will search for input addresses and the format in which output addresses are returned.

QAS Batch API bases these processing decisions on a configuration (INI) file. This file contains many default settings which govern the basic processing that QAS Batch API does, and allows you to define options such as the dataset(s) you search in, cleaning options, and how the output address should look.

The configuration file which contains these settings by default is called qaworld.ini. Within this file, configuration information is stored within layouts. You can create multiple layouts within this configuration file, to encapsulate several different processing options and address formats.

However, if you would prefer to store layout information separately from other global settings, you should do so within a new INI file (custom.ini, for example) and specify its name in your call to **QABatchWV_Open**.

The default handling of custom INI files is to only load the output formatting settings as described in "Setting The Output Address Format" on page 172. It is possible to start Batch API in **Session Mode** by using the qabwvflags_SESSION flag in **QABatchWV_Startup**. In Session Mode all layout settings will be loaded from the custom INI file.

In addition to layout settings, the file qawserve.ini contains settings pertaining to the installed datasets. This file is automatically used when the API is initialised. You should not rename it, move it or attempt to call it with any of the API functions.

Format Of A Configuration File

A configuration file can contain several layouts, each comprising a set of instructions. To view a configuration file, such as the main qaworld.ini file, use a text editor such as Notepad. Do not use a formatting editor such as Microsoft Word because it will corrupt the configuration file with its own formatting codes if you save it.

The layouts within the configuration file have their titles in square brackets:

[layout name**]**

For example, the supplied layout for France is called:

[FRX]

It is advised that you do not alter anything within the supplied layouts. To create alternative settings, copy the layout into a new layout and then make changes to it.

Within the qaworld.ini file, layout names define the beginning of each layout. Layout names must be enclosed within square brackets and be left-justified. A layout ends when a new layout name is declared. The final layout is terminated by the end of file.

Each layout comprises a set of instructions in the form of keyword assignments, like this:

keyword=value

A keyword is the name of a setting. It can consist of any combination of letters and digits in uppercase or lowercase, and it must be followed immediately by an equals sign (=), which introduces the value assigned to the keyword. The value can be an integer, a string, or a special symbol, depending on the type of setting. Note that there should be no space between the '=' and *value*.

> You must not alter any keyword assignments in qawserve.ini or qaworld.ini apart from those documented in this manual.

Not all entries have to be keyword assignments. You can add comments by prefixing the comment with a semi-colon (;).

The keyword assignments can come in any order. A typical keyword assignment looks like this:

```
AUSAddressLineCount=6
```

This example tells QAS Batch API to create an output address consisting of six lines.

Configuring QAS Batch API

There are three steps to configuring the QAS Batch API process:

1. Ensure that one or more datasets have been installed to the required location.
2. Specify which of these datasets should be used in address matching, and how the matching process should proceed.
3. Give QAS Batch API a format in which to return matched addresses.

The first step involves checking settings in qawserve.ini. The remaining two steps require you to specify keyword values in qaworld.ini.

QAWSERVE Settings

Checking Dataset Installation

Open the file qawserve.ini in a non-formatting text editor. This file can be found in the same directory as the library files.

The standard default settings are listed under [QADefault]. This layout includes a setting called `InstalledData`, which can be used to check dataset installation. You should not need to alter it, but you should check that it meets your requirements.

InstalledData

InstalledData={identifier},{path}

Default: Must be explicitly set

Purpose: This keyword lists the installed datasets by a three letter identifier and location. These datasets are the ones installed by the setup program or copied across from the supplied data CDs/DVDs. If you wish to change or add to them, you should run the setup program again or copy them from the supplied medium. Note that if you are also using Additional Datasets, they do not need to be listed in this setting.

If you have more than one dataset installed, the first dataset appears directly after the = sign, and each subsequent dataset appears on a new line preceded by a + sign. For every line that you have specified here, you should also add a line in the `DataMappings` setting.

If you need to move a dataset, you should update this setting accordingly.

Example: If you have installed the UK, Australia and Netherlands datasets in C:\Program Files\QAS\Data, this setting would appear as follows:

```
InstalledData=GBR,C:\Program Files\QAS\Data\
+AUS,C:\Program Files\QAS\Data\
+NLD,C:\Program Files\QAS\Data\
```

DataMappings

DataMappings={identifier},{dataset/group name},{dataset identifier+additional datasets}

Default: Must be explicitly set

Purpose: This keyword allows you to map combinations of datasets and additional datasets to use for different cleaning runs. You choose an identifier and dataset/group name then specify the dataset and related additional datasets you want to include for each mapping. The identifier must be a 3-character alphanumeric code.

If you want to set up more than one mapping, the first should appear directly after the = sign, and each subsequent mapping should appear on a new line preceded by a + sign. If you add or remove datasets in the `InstalledData` setting, you should update this setting accordingly.

If you specify multiple additional datasets, you can set the order of precedence with the `DatasetPrecedenceOrder` setting.

Example 1: If your `InstalledData` setting includes the UK, Australia and Netherlands datasets, and you also have the United Kingdom Names and Electricity additional datasets, and you want to search against different combinations of data simultaneously, you might use this setting as follows:

```
DataMappings=GBR,United Kingdom,GBR
+GBN,UK With Names,GBR+GBRNAM
+GBE,UK With Electricity,GBR+GBRELC
+GBA,UK With Names And
Electricity,GBR+GBRNAM+GBRELC
+AUS,Australia,AUS
+NLD,Netherlands,NLD
```

Example 2:

> This example is only relevant when using GBR data with additional Suppression data.

If your `InstalledData` setting includes the UK dataset, and you also have the United Kingdom Suppression additional datasets, and you want to search against different combinations of data simultaneously, you might use this setting as follows:

```
DataMappings=GBR,United Kingdom,GBR
+GBS,United Kingdom With Suppression,
GBR+GBRABC+GBRABS+GBRMSS+GBRNCA+GBRUSS+GBRMOR
+GBRMPS
+GBM,United Kingdom With Suppression
Movers,GBR+GBRABC+GBRABS+GBRNCA+GBRUSS+GBRGSF
+GBD,United Kingdom With Suppression
Deceased,GBR+GBRMSS+GBRMOR+GBRUSS+GBRTBR
+GBP,United Kingdom With Suppression
Preferences,GBR+GBRMPS+GBRTPS
```

Example 3:

> This example is only relevant when using AUS data with additional Suppression data.

If your `InstalledData` setting includes the AUS dataset, and you also have the Australia With Suppression additional dataset, and you want to search against different combinations of data simultaneously, you might use this setting as follows:

```
DataMappings=AUS,Australia,AUS
+AUX,Australia With Suppression,AUS+AUSMOR
```

CorrectAApiLoc

This information is only relevant when using USA or Canada data.

CorrectAApiLoc={path}

Default: Must be explicitly set

Purpose: This setting is required if you are using the USA or CAN data. The setting specifies the location of the certified address matching engine supplied on the supplementary USA or CAN QAS Batch data disk (Windows) or separately (UNIX). You must ensure that {path} is the location of the directory containing "CorrectA.dll" (Windows) or the "libCorrectA " shared object file (UNIX).

You do not need to use this setting if the CorrectA library is in the same location as your core QAS Batch API libraries.

Example: If the certified address matching engine was copied to the /Data/USA directory, you would use the following setting:

```
CorrectAApiLoc=/Data/USA
```

CorrectADataLocUSA

This information is only relevant when using USA data.

CorrectADataLocUSA={path}

Default: Must be explicitly set

Purpose: This setting is only required if you are using the USA data. The setting specifies the location of the supplementary USA QAS Batch data files. The setting will be set by the installation program on the USA QAS Batch data disk. If you are a UNIX user or copy the data files manually, you must ensure that {path} specifies the location of the parent directory containing the data files.

Example: If the supplementary USA QAS Batch data was copied to the /Data/USA directory, you would use the following setting:

```
CorrectADataLocUSA=/Data/USA
```

CorrectADataLocCAN

This information is only relevant when using Canada data.

CorrectADataLocCAN={path}

Default:	Must be explicitly set
Purpose:	This setting is only required for CAN data. It specifies the location of the supplementary CAN QAS Batch data files. The setting will be set by the installation program on the CAN QAS Batch data disk. If you are a UNIX user or copy the data files manually, you must ensure that {path} is the location of the parent directory containing the data files.
Example:	If the supplementary CAN QAS Batch data was copied to the /Data/CAN directory, you would use the following setting:

```
CorrectADataLocCAN=/Data/CAN
```

QAWORLD Settings

Defining Processing Options

These settings appear in the qaworld.ini file, which is the configuration file called with **QABatchWV_Open** by default.

Settings which are found in the [QADefault] section are general settings which are specified once and apply to all layouts. Other settings can be specified for each layout as required, either in qaworld.ini or your preferred layout configuration file. For more information about configuration files see page 147.

CountryBase

CountryBase={identifier 1} ... {identifier X}

Default: Must be explicitly set

Purpose: This keyword must be specified for each layout in the configuration file.

This keyword is used to list the data mapping identifiers that QAS Batch API should use for searching. You can include any of the mapping identifiers which have been set up with the `DataMappings` keyword in qawserve.ini.

The first identifier in the list is made the default. QAS Batch API will search using that mapping if the country of the input address cannot be identified, for example if it does not include a country name. If you do not want to set a default, use the code 'NUL' in place of the first identifier. The identifiers should appear on the same line, separated from the next by a space.

It is not possible to use multiple core country datasets together in the same QAS Batch API run; for example GBR, APR and LPG, or AUS and AUG. If this setting contains more than one core dataset from one country, an error will be returned from the **QABatchWV_Open** function. For more information, see page 129.

Example 1: If you want your QAS Batch API layout to use your AUS data mapping by default, but also have the option to search using GBR and DEU mappings, this keyword would be set as follows:

```
CountryBase=AUS GBR DEU
```

Example 2: You can configure QAS Batch API to run without a default mapping even if only one country is configured for cleaning. To search using your IRL mapping but prevent QAS Batch API using it by default, you would set the keyword as follows:

```
CountryBase=NUL IRL
```

This setting means QAS Batch API will only attempt to match the address if the country is identified. If the country is not identified, a match type of 'D' is assigned to the record, instead of QAS Batch API using the default data mapping.

CountryRevert

CountryRevert={Boolean}

Default: FALSE

Purpose: This setting controls the process of country spotting.

Country spotting is used by QAS Batch API to identify which dataset should be searched against. Occasionally, this can result in matching errors.

For example, if this keyword is set to FALSE (or not present) country spotting will be active. In this case, the following input address would lead QAS Batch API to search for an address in Spain:

Roadside Cottage, Cluer, Isle of Harris, HS3 3EP, Esp

However, if ESP (Spain) data is not present in your `CountryBase` setting, this would cause QAS Batch API to return a **C** match code"Match Success" on page 30.

Setting this keyword to TRUE disables country spotting and instructs QAS Batch API to disregard any country identifiers unless they correspond to a dataset that you have installed. If no corresponding datasets are present, the country identifier is disregarded and QAS Batch API will attempt to match the address against your default identifier only. In the example above, this would return a successful match if GBR data existed in your `CountryBase` setting.

See page 156 for more information about the `CountryBase` keyword and the default identifier.

Example: If you want QAS Batch API to restrict matching to countries in the `CountryBase`(see page 156), you should set this keyword to TRUE:

```
CountryRevert=True
```

This setting is particularly useful if you know that your input file contains addresses from one country only, or only from the countries specified in your `CountryBase` keyword.

LogErrors

LogErrors={Boolean}

Default:	FALSE
Purpose:	This keyword applies to all layouts and must be specified in the [QADefault] section of the configuration file. If this keyword is set to TRUE, QAS Batch API will record all errors in the log file, which is specified with the `LogFile` setting. Setting this keyword to FALSE disables logging.
Example:	Use this setting to enable logging:

```
LogErrors=TRUE
```

LogFile

LogFile={filename}

Default: None

Purpose: This keyword applies to all layouts and must be specified in the [QADefault] section of the configuration file.

`LogFile` enables you to specify a log file to which any errors that occur when you call API functions are written. These errors are only written if you set `LogErrors=TRUE` as well as specifying the name of the file you want to write to with `LogFile`.

It is recommended that you create a log file when integrating the API.

Example: The following creates a log file called error.log in the same directory as the program files.

```
LogFile=error.log
```

BatchTimeout

BatchTimeout={integer}

Default:	5000
Purpose:	This keyword applies to all layouts and must be specified in the [QADefault] section of the qaworld.ini configuration file. You can set the length of time in milliseconds that QAS Batch API spends on a search with this keyword. The timeout that you set comes into operation with the function **QABatchWV_Clean**. The default setting of 5000 sets a timeout period of 5 seconds (5,000 milliseconds).
Example:	To set a timeout period of 30 seconds (30,000 milliseconds), make this assignment: `BatchTimeout=30000`

CleaningAction

CleaningAction={string value}

Default: Address

Purpose: This keyword must be specified for each layout in the configuration file.

This keyword determines the action performed by QAS Batch API when an address has been matched. There are five possible values for this keyword:

None	No address is returned.
Update postcode	The postal code is checked to see if it has been recoded.
Address	Returns a full address with postal code.
Enhanced	Returns a full address plus additional information; that is, components of the input address which could not be matched. See below for an example.

> This option is not available if USA is the only dataset configured.

Example: The following setting tells QAS Batch API to return non-matched leading elements of the input address in the output address:

```
CleaningAction=Enhanced
```

Given this input address for the Australia dataset:

John Smith, Suite 1, Level 9, 60 Miller St, North Sydney, NSW, 2060

QAS Batch API returns:

John Smith,
Suite 1, Level 9
60 Miller St
North Sydney NSW 2060

If `CleaningAction` is set to anything except 'Enhanced', the component 'John Smith' would be returned as an unused address line (see the function **QABatchWV_GetUnusedInput**) as it is not stored in the Australia dataset, and is therefore not part of the matched address.

SearchLevel

This information is only relevant when using GBR or LPG data.

SearchLevel={string value}

Default:	Full
Purpose:	This keyword can be specified for each layout in the configuration file. This keyword controls how intensely QAS Batch API will search for address matches. You should consider that more stringent matching will take longer whereas less stringent matching will take less time. There are three possible values for this keyword:

Full	Performs a full search, balancing the quality of throughput with the time taken. This is the recommended mode for most situations.
Extended	Performs an extensive search to find address matches for particularly inexact address data. This search mode is the most thorough and takes the most time.
Verification	Performs a rapid lookup of each input address using key identified address components (such as a supplied postcode and premise information). This mode maximises the throughput of data and is particularly effective if your address data is relatively clean.

Example: The following setting tells QAS Batch API to use the recommended Full search mode:

```
SearchLevel=Full
```

CacheMemory

CacheMemory={Integer}

Default:	0
Purpose:	This keyword applies to all layouts and must be specified in the [QADefault] section of the configuration file. This keyword is used to specify the amount of system memory (in MB) that QAS Batch API can use for data caching. Caching is disabled by default, but you can use this keyword to increase the performance of QAS Batch API by allowing it to use system memory. If your system has less than 64MB of memory you should not use this setting.
Example:	The following setting would allow QAS Batch API to use up to 1024MB of memory for data caching: `CacheMemory=1024`

CorrectACacheLevel

This information is only relevant when using USA or Canada data.

CorrectACacheLevel={String}

Default:	None
Purpose:	This setting determines the level of data caching which should be used for USA and/or Canadian Batch cleaning. {String} can take one of the following values:

- ALL (QAS Batch will attempt to cache all the data required)
- NONE (QAS Batch will not cache any of the data required)
- AUTO (QAS Batch will determine how much of the data to cache).

If you use the ALL setting, you must ensure you have sufficient RAM available (at least 3GB) otherwise you will receive an out of memory error. You should also ensure that sufficient memory has been specified by the `CacheMemory` setting to cache the address data. Any other datasets including USA data, will be cached in the remaining memory.

If you are using the USA dataset, more detailed system requirements and performance-related tips can be found in the USA Data Guide.

NamesTolerance

This information is only relevant when using GBR or AUS data, with additional Names or Suppression data.

NamesTolerance={String}

Default: Blank

Purpose: This keyword can be specified for each layout in the configuration file.

This specifies how strict QAS Batch API should be when matching names. For a detailed description of the differences between these options, see "Appendix E: Names Matching Tolerance Levels" on page 200.

The cleaning process will be faster if you allow fewer errors in the supplied names information.

The following options are available:

Relaxed	Allows several errors in the supplied names.
Standard	Allows one or two errors in the supplied names. This is the recommended value.
Intermediate (GBR only)	Minor variations in the supplied names are permitted.
Exact	No errors in the supplied names are permitted.

Example: The following setting means that QAS Batch API does not allow any errors in the input name information:

```
NamesTolerance=Exact
```

OemCharacterSet

OemCharacterSet = {String} [NoDiacritics]

Default:	ANSI
Purpose:	This keyword applies to all layouts and must be specified in the [QADefault] section of the configuration file.

The QAS Batch API includes support for character sets that contain non-standard characters, such as diacritics (e.g. accents and umlauts). The API also provides the ability to remove diacritic characters on address output.

String indicates the generic character family. If 'NoDiacritics' is specified, all diacritic characters are removed on output from API routines.

The following character set families are supported by QAS Batch API. They are 8-bit character sets and can support diacritics and multiple code pages:

Family	**Description**
ANSI	The character sets as defined by the American National Standards Institute.
ASCII	As above but without diacritics.
DOS	DOS code page 850.

DatasetPrecedenceOrder

DatasetPrecedenceOrder={additional dataset}

Default: Blank

Purpose: This keyword can be specified for each layout in the configuration file.

If you have configured more than one additional dataset in any line of the `DataMappings` setting, this keyword can be used to specify the precedence order.

QAS Batch API matches an input address against each additional dataset individually. In cases where equally good matches are found in multiple configured datasets but the matched addresses are different, you can specify the precedence order of which dataset match to return. If you do not specify the precedence order with this setting then QAS Batch API will return a partial match including only the elements common to all datasets.

Example: If you have a GBR data mapping which includes Electricity, Names and Business data, you might use this setting as follows:

```
DatasetPrecedenceOrder=GBRELC
+GBRNAM
+GBRBUS
```

Certification

This information is only relevant when using USA or AUS data.

Certification={Boolean}

Default:	Yes
Purpose:	This setting determines whether QAS Batch should run in Certified mode. For the USA dataset,certified mode ensures that the results conform to the CASS rules, including the mandatory use of Delivery Point Validation (DPV). In certified mode, QAS Batch API will return a +4 code **only** when the address has been DPV-confirmed. If an address is not DPV confirmed, a +4 code will not be returned, and by extension, any DataPlus items you have configured as part of the address output format may not be returned either. The certified cleaning mode includes USPS SuiteLink as a standard part of the CASS certified cleaning process. This uses USPS SuiteLink data to enhance your organisation addresses where possible. For the AUS dataset, certified mode ensures that the results conform to the AMAS rules, and Address Delivery Point Identifier (DPID) or Default Identifier (DID) is allowed to be returned in the output.
Example:	If you want QAS Batch to run in Certified mode, use the following:

```
Certification=Yes
```

Setting The Input Address Format

The two settings `InputLineCount` and `InputLineN` should be set up in each layout in qaworld.ini, or your preferred layout configuration file, to describe the contents of the input address fields as they are passed into the QAS Batch API engine. Doing so can enhance engine processing, and increase the accuracy and speed of address matching.

InputLineCount

InputLineCount={integer}

Default:	Blank
Purpose:	Use this setting to define the number of lines your input addresses contain. The format of each individual address line is specified with the `InputLineN` setting. Note that any lines not covered by your specified input address format will not be constrained to match to specific address element types.
Example:	The following tells QAS Batch API that each input address contains four lines: `InputLineCount=4`

InputLineN

InputLine1={element list}
InputLine2={element list}
...
InputLineN={element list}

Default:	Blank
Purpose:	This specifies which address element is to appear on which line in the input address. If you know that a line in your database always contains the same type of address element, (for example, if line 4 always contains a town name), you can mark that line with an element code. This improves the speed and quality of matching. **{element list}** represents a comma-separated list of element codes, either dataset-specific or generic. By specifying element codes, you tell the QAS Batch API which elements to expect on that line (if the elements exist in the matched address).

It is important not to supply non-address information to the QAS Batch API (where possible), as this may compromise the address matching process. In particular, supplying non-address numeric values (such as DPIDs in Australia) can cause confusion when matching against premises information. If your data includes non-address information which cannot be identified using a generic or dataset-specific element code then it should not be supplied to the QAS Batch API.

Example: The following instructs QAS Batch API to expect premises details on line 1 of the input address:

```
InputLine1=P00
```

The generic element codes are listed below, in the order in which they will appear in a formatted address (unless you fix them in a different order on the address line).

Order	Element Code	Description
1	N00	Name
2	O00	Organisation
3	P00	Premises
4	S00	Street
5	B00	PO box
6	L00	Locality
7	C00	Postal code
8	X00	Country name

If you are using QAS Batch API with Suppression data, you must set at least one of the input lines to contain generic names information.

If you are using TPS data, you must also ensure that one of the input lines contains a telephone number.

See the Data Guide supplied with your data for a list of dataset-specific element codes.

Setting The Output Address Format

These settings appear in each layout in the qaworld.ini file, which you should call with **QABatchWV_Open**. The keywords in this section can be prefixed by [identifier]. This makes it possible to define address formats for more than one data mapping within a single configuration layout. Identifiers are set up using the `DataMappings` keyword. For example, the setting `CapitaliseItem` would become `AUSCapitaliseItem` for the default Australian data mapping.

See "DataMappings" on page 152 for more information.

AddressLineCount

[identifier]AddressLineCount={integer}

Default:	0
Purpose:	This defines the number of lines in the formatted output address. The format of each individual line is specified with the `AddressLineN` keyword. The number of lines you specify should include any lines of address and DataPlus information.
Example:	The following setting tells QAS Batch API to produce formatted output addresses of six lines for the 'AUS' data mapping. `AUSAddressLineCount=6`

AddressLineN

[identifier]AddressLine1=W{width},{element list}
[identifier]AddressLine2=W{width},{element list}
...
[identifier]AddressLineN=W{width},{element list}

Default:	Blank

Purpose: This specifies which address element or DataPlus information is to appear on which line. W signifies that the number that follows it is the maximum width of the line in characters, and **{element list}** is a comma-separated list of element code. Address element codes are listed in the Data Guide supplied with your dataset. If you do not specify **{element list}** QAS Batch API will automatically spread the standard address over the available lines. By specifying element codes, you force QAS Batch API to place the elements on a particular line (if the element exists in the matched address).

> Unless explicitly configured, names information will always precede address information.

By default, if an input address contains a recognised alternative version of an official address element it will be replaced by the official version. To configure QAS Batch API to retain the version in the input address, append the element with a #. For example, `NZLAddressLine3=L21#`.

If you want to return DataPlus information use the base name and the element name in place of an address element (see example 2).

You can allow QAS Batch API to insert other suitable elements before, after or between fixed elements by using the format specifier '...'.

Example 1: The following instructs QAS Batch API to give line 1 of the 'NZL' output address a maximum width of 30 characters:

```
NZLAddressLine1=W30,S21,...
```

The Whole Street element is fixed to the line, and any subsequent elements can also appear on the line if they fit there.

Example 2: This example tells QAS Batch API to give line 6 of the 'AUS' output address a maximum width of 40 characters, and fix the description part of the MOSAIC DataPlus set to that line:

```
AUSAddressLine6=W40,AUSMOS.Desc
```

CapitaliseItem

[identifier]CapitaliseItem={element list}

Default: Blank

Purpose: This keyword defines which address elements should appear in upper case in the formatted address. The value of the keyword is a list of element codes separated by spaces.

Example: The following setting means that the building name and country name elements will be capitalised:

```
AUSCapitaliseItem=P21 X11
```

AbbreviateItem

[identifier]AbbreviateItem={element list}

Default: Blank

Purpose: This keyword defines which address elements should be abbreviated in the formatted address. The value of the keyword is a list of element codes separated by spaces.

Example: The following setting means that the Australian state name will be abbreviated:

```
AUSAbbreviateItem=L12
```

ConditionalFormat

[identifier]ConditionalFormat={text string}

This information is only relevant when using GBR with additional Business data, or USA data.

Default: ExperianOrgPref

NormCity

Purpose: This setting has a different function depending on the dataset being used.

For the GBR with additional Business dataset this setting allows the user to specify whether to display the PAF or Experian organisation data, or a combination of both.

There are four possible values for this keyword:

ExperianOrgPref (default)	The Experian organisation name takes priority, but the PAF organisation name will be used if there is no Experian equivalent for the address.
ExperianOrgOnly	Only Experian organisation names will be returned, and PAF organisation names will be suppressed if there is no Experian equivalent for the address.
PostOrgPref	The PAF organisation name takes priority, but the Experian organisation name will be used if there is no PAF equivalent for the address.
PostOrgOnly	Only PAF organisation names will be returned, and Experian organisation names will be suppressed if there is no PAF equivalent for the address.

For the USA dataset this setting allows the user to specify whether to return the full city name or the abbreviated city name.

There are two possible values for this keyword:

NormCity (default)	The full city name will be returned.
AbbCity	The abbreviated city name will be returned. This will have a maximum of 13 characters.

AbbreviateAddr

This information is only relevant when using USA data.

AbbreviateAddr={Boolean}

Default:	No
Purpose:	This setting allows you to limit the first line of output addresses to a maximum of 30 characters. This setting works with QAS Compatibility Formatting mode and in CASS Certified Mode.
Example:	If you want to limit the first line of output addresses to a maximum of 30 characters, use the following:

```
AbbreviateAddr=Yes
```

CompatibilityFormatting

This information is only relevant when using USA data.

CompatibilityFormatting={Boolean}

Default:	No
Purpose:	This setting determines whether QAS Batch should run in QAS Compatibility Formatting mode. Compatibility Formatting mode is not certified, and will not use Delivery Point Validation, but does offer increased flexibility in matching and output address formatting.
Example:	If you want QAS Batch to run in QAS Compatibility Formatting Mode, use the following:

```
CompatibilityFormatting=Yes
```

MultiValueDPSeparator

MultiValueDPSeparator={string}

Default:	|
Purpose:	This keyword can be specified for each layout in the configuration file. If you are using United Kingdom with Gas or Electricity data, QAS Batch API will return all multiple meter numbers. This keyword can be used to change the default delimiter. The delimiter cannot be alphanumeric. The API will verify the setting, and use the default character if an invalid character is used.
Example:	If you want the multiple meter numbers to be returned separated by a comma, use the following:

```
MultiValueDPSeparator=,
```

QALICN Settings

Licence key information is located in qalicn.ini. This .ini file can be used for adding, deleting and viewing licence information.

The licence keys can be found on the despatch note supplied with the data.

Each licence key should be inserted on a separate line.

Appendix A: Error Code Listing

Below is a full list of error codes and their descriptions. Call the system function **QAErrorMessage** to retrieve the top level message associated with the returned error code, and **QAErrorLevel** to ascertain whether the error is serious or a warning. If you require more specific error information, **QAErrorHistory** may be repeatedly called.

It is strongly recommended that checks for specific return errors are not hardcoded into your integration.

Code	Internal name	Meaning
-1000	qaerr_FATAL	Fatal error
-1001	qaerr_NOMEMORY	Out of memory
-1002	qaerr_INITINSTANCE	Invalid multithreading instance
-1005	qaerr_INITOOLARGE	INI file too large
-1006	qaerr_ININOEXTEND	Could not extend INI file
-1008	qaerr_FILETOOLARGE	File too large
-1009	qaerr_FILECHGDETECT	Cannot detect file changes
-1010	qaerr_FILEOPEN	File not found
-1011	qaerr_FILEEXIST	File already exists
-1012	qaerr_FILEREAD	File read failure
-1013	qaerr_FILEWRITE	File write failure
-1014	qaerr_FILEDELETE	Could not delete file
-1016	qaerr_FILEACCESS	File access denied

Code	Internal name	Meaning
-1017	qaerr_FILEVERSION	Incorrect version of data file
-1018	qaerr_FILEHANDLE	Maximum number of files open
-1019	qaerr_FILECREATE	Could not create file
-1020	qaerr_FILERENAME	Could not rename file
-1021	qaerr_FILEEXPIRED	Data file has expired
-1022	qaerr_FILENOTDEMO	Can only access demonstration data
-1023	qaerr_FILETIMEGET	Failed to obtain file timestamp
-1024	qaerr_FILETIMESET	Failed to modify file timestamp
-1025	qaerr_READFAIL	Disk read failure
-1026	qaerr_WRITEFAIL	Disk write failure
-1027	qaerr_BADDRIVE	Invalid drive
-1028	qaerr_BADDIR	Invalid directory
-1029	qaerr_DIRCREATE	Could not create directory
-1030	qaerr_BADOPTION	Invalid command line option
-1031	qaerr_BADINIFILE	Could not locate INI file
-1032	qaerr_BADLOGFILE	Could not create log file
-1033	qaerr_BADMEMORY	Invalid memory configuration
-1034	qaerr_BADHOTKEY	Invalid hot key
-1035	qaerr_HOTKEYUSED	Hot key already in use
-1036	qaerr_BADRESOURCE	Could not locate language file
-1037	qaerr_BADDATADIR	Invalid data directory
-1038	qaerr_BADTEMPDIR	Could not create temporary directory
-1040	qaerr_NOTDEFINED	Entry not defined
-1041	qaerr_DUPLICATE	Entry duplicated
-1042	qaerr_BADACTION	Invalid action
-1045	qaerr_BADDATE	Invalid date or time
-1046	qaerr_BADTIMEZONE	Invalid time zone
-1050	qaerr_CCFAILURE	Copy control failure
-1051	qaerr_CCBADCODE	Invalid copy control code

Code	Internal name	Meaning
-1052	qaerr_CCACCESS	Copy control access denied
-1053	qaerr_CCNODONGLE	Dongle not configured
-1054	qaerr_CCNOUNITS	No units left on meter
-1055	qaerr_CCNOMETER	Meter not initialised
-1056	qaerr_CCNOFEATURE	Feature not supported
-1057	qaerr_CCINVALID	SoftKey integrity failure
-1058	qaerr_CCNODCHKFAIL	Node lock check failure
-1060	qaerr_CCINSTALL	Copy control not installed
-1061	qaerr_CCEXPIRED	Allowable time expired
-1062	qaerr_CCDATETIME	Invalid copy control date or time
-1063	qaerr_CCUSERLIMIT	Number of concurrent users exceeded
-1064	qaerr_CCACTIVATE	Copy control installed but not activated
-1065	qaerr_CCBADDRIVE	Invalid copy control drive
-1066	qaerr_CCREGISTER	Product must be registered
-1070	qaerr_UNAUTHORISED	Not authorised
-1074	qaerr_NOLOCALEFILE	Locale file not found
-1075	qaerr_BADLOCALEFILE	Invalid locale file
-1076	qaerr_BADLOCALE	Unknown language/country
-1077	qaerr_BADCODEPAGE	Unknown code page
-1078	qaerr_RESOURCEFAIL	Resource lookup failure
-1080	qaerr_NOTHREAD	Could not create thread
-1081	qaerr_NOTLSMEMORY	Out of thread-local-storage
-1090	qaerr_NOTASK	Could not create task
-1091	qaerr_LOADLIBRARY	Could not load DLL or shared object
-1094	qaerr_API_WORD	Input value exceeds 16 bits
-1095	qaerr_API_DWORD	Input value exceeds 32 bits
-1096	qaerr_CHARSET	Invalid input characters

Code	Internal name	Meaning
-1097	qaerr_BUFFERTRUNC	Buffer value truncated
-3801	qaerr_FORMATSYNTAX	Incorrect formatting syntax
-3802	qaerr_TOOMANYADDRLINES	Too many address lines requested
-3803	qaerr_INVALIDADDRESSLINE	Address line out of range
-3804	qaerr_NOFORMATSPEC	No format spec in INI file
-3805	qaerr_FORMATOVERFLOW	Element(s) have overflowed
-3806	qaerr_FORMATTRUNCATED	Element(s) are truncated
-3809	qaerr_DPFAILURE	One or more DataPlus sets failed to open
-3811	qaerr_ COUNTRYBASEMISMATCH	Invalid data mapping in layout
-4362	qaerr_NODATAMAPPINGS	No valid data mappings were found
-4363	qaerr_MISSINGDATAMAP	Expected data mapping not available
-4364	qaerr_NOLAYOUTSELECTED	No layout selected
-4576	qaerr_DATASETSAMECOUNTRY	Multiple base datasets configured for the same layout
-4577	qaerr_NZLDELMISSING	New Zealand Deleted Records additional data missing
-4599	qaerr_ INPUTLINECOUNTREQUIRED	No InpuLineCount setting has been found for Suppression
-4587	qaerr_NOMODECHANGE	Estimate Mode cannot be changed when any counter reports are open
-4588	qaerr_MORECLICKSNEEDED	The function being called cannot be run until more clicks have been used
-4589	qaerr_BADAUDITCODE	The audit code entered is invalid
-4593	qaerr_ NOACTIVESUPPRESSIONSETS	No active Suppression datasets found
-4594	qaerr_ NOSUPPRESSIONHIERARCHY	Suppression hierarchy not found

Code	Internal name	Meaning
-4595	qaerr_DATAPLUSNOTAVAILABLE	Suppression DataPlus configured when Suppression is switched off
-4596	qaerr_INVALIDHIERARCHY	Hierarchy must contain all active Suppression datasets
-4597	qaerr_ MULTIPLEDATAPLUSPERLINE	Multiple suppression DataPlus per line with permanent hierarchy enabled
-4598	qaerr_ SUPPRESSIONNAMEREQUIRED	One of the input fields must contain names information
-4785	qaerr_DDFDEFINITION	Definition is incorrect in data
-4786	qaerr_INVALIDPRIORITY	Priority rules is invalid
-4789	qaerr_PRIORITYEVALAMBIGUITY	Priority rule evaluation is ambiguous
-4960	qaerr_SLCASFUNC	Invalid function call to CorrectAddress engine
-4961	qaerr_SLCASFUNCMAP	CorrectAddress function not mapped
-4962	qaerr_SLCASDATAEXPIRED	CorrectAddress data expired
-4963	qaerr_SLCASDATA	Failed to open CorrectAddress data files
-4964	qaerr_SLCASEWSDATA	Failed to open EWS files
-4965	qaerr_SLCASSYSTEMERROR	CorrectAddress API issued a system error
-4966	qaerr_SLCASOUTOFMEM	CorrectAddress API out of memory
-4968	qaerr_SLCASINVALIDLIC	Invalid DPV licence key
-4969	qaerr_SLCAPIEXPIRED	CorrectAddress engine has expired
-8601	qaerr_NODEFAULTCOUNTRY	Default country not specified
-8602	qaerr_NOINSTALLEDCOUNTRIES	No installed countries found
-8603	qaerr_DEFAULTNOTINSTALLED	Default country not installed
-8604	qaerr_INVALIDPRIORITY	Error in priority string
-8605	qaerr_CALLPENDING	A call is pending

Code	Internal name	Meaning
-8606	qaerr_NOTRUNNING	The API is not running properly
-8608	qaerr_APIABORTED	The API has shut down
-8609	qaerr_RUNNING	The API is already running
-8610	qaerr_NOHANDLES	No free API handles
-8611	qaerr_BADHANDLE	Handle out of range
-8612	qaerr_NOSEARCHRESULTS	Attempt to retrieve uninitialised results
-8613	qaerr_OUTOFRANGE	Line is out of range
-8614	qaerr_NOCOUNTRYFILE	Country resource file not found
-8615	qaerr_COUNTRYRANGE	Country out of range
-8616	qaerr_NOCOUNTRIES	No countries are active
-8617	qaerr_LAYOUTRANGE	Layout out of range
-8618	qaerr_INVALIDCOUNTRY	Country name was not found
-8619	qaerr_COUNTRYVERSION	Data file version incompatible with product
-8620	qaerr_BADPARAMETER	Invalid API function parameter supplied
-8621	qaerr_PARAMETERTRUNCATED	API function parameter truncated
-8622	qaerr_TOOMANYINPUTLINES	Input line configuration limit exceeded
-8623	qaerr_INVALIDINPUTITEM	Invalid input line item configured
-8625	qaerr_INVALIDLAYOUT	Specified layout is invalid
-8626	qaerr_LICENSINGERROR	Licensing error has occurred with one or more datasets
-8701	qaerr_DATANOTINITIALISED	Data not initialised
-8702	qaerr_ COUNTRYALREADYINSTALLED	Data unexpectedly initialised
-12001	qaerr_TOKFUNC	Unknown tokeniser API request
-12002	qaerr_TOKNOKERNEL	Kernel not initialised
-12003	qaerr_TOKNOCOUNTRYFILE	Cannot find country.ini

Code	Internal name	Meaning
-12004	qaerr_TOKBADCOUNTRYFILE	Bad country.ini format
-12005	qaerr_TOKTOOMANYLINES	Too many input specification lines
-12006	qaerr_TOKINVALIDINPUTITEM	Invalid input item
-12007	qaerr_TOKPOSTCODEFORMAT	Invalid postcode format
-12008	qaerr_TOKFOREIGNCITIES	Unable to load foreign cities
-12009	qaerr_TOKCOUNTRYSPOT	Unable to perform country spotting
-12010	qaerr_TOKCITYSPOT	Unable to perform city spotting
-12101	qaerr_SLUFUNC	Invalid USPS server request
-12102	qaerr_SLUNOHK	No housekeeping instance present
-12103	qaerr_SLUNOUSPSDATA	No USPS data present
-12104	qaerr_SLHNOKERNEL	Kernel has not been started
-12105	qaerr_TPAPINOTLOADED	Required third party API not loaded

Appendix B: Data Checker Utility

You can check the integrity of Experian Data Quality data files using **quchkn.exe**.

Quchkn.exe is called as follows:

Windows Syntax	QUCHKN (filespec) QUCHKN -log -? show all command line options
UNIX Syntax	QUCHK (filespec) QUCHK -log -? show all command line options
Example	QUCHKN H:\QAS\DATA\*.*
Description	Performs an integrity check on selected Experian Data Quality data files. Uses a CRC (Cyclic Redundancy Check) to verify that the contents of the data files are not corrupt. Can be used to check for problems with the data or file corruption.
Windows options	The buttons on the dialog once the application has been launched can be used to perform the following actions: **Add...** Add a data file to the list. **Remove...** Remove the selected file from the list. **Check File** Check the currently-selected file. **Check All** Check all the files in the list.
UNIX options	The following command-line arguments are available: **-INI:<file>** Specify configuration file. **-SECTION:<file>** Specify configuration section. **-DATADIR:<dir>** Specify directory for data files. **-LOG:<file>** Specify log file. **-ERRORS** Enable error logging.

Appendix C: Suppression Data – Uses and Benefits

This information is only relevant when using GBR or AUS data with additional Suppression data.

Suppression data contains additional information associated with an address. Specifically, you can clean your records against Suppression data and then return relevant Suppression DataPlus information for any matching addresses in your database. This makes it possible to see easily any names and addresses which may not be useful to your business, and also to enhance your data with additional information.

It is not possible to perform a clean against multiple datasets **and** one or more Suppression dataset(s) at the same time.

There are a number of reasons why certain customers' addresses may be unsuitable for business use. These depend upon the particular Suppression set in use:

Movers

- NCOA Update
- NCOA Suppress
- Absolute Contacts (ABC)
- Absolute Movers (ABS)
- Goneaway Suppression File (GSF)

This data contains details of those people who have recently changed address. Some Suppression data also provides associated forwarding addresses.

Deceased

- Mortality Suppressions (MSS)
- The Bereavement Register (TBR)
- Mortascreen (MOR)
- Australia Mortalities (AUSMOR)
- NCOA Suppress

This data contains the names and address details of people who have passed away.

Preferences

- Mailing Preference Service (MPS)
- Telephone Preference Service (TPS)

This data contains the details of people who have opted not to receive unsolicited mail / telephone calls.

For detailed information about the types of Suppression data that are available with your dataset, refer to the Additional Data Guide that shipped with your Suppression data.

Suppression data can be used in a variety of ways, depending on your business needs. This section describes four of the most suitable and efficient uses of Suppression data within QAS Batch API:

- Generating a high quality mailing list;
- Generating a complete mailing list;
- Generating a Suppression report;
- Adding Suppression information to your data.

Generating A High Quality Mailing List

Effective use of Suppression data enables you to save time and money, by excluding from your mailing list those households or individuals who will choose not to respond, or who are unable to do so.

Suppression data can be used to flag those people who have moved, who are deceased, or who have expressed a preference not to be contacted. In addition, you can filter out poor quality addresses which cannot be matched with confidence against the official postal address files. Removing these details from your mailing list will reduce the size of the list, while retaining its effectiveness.

To generate a high quality mailing list, follow these steps:

1. Clean your data against the official postal address files, in order to maximise the chance of delivery. The addresses in your database may be cleaned while retaining non-address information such as names and departments, depending upon the nature of the records.
2. Clean your data against your Suppression dataset.
3. Produce a file containing only those addresses which are likely to be worth mailing.

   This should include both of the following:

- Addresses which produce Good or Verified matches when cleaned against the official address files. See Match Success for more information about match results.
- Records which were **not** matched against Suppression data.

To keep your address records up to date, it is recommended that you clean your database periodically, and update all Suppression information stored within it.

Generating A Complete Mailing List

To generate a complete mailing list, use QAS Batch API to produce a mailing list containing all addresses from your original database, regardless of quality, omitting only those which were matched against Suppression data.

This list will retain all other input records from your database, including those addresses which could not be matched against the official postal address files.

Generating A Suppression Report

To generate a Suppression report, use the **QABatchWV_CounterReport** function in file mode, using the command:

batwv *<input file name> <output file name> <report file name>*

This function is described in terms of how to perform it using the C test harness supplied with QAS Batch; however, you may choose to use your own implementation instead.

Adding Suppression Information To Your Data

You may find it useful to add Suppression information to your current database, in order to carry out a statistical analysis of your data.

If you have configured Suppression DataPlus in your layout and an address record in your database has been matched against a configured Suppression DataPlus set, you will be charged a click for that match. You will be able to see which Suppression set the record has been matched against, and will also be able to view DataPlus information such as date of death.

Refer to the Suppression Additional Data Guide that shipped with your data for more information about Suppression DataPlus.

Appendix D: Analysing Costs of Suppression Data

If you have cleaned your data against one or more Suppression dataset(s), you will be invoiced for the number of clicks you have used. See About Clicks for more information.

Suppression data can be paid for in multiple ways, and will depend on the meter types associated with each suppression dataset, and which actions are performed. Some datasets will include all of these different meter types, while others may include only one. They include:

- Permanent Flagging
- One-Off Suppression
- Dual Suppression
- Tracking Suppression

For GBR Suppression, the cost will also be affected if you have activated the Suppression Hierarchy (see page 194).

About Clicks

A click is a single count against a meter.

Depending on how an address record is matched against a Suppression dataset, you may be charged for a permanent, one-off (temporary), tracking or dual click. If an address record matches more than one Suppression dataset, the order in which matches should be used (and therefore your costs) depends on the options set in the Suppression Hierarchy.

If an address record is matched against a Suppression dataset, you will be charged for a dual click.

Permanent Clicks

This information is only relevant when using GBR data with additional Suppression data.

Users will be charged a permanent click if all of the following conditions are met:

- The user has set the `UseSuppression` ini setting to **On**. Refer to the United Kingdom with Suppression Data Additional Data Guide for more information about this setting.
- The user has configured QAS Batch API to return at least one Suppression DataPlus item.
- An address record has matched to the Suppression dataset which the configured DataPlus set belongs to.

For more information about configuring Suppression and about Suppression DataPlus, see the United Kingdom with Suppression Additional Data Guide.

One-Off Clicks

This information is only relevant when using GBR data with additional Suppression data.

Users will be charged for a one-off (temporary) click if all the following conditions are met:

- The user has set the `UseSuppression` ini setting to **On**. Refer to the United Kingdom With Suppression Additional Data Guide for more information about this setting.
- Address records in their database have only matched against Suppression datasets for which DataPlus is not configured. The user will not be told which Suppression dataset has been matched, and obviously no DataPlus information will be returned.

Dual Clicks

Dual meters are used when there is no price difference between Permanent and One-Off suppression. If this option is available for a dataset it will usually be the only meter available.

The user will be charged a dual click if they have set the `UseSuppression` ini setting to **On** (refer to the Suppression Additional Data Guide for more information); and either

- They have configured QAS Batch API to return at least one Suppression DataPlus item and an address record has matched to the Suppression dataset which the configured DataPlus set belongs to.

or:

- An address record in their database has matched to the Suppression dataset without having any output DataPlus items configured. The record is suppressed but no DataPlus information is returned.

For more information about configuring Suppression and about Suppression DataPlus, see the Suppression Additional Data Guide that shipped with your data.

Tracking Suppression

This information is only relevant when using GBR data with Tracking Suppression data.

When using Tracking mode with the NCOA Update or Absolute Contacts Suppression datasets, users will be charged one tracking click for returning a forwarding address. Furthermore, if the forwarding address matches against other Suppression datasets, these matches will be charged according to the criteria for permanent and one-off clicks. See the United Kingdom With Suppression Additional Data Guide for more information about NCOA Update and Absolute Contacts data.

Suppression Hierarchy

This information is only relevant when using multiple GBR Suppression additional datasets.

If an address record matches more than one Suppression dataset, the order in which matches should be used (and therefore your costs) depends on the options set in the Suppression hierarchy.

Once a record has been matched against a dataset in the hierarchy, the others further down the list are not considered.

The Suppression hierarchy therefore allows the user to do the following:

- Control in what order they will be charged for clicks, which will minimise their total expenditure;
- Specify some types of Suppression data (e.g. mortalities) above others (e.g. goneaway or movers), so that they can filter some types of Suppression matches from their database without removing others completely.

Notes On Suppression Hierarchies

- Hierarchies are specified within layouts.
- The `SuppressionHierarchy` ini setting is used to specify the datasets that are in the hierarchy, and their order. If the hierarchy is not specified, an error will be returned.
- The datasets listed in the hierarchy must match the sets in the `Suppression` ini setting, or an error will be returned.

- The Suppression hierarchy is always used for one-off (temporary) clicks, but is optional for permanent ones. The `PermanentHierarchy` ini setting is used to activate the hierarchy for permanent clicks.
- When a permanent hierarchy is active, you will only be charged for the highest matching set with DataPlus configured, and DataPlus information will only be returned for that set.
- When you are using NCOA Update or Absolute Contacts data in tracking mode, this operates outside of the permanent Suppression hierarchy, as described below.

> Refer to the United Kingdom with Suppression Additional Data Guide for more information about Suppression hierarchy settings.

Tracking Hierarchy

NCOA Update and ABC data have their own Tracking Hierarchy. The Tracking Hierarchy determines which of the Tracking datasets selected records are searched against first and is completely separate from the Suppression Hierarchy.

If you have returned a match against NCOA Update or Absolute Contacts (ABC) data, these records will not be suppressed. Instead, the forwarding address will replace the original address, ready to be exported or committed back to your database.

You will be charged once for each match against the NCOA Update or ABC data which returns a forwarding address. Subsequent matches against other datasets will be made against the forwarding address, rather than against the previous (obsolete) address.

Paying For Suppression Data

There are a number of steps which you must follow in order to pay for your use of Suppression data. These functions are described in terms of how to perform them using the test harness supplied with this product; however, you may choose to use your own implementation instead.

1. Install QAS Batch API With Suppression data. For more information about how to do this, see "QAS Batch API Installation" on page 5.

2. Use the **QABatchWV_GetAuditCode** function to extract a text-based audit code from the counters file on the disk where QAS Batch API is installed.

   To extract the audit code using the C test harness, start the test harness using the following syntax:

   batwv –audit

   The harness will not start as normal; instead it will output the command to the standard output channel, which can then be displayed or piped to a text file if required. The following command will output text to <*filename*> instead of displaying it:

   batwv –audit > <*filename*>

3. Send the audit code by email to uk.support.qas@experian.com.
4. Experian Data Quality will process the audit code and will return a counter update code to you by email.
5. Call the **QABatchWV_ApplyUpdateCode** function and enter the update code received from Experian Data Quality in order to populate the counters file with specified post-pay meters for each supported Suppression dataset.

   As this is a one-off action, it can be performed as part of your initial API integration, or by using the C test harness. However, the code will be supplied in case you want to integrate the meter creation yourself at a later date; for example, if your existing counters file becomes corrupted or if you want to add extra Suppression DataPlus sets at a later date.

   To create a meter, use the following switches when starting the test harness:

   batwv –apply <*update code*>

   batwv –applyfile <*filename*>

   where <*filename*> is the name of a file containing the counter update code as supplied by Experian Data Quality.

   In the above cases the harness will not start as normal, but will attempt to apply the update (printing any errors to the standard output channel).

6. The QAS Batch API product is then unlocked for use and begins to count the clicks that will be charged for matches against Suppression data.

> You should send your audit code to uk.support.qas@experian.com on a monthly basis; you will then be invoiced according to the number of clicks used.

Managing Suppression Costs

Though payment is submitted to Experian Data Quality on a month by month basis, often it is useful to have more specific payment information available. For example, it may be desirable to check how many Suppression clicks have been used since the last time an audit code was sent to Experian Data Quality, or how much one specific Suppression run would cost. The following three functions can be used to determine more accurate payment information:

- To-Date Billing
- Temporary Counters
- Estimate Mode

To-Date Billing

To-Date Billing allows you to check the expenditure of an installation of QAS Batch API without having to send the audit code to Experian Data Quality. This has two advantages: that expenditure can be checked throughout the month, as opposed to only at the end of the month, and that the information can be derived locally and therefore immediately.

To use To-Date Billing, follow these steps:

1. Call the **QABatchWV_GetAuditCode** function to return the current audit code.
2. Call **QABatchWV_CompareAuditCode**, using the current audit code and the previously generated audit code (this can be the audit code previously sent to Experian Data Quality, or one created since).

   This function will output an XML report into the specified buffer.

The XML report contains information on click usage and Suppression, which will allow a cost estimation to be calculated.

This report does not contain statistics from your QAS Batch API clean.

Temporary Counters

If accurate payment information is required during the cleaning and suppression of a smaller part of a database, temporary counters can be created. Reports can then be run, incorporating only the cleaning and payment information that has been accumulated since the creation of the counters.

Once created, a counter is given a handle and can then be used to return an XML report at any time, as many times as needed.

To create and use temporary counters, follow these steps:

1. Call the **QABatchWV_CounterOpen** function.
2. Run QAS Batch as normal. Suppression is optional.
3. Call the **QABatchWV_CounterReportLength** function and create a buffer of the correct size for the XML report.
4. Call the **QABatchWV_CounterReport** function. The XML report will be returned into the specified buffer.
5. Call the **QABatchWV_CounterClose** function when you have created all the reports needed.

Once the counter has been closed, all the information that has not been stored in an XML report will be lost. All reports required must be generated before the counter is closed.

Estimate Mode

If it is required to determine the estimated price of a Suppression run without being committed to paying for the clicks, an instance of the API can be run in Estimate mode.

While in Estimate mode, Counters can be opened and closed, QAS Batch API can be run as normal, and reports can be created. However no address will be cleaned, all items will return the match code A0000000000000000000, and the formatted line count will be zero.

The QAS Batch API statistics will be populated with the match details as if the run has taken place, and the number of suppressed addresses and cleaning information will be displayed.

To run the instance of the API in Estimate Mode, follow these steps:

1. Call the function **QABatchWV_RunMode**, passing a non-zero value into the parameter *viEstimateMode*.
2. Call the **QABatchWV_CounterOpen** function.
3. Run QAS Batch as normal.
4. Call the **QABatchWV_CounterReportLength** function and create a buffer of the correct size for the XML report.
5. Call the **QABatchWV_CounterReport** function. The XML report will be returned into the specified buffer.
6. Call the **QABatchWV_CounterClose** function when you have created all the reports needed.
7. Call the function **QABatchWV_RunMode**, passing zero into the *viEstimateMode* parameter.

   Estimate Mode will now finish. Information in any unclosed counters will be lost.

The QAS Batch API statistics and counter usage are **estimates only**. Please be aware that caching and threading changes could make a difference to the actual run.

Troubleshooting

For information about possible errors that may be returned when you use QAS Batch API with Suppression data, see -4593 to -4597 of the Error Code Listing.

For more information about Suppression datasets, including information about how to configure Suppression and how to use Suppression DataPlus, refer to the Suppression Additional Data Guide supplied with your data.

Appendix E: Names Matching Tolerance Levels

> This information is only relevant when using GBR data.
>
> If you are using AUS Suppression data, refer to your Suppression Additional Data Guide for more specific information about AUS names matching.

The `NamesTolerance` setting allows you to configure how QAS Batch matches names records. The available settings for names matching are described below:

- "Exact" on page 201
- "Intermediate" on page 202
- "Standard" on page 203
- "Relaxed" on page 203

For information on how to configure Names Matching Tolerance, see "NamesTolerance" on page 166.

Exact

In order to achieve a match under the Exact setting, an input name must achieve the following criteria:

- There must be no spelling differences between the input name and the data.

Input	Data	Match?
Jonathan Smith	Jonathan Smith	Yes
Jonathan Smith	Jonathon Smith	No

- Forename aliases will not be matched.

Input	Data	Match?
William Smith	William Smith	Yes
Bill Smith	William Smith	No

- A gender-specific title in the input name must match the gender of the title of the name in the data. Gender-neutral titles, such as 'Dr' may match to titles of either gender.

  However, gender-specific forenames will still factor into whether a match is made.

Input	Data	Match?
Mr Alex Smith	Mr Alex Smith	Yes
Mr Alex Smith	Mrs Alex Smith	No
Dr Alex Smith	Mrs Alex Smith	Yes
Dr Alexander Smith	Mrs Alex Smith	No

- A full forename in the input will match to a forename initial in the data. However, a forename initial will not match to a full name in the data.

Input	Data	Match?
William Smith	W Smith	Yes
W Smith	William Smith	No

- All name elements in the data must also be present in the input, and must occur in the same order.

Input	Data	Match?
William Gerald Smith	William Smith	Yes
William Gerald Smith	William Gerald Tony Smith	Yes
William Gerald Smith	Gerald Smith	No
William Gerald Smith	Gerald William Smith	No

- If the input contains only a forename and surname, matches will still be allowed, even if the matching record in the data also contains a middle name or middle initial.

Input	Data	Match?
William Smith	William Gerald Smith	Yes
William Smith	William G Smith	Yes

- If the input contains a middle name, matches will still be allowed, even if the matching record in the data also contains an additional middle name or middle initial.

Input	Data	Match?
William Edward Smith	William Edward Gerald Smith	Yes
William Gerald Smith	William Edward Gerald Smith	Yes
William Edward Smith	William Gerald Tony Smith	No

Intermediate

The criteria for achieving a name match under the Intermediate setting are the same as for the Exact setting, with the following exception:

- It is not necessary for all the name elements in the data to also be present in the input to achieve a match. However all matching name elements must still be in the same order.

Input	Data	Match?
William Gerald Smith	Gerald Smith	Yes
William Gerald Smith	Gerald William Smith	No

Standard

The criteria for achieving a name match under the Standard setting are the same as for the Intermediate setting, with the following exceptions:

- Full forenames in the input can match to forename initials in the data.

Input	Data	Match?
W Smith	William Smith	Yes
William Smith	W Smith	Yes

- Matches may still be allowed, even with minor spelling differences between the input name and data.

Input	Data	Match?
Jonathan Smith	Jonathan Smith	Yes
Jonathan Smith	Jonathon Smith	Yes

- Alias matches are allowed.

Input	Data	Match?
William Smith	William Smith	Yes
Bill Smith	William Smith	Yes

Relaxed

The criteria for achieving a name match under the Relaxed setting are the same as for the Standard setting, with the following exception:

- A match can be achieved if the forename and middle names occur in a different order in the input compared to the data.

Input	Data	Match?
William Gerald Smith	Gerald William Smith	Yes

Appendix F: Delivery Point Validation

This information is only relevant when using USA data.

Delivery Point Validation (DPV) has been developed by the United States Postal Service (USPS) to help you validate the accuracy of your address information. It enables you to determine whether the actual address exists - all the way down to apartment or suite information.

DPV Seed Addresses

The DPV functionality has in-built protection against the illegal creation of verified address lists. This is ensured by having a concept of 'seed' addresses. These are non-existent addresses that if searched upon, will deactivate the DPV functionality within QAS Batch API.

If you encounter a seed address, you need to obtain an unlock code to re-enable DPV functionality. You can then use the DPV unlock tool to unlock the functionality. The unlock tool is an executable file named either dpv.exe (for 32-bit installations) or dpv64.exe (for 64-bit installations).

QAS Batch will warn you if your DPV functionality is locked and you try to perform certified cleaning on your data.

Encountering A Seed Address

If a seed address is encountered, a lock code is reported, together with instructions on the re-enabling process.

The lock code is written to a file called DPVStatus.txt within the program directory. Where it is not possible to write to the DPVStatus.txt file, no error will be returned – the file will not be updated on the assumption that another instance of QAS Batch API is attempting to write to it simultaneously.

Experian Data Quality is contracted to report this information on behalf of the USPS:

> "DPV processing was terminated due to the detection of what is determined to be an artificially created address. No address beyond this point has been DPV validated. In accordance with the Agreement between the USPS and Experian Data Quality, DPV shall be used to validate legitimately obtained addresses only, and shall not be used for the purpose of artificially creating address lists. The written agreement between Experian Data Quality and the End User also includes this same restriction against using DPV to artificially create address lists. Continuing use of DPV requires compliance with all terms of the Agreement. If you believe this address was identified in error, please contact Experian Data Quality."

When a seed address is encountered and the DPV system disabled, you will need to submit the lock code to Experian Data Quality Technical Support. You will be provided with a corresponding alphanumeric unlocking key. This key, when supplied back to QAS Batch API, will allow DPV functionality to be re-enabled.

QAS Batch API is supplied with a DPV Unlock Utility which can handle the unlocking process for you. The Unlock Utility is provided for Windows users only, and is called dpv.exe (for 32-bit installations) or dpv64.exe (for 64-bit installations). If you are using UNIX or do not want to use the Unlock Utility, you can manually supply the following information by contacting Experian Data Quality Technical Support:

- The DPV lock code generated by QAS Batch API;
- The date that the seed address was hit;
- The full seed address that was hit;
- Your name, company name, full address and telephone number.

The lock code is reported when a seed address is encountered and can also be obtained using the **QABatchWV_DPVGetCode** function (see page 84). The remaining information can be obtained using the **QABatchWV_DPVGetInfo** function (see page 87) provided the DPVStatus.txt file has successfully been written.

Entering The Unlock key

The DPV Unlock Utility can apply the unlock code for you, or you can use the **QABatchWV_DPVSetKey** function (see page 89).

The unlock key provided to you by Experian Data Quality Technical Support will re-enable DPV functionality only once. USPS reserves the right to require Experian Data Quality to suspend a user's ability to perform DPV processing when multiple incidents of artificial seed address detection occur.

If you have problems applying the unlock code, contact Experian Data Quality Technical Support by visiting: http://support.qas.com/contact.

Multithreading Considerations

QAS Batch API can support up to 8 simultaneous threads (providing you have sufficient RAM available). QAS Batch API internally synchronises calls into it to allow the API to support multithreaded integrations using a greater number of threads.

DPV functionality affects the integration of the API in several ways.

First, if a search thread causes the DPV system to lock itself following a search involving a "seed address", no further delivery point validation will take place for any subsequent searches, regardless of the instance or search handle used.

Any call using a QAS Batch API instance handle that has a country base including USA is able to interrogate the state of the DPV component by using **QABatchWV_DPVState** (see page 90) and, where it is locked, will be able to obtain the DPV lock code by using **QABatchWV_DPVGetCode** (see page 84) and set a corresponding DPV unlock key with **QABatchWV_DPVSetKey** (see page 89). However, the ability of one QAS Batch API instance thread to unlock the DPV component that another QAS Batch API search related thread may be using means that synchronisation related complexities must be taken into account.

For multithreaded integrations that require DPV functionality:

- Ideally, a dedicated top level thread should be used to perform DPV code/key interrogation/unlocking to prevent multiple unlock attempts from occurring.

- Other threads may happen to be synchronised to search ahead of an attempt to unlock the DPV component (i.e. at the low level within QAS Batch API), so searches should be re-attempted where the "DPV disabled" bit is set in the extended dataset-specific information component of QAS Batch API's return code. See your Data Guide for further information on extended dataset-specific information bits.

Appendix G: Generic Information Bits

The generic information bits provide detailed information on how well an address match conforms to the QAS Batch API matching rules. See "Matching Rules" on page 38 for more information.

The numbers below are added together to make up the hexadecimal match code. For example, 000A0000 would be the result of an error in the input street name (00080000) and an added or changed street descriptor (00020000).

A brief explanation of each of the generic information bits is provided below.

10000000

The elements in the input address were not in the expected order.

For example, in the address 'Top Flat, 4 Baker Street, Dyfed, Aberystwyth, SY23 2BJ', the county (Dyfed) should appear after the town (Aberystwyth). In this example, infobit 00000020 is also returned as a result of the address cleaning process.

20000000

Preferred matching rules were not satisfied, and the match will be marked with, at best, intermediate confidence.

40000000

The address has been marked with, at best, intermediate confidence because the close matching rules were not satisfied.

80000000

Conditional formatting has taken place. Some address elements in the specified output format were not present in the matched address record and have been replaced with equivalent address elements. For more information on how QAS Batch API can apply conditional formatting when using a particular dataset, and which address elements may be affected, consult the relevant Data Guide.

01000000

Extra numbers were found in the address. An example might be '4 Granary 8 Road, Devizes, Wiltshire, SN10 3DP'. A full match was achieved (4 Granary Road, Devizes, Wiltshire, SN10 3DP), but the additional number(s) (i.e. '8' in the above example) may reduce the confidence level to intermediate.

02000000

Additional text between a number and its expected adjacent component has been found, for example between a property number and a street name. The confidence level has been reduced to intermediate.

04000000

No place element (e.g. a locality in Australia) was found in the address, so the confidence level may be reduced.

08000000

The item associated with a number is missing. For example, the British address '4, South Marston, Swindon, SN3 4XX' is missing the street name 'Ash Gardens' after the building number. In this example, infobits 10000000 and 2000000 are also returned due to the absence of a street name. In addition, further infobits are returned as a result of the address cleaning process (i.e. adding the street name and reformatting the address).

00100000

One or more essential matching rules were not satisfied, so the match confidence has been reduced to low.

00200000

A timeout has occurred, and the address was not matched. The default timeout period is 5 seconds; you can change this with the configuration setting `BatchTimeout` in qaworld.ini.

00400000

The input address begins with unmatched text before a premise number. For example, the input address 'Village Arcade, 5 Hillcrest Road, Pennant Hill, NSW, 2120' contains more information than the official version, which does not contain the 'Village Arcade' element before the premise number, '5'. See also 00004000, which is similar but occurs even if no premise number is present.

00800000

A leading number was unused in the input address. For example, the Level 5 element in the address 'L 5, 2/6 The Bollard, Corlette, NSW' is not found in the official address.

00010000

There was ambiguity in the supplied range of the input address. For example, the address '26-30 Delhi Street, Adelaide SA 5000' has an ambiguous range because there is a 26, 28 and 30 Delhi Street and the input address cannot be matched to a specific property.

00020000

A street descriptor has been added or has been changed. For example, with '10 Railway Road, Serviceton, Vic', the correct descriptor 'Street', is returned instead of 'Road'.

00040000

Additional text in the input address was too significant to ignore. For example, the French address '18 boulevard Voltaire, 75011 Paris CEDEX 11' contains unmatched, but significant, information 'CEDEX 11'. This returns an intermediate confidence level.

00080000

There was an error in the input street name that QAS Batch API has amended.

00001000

There was an error in an input place name (for example, an Australian locality) that QAS Batch API has amended.

00002000

QAS Batch API has added or changed a key premise number or range compared with the input address, such as a building number in Australia data where a single number matched to a range, or organisation names in the France data.

00004000

The input address begins with unmatched text. For example, the input address 'Rose Cottage, Hillcrest Road, Pennant Hill, NSW, 2120' contains more information than the official version, which does not contain the 'Rose Cottage' element. See also 00400000, which is similar but occurs only when the unmatched text occurs directly before a premise number.

00008000

A name was used to secure an address match.

00000100

The address line settings of the currently configured layout are of an insufficient width to contain each address element. Widen the address lines to ensure that the address elements are not truncated.

00000200

Complete address element(s) are unable to fit on the address line. Widen the address lines to ensure that all of the address elements are visible.

00000400

QAS Batch API failed to generate one or more non-address items. It is likely that the DataPlus set could not be opened.

00000800

When in enhanced cleaning mode, QAS Batch API cannot fill the unmatched address elements back into the database. To resolve this, widen the address lines or add additional lines.

00000010

Postal address elements have been moved to the right or downwards to allow unmatched elements to be incorporated in an enhanced address.

00000020

QAS Batch API has determined that the supplied address has been non-trivially cleaned. This means that spelling may have been corrected, capitalisation changed, or the input address elements could have been reformatted in some way. Note that quotes and spaces are ignored during QAS Batch API's comparison process.

00000040

Key input address elements were judged effectively correct as supplied, although the output address's representation or formatting may differ (for example, address elements may have been expanded or abbreviated, capitalisation changes made, etc).

00000080

If the user defines `InputLineCount` (including blank) and the input line count does not match the number of lines defined in the input search string (allowing for quotes), this bit will be set. This bit does not affect match confidence.

00000001

The final tests on the address have failed against the strict matching rules; hence the match confidence level is reduced to, at best, intermediate confidence.

00000002

QAS Batch API has found a premise level partial address match.

00000004

QAS Batch API has found a street level partial address match.

00000008

QAS Batch API has found a place level partial address match.

Appendix H: Integration Example

This information is only relevant when using GBR or AUS data with additional Suppression data.

Below is an example of integrating the reporting functions into an instance of the API. For more general examples of using the API, see "Pseudocode Example Of QAS Batch API" on page 49. This integration example assumes you have knowledge of the API functions. For more information on these, see Suppression-Specific Functions.

Integrating XML reports

The conventions within the pseudocode are below:

Convention	Meaning
/ Comment */*	Italic text between asterisks and forward slashes denotes a comment.
`[`**`QABatchWV_DataSetInfo`**`]`	The functions which relate to each part of the pseudocode appear in bold type on the right hand side of the page.
`[`**`QABatchWV_Open`**` (Close)]`	Some API functions are 'paired', i.e. when a function is called, its pair must also be called at some point. When a paired function is used in the pseudocode, its pair appears in brackets directly after the function name.

Pseudocode Listing

```
/* Before calling any function in the QAS Batch API, you must initialise it with
QABatchWV_Startup. Once initialised, you must open an instance of the API
*/
Initialise API                          [QABatchWV_Startup
                                        (Shutdown)]
  Initialise an instance of the API     [QABatchWV_Open
                                        (Close)]
    Open a counter handle               [QABatchWV_
                                        CounterOpen
                                        (CounterClose)]
/* Once the counter handle is open, addresses may be cleaned in the normal
way. For more information on how to do this, see "QAS Batch API Functions"
on page 54, and for examples see "Pseudocode Example Of QAS Batch API"
on page 49. Each address cleaned will be added to the open report, as well
as information about the number of clicks used. When address cleaning is
completed, you can output the information stored in the open report. */
      Extract report using counter      [QABatchWV_
      handle                            CounterReport]
/* A XML report is returned into the specified buffer as a string. You can clean
further addresses and extract the report as many times as required. Once
completed, close the counter instance and the instance of the API */
    Close the counter handle            [QABatchWV_
                                        CounterClose]
  Close instance of API                 [QABatchWV_Close]
Shutdown API                            [QABatchWV_Shutdown]
```

While the instance of the API is open, you can use the function **QABatchWV_ CounterReportLength** at any time to return the current size in bytes of the XML report that would be returned by **QABatchWV_CounterReports**

Viewing The Report

To view XML reports generated by **QABatchWV_CounterReport**:

1. Write the contents of the **QABatchWV_CounterReport** output buffer to an .xml disk file.
2. A stylesheet is provided with QAS Batch to convert the XML into a browser based report.

To generate the report ensure that the XSL stylesheet and the flash components (supplied with QAS Batch) have been copied into the same directory as the XML file.

3. The report will be displayed when the XML file is opened in your Internet Browser.

Glossary Of Terms

Absolute Contacts

Absolute Contacts Tracking data is a subset of the *Absolute Movers* Suppression Set which also contains contacts' forwarding addresses.

Absolute Movers

Absolute Movers data contains details of individuals who have moved from an address.

Additional Dataset

Additional datasets are available with some *datasets* to enhance the data. They cannot be used without the base dataset they are associated with.

For example, the Names additional dataset is available for USA and GBR data only.

Address Elements

The fields that comprise an address. Each dataset contains a set of address elements which are specific to that country.

For example, in United Kingdom data, these fields may include building number, thoroughfare, town, and postcode.

Address Layout

The format of an address, arranged with the *Address Elements* in an order specific to the convention of the country.

Address Match Code

See *Match Code*.

Audit Code

A text-based code located in the counters file on the disk where QAS Batch API is installed. It should be extracted and sent to Experian Data Quality on a monthly basis, in order that you can be invoiced for the number of clicks used. See "Managing Suppression Costs" on page 197 for more information.

Audit Log

A log created at installation where the API regularly updates all current meter values. This allows you to check your meter usage against your billing history.

Post-pay meters (for use with QAS Batch API with Suppression) will count down, rather than counting up. See "Managing Suppression Costs" on page 197 for more information.

Australia Mortality Data

Mortality data contains details of people who have passed away. Depending on the options selected, you can match the records in your data to a household, to a surname or to a named individual.

Clicks

A click is a single count against a *meter*. You may be charged for a permanent, one-off (temporary), tracking or dual click when an address record is matched against a Suppression dataset.

If an address record matches more than one GBR Suppression dataset, the order in which matches should be used (and therefore your costs) depends on the options set in the Suppression Hierarchy.

Configuration Files

Both Windows and UNIX users can use the configuration files (qawserve.ini and qaworld.ini, as well as any layout-specific configuration files) to configure the QAS server, to specify which *Datasets* are installed, to configure search options and results settings and to set output address formats.

DataPlus

Extra information that can be returned with each matched address.

Examples of DataPlus information depend on the dataset being used, but examples for United States data could include Country Code or a Postnet barcode.

Dataset

A collection of proprietary data files, containing address information or address-enhancing information, that are shipped to customers on a data CD-ROM or DVD-ROM and are also available via Electronic Updates.

Dataset Identifier

The three character ISO code that uniquely identifies a *dataset*. For example, the dataset identifier for United Kingdom data is 'GBR'.

Data Caching

Caching stores data in memory for future reference, instead of reading it from a hard disk or across a network as required. In general, the less data is accessed over a network or from a disk, the faster QAS Batch API will run (subject to your computer having sufficient free physical memory).

Data Expiry

Each dataset has an expiry period.

The expiry period is the number of days remaining until the data is no longer valid.

Data Guide

A reference guide that is supplied with each *dataset* that you purchase. It provides dataset-specific information and search tips for each dataset.

Data Mapping

A combination of a dataset and additional datasets which is used as a *dataset*. Each data mapping has a unique *identifier* and name.

Data Mapping Identifier

See *Identifier*.

DPV

DPV (Delivery Point Validation) is a USPS (United States Postal Service) system designed to validate the accuracy of your USA address information. It enables you to determine whether the address exists at a sub-premise level. DPV may not be used to create address lists artificially. DPV uses *Seed Addresses* to detect when address records appear to be the result of artificial manufacture.

Goneaway

GBR Goneaway Suppression File (GSF) additional suppression data contains names and addresses of people who have moved house.

High confidence

QAS Batch API returns a High confidence match when it is sure that the output address matches the input data. This happens when an input address is fully accurate, or when incomplete address data is detailed enough (for example, exact property number, street and locality) to have the remaining address details appended.

Identifier

The identifier is a unique three character alpha-numeric code that defines a *data mapping*.

Ini File

A common name for the *Configuration Files* qawserve.ini, qaworld.ini and qalicn.ini, as well as any layout-specific configuration files you might create.

Input Fields

The fields in your source database containing the address information to be cleaned.

Intermediate Confidence

A confidence level of Intermediate is returned when QAS Batch API is reasonably sure that it has found the right match. This might occur if the input address is misspelled slightly or if the address contains extra numbers.

Hullenbergweg, 1-3, 1101 BW, Amsterdam Zuidoost

In the Netherlands address example above, QAS Batch API finds an extra number in the address (3). However, with a full postcode available to search on, QAS Batch API is able to find the correct address. Only the extra number in the address prevents a High confidence match.

ISO Code

International Standards Organisation code. See *Dataset Identifier*.

Licences

You receive a licence key for each combination of data and product that you purchase. Failure to enter a valid licence key means that the product will be unable to use data.

When you have an evaluation of an Experian Data Quality product or data, evaluation licence keys are provided that set time limits on the usage of the data. To continue using the product and data after these time limits have been reached, you must purchase a full licence.

In addition, you can purchase a metered licence for use with QAS Batch API, to control how address lookups, address matches and search results are charged.

Low Confidence

QAS Batch API sets the confidence level to low if it finds a match which differs considerably from the input address.

> Gehilweg 16, 1101 CD, Amsterdam Zuidoost

In the Netherlands address example above, QAS Batch API cannot find a Gehilweg in Amsterdam, and returns another match (Hogehilweg 16) instead. As the returned address is significantly different from the input address, QAS Batch API is not confident that this is the right match.

Mailing Preference Service

GBR Mailing Preference Service suppression data contains details of individuals who have opted not to receive unsolicited mail.

Match Code

When QAS Batch API cleans an address, any processing which takes place and any changes which are made to the address are recorded in a match code.

Meter

A meter is used to measure usage of *clicks*. It contains the total number of clicks used (by dataset and click type) since it was activated. There are four types of click:

- Permanent
- One-off

- Dual
- *Tracking*

See "Managing Suppression Costs" on page 197 for more information about how to pay for Suppression data via a meter.

MOR

See "Mortality Data" below.

Mortality Data

Data which contains details of people who are recently deceased. Depending on the options selected, you can match the records in your data to a household, to a surname, or to a named individual.

Examples of this type of data include the GBR Mortascreen, GBR Mortality Suppressions and AUS Mortality suppression datasets.

MSS

See "Mortality Data" above.

NCOA Suppress

GBR National Change of Address Suppression data contains details of individuals who have moved from an address.

NCOA Update

GBR National Change of Address Update tracking suppression data contains names of customers and the details of addresses that mail has been redirected to and from.

Output Fields

The fields in your database as they will appear after cleaning.

Sample Files

A number of files are shipped with QAS Batch API, containing sample addresses for use with QAS data. You may find it useful to clean one of these databases before you use QAS Batch API to clean your own data.

Seed Addresses

Seed addresses are used by the *DPV* system to protect against the illegal creation of verified address lists. If one of these non-existent 'seed' addresses is searched upon, the DPV functionality within QAS Batch will be deactivated. For more information see "Appendix F: Delivery Point Validation" on page 205.

Suppression

Suppression data contains additional information associated with the occupant(s) of an address. Specifically, you can clean your records against Suppression data and then return relevant Suppression DataPlus information for any matching addresses in your database. This makes it possible to see easily any addresses which may not be useful to your business.

Telephone Preference Service

The GBR Telephone Preference Service suppression dataset contains details of individuals who have opted not to receive unsolicited telephone calls.

The Bereavement Register

GBR Bereavement Register suppression data contains names and addresses of deceased individuals.

Tracking

A Tracking dataset is similar to a Suppression dataset, but the forwarding address of the property replaces the original record instead of it being suppressed.

Tracking Mode

When using NCOA Update or Absolute Contacts Tracking mode, users will be charged one tracking click for returning a forwarding address. Furthermore, if the forwarding address matches against other Suppression datasets, these matches will be charged according to the criteria for permanent and one-off clicks. See the United Kingdom With Suppression Additional Data Guide for more information about NCOA Update and Absolute Contacts data.

Update Code

Code provided by Experian Data Quality on receipt of initial audit code. When applied to an API installation, it initialises all post-pay Suppression counters.

Index